Single Regression Trees: Estimation of Claims Frequency on Car Insurance data

In this markdown, we will estimate the claims frequency of car drivers using a Regression Tree with the R package {rpart}. The data come from the chapter one of the book "Predictive Modelling Applications in Actuarial Science, Vol.2" (E. Frees & al.). The website is at the following address: https://instruction.bus.wisc.edu/jfrees/jfreesbooks/PredictiveModelingVol1/glm/v2-chapter-1.html

Data have been already explored in a previous study (cf. EDA for Insurance) stored in another repository where a description of the fields is also available.

Introduction

Decision tree learning is one among many other predictive modelling approaches. They have the advantage to be easily interpreted and work for both classification and regression tasks. However, single tree model typically lack in predictive performance comparing to ensemble methods like Random Forest or GBM. The aim of this study is purely educative and stands as an introduction to explore the other Tree-based algorithms.

In a nutshell, Classification And Regression Tree algorithm, aka CART, developed by Breiman et al. in 1984 works by partitioning the feature space into a number of smaller (non-overlapping) regions with similar response values using a set of splitting rules. The goal is at each threshold, the minimum sum squared of residuals between the observed value and the predicted is minimal.

We will add more explanation along the way while fitting the model.

1. Loading the data

```
load("data/all-data.RData")
```

```
str(dta)
```

```
## 'data.frame':    40760 obs. of  46 variables:
##  $ row.id        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ year          : Factor w/ 4 levels "2013","2010",..: 2 2 2 2 2 2 3 2 3 2 ...
##  $ exposure      : num  1 1 1 0.08 1 0.08 1 1 0.08 1 ...
##  $ nb.rb         : Factor w/ 2 levels "NB","RB": 2 1 2 2 2 2 2 2 2 2 ...
##  $ driver.age    : num  63 33 68 68 68 68 53 68 68 65 ...
##  $ drv.age       : Factor w/ 74 levels "38","18","19",..: 46 17 51 51 51 51 36 51 51 48 ...
##  $ driver.gender : Factor w/ 2 levels "Male","Female": 1 1 1 1 1 1 1 1 1 1 ...
##  $ marital.status: Factor w/ 4 levels "Married","Divorced",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ yrs.licensed  : num  5 1 2 2 2 2 5 2 2 2 ...
##  $ yrs.lic       : Factor w/ 8 levels "1","2","3","4",..: 5 1 2 2 2 2 5 2 2 2 ...
##  $ ncd.level     : Factor w/ 6 levels "1","2","3","4",..: 6 5 4 4 3 3 4 3 3 3 ...
```

```
##  $ region        : Factor w/ 38 levels "17","1","10",..: 23 32 27 27 23 23 23 23 23 14 ...
##  $ body.code     : Factor w/ 8 levels "A","B","C","D",..: 1 2 3 3 4 4 5 4 4 4 ...
##  $ vehicle.age   : num  3 3 2 2 1 1 3 1 1 5 ...
##  $ veh.age       : Factor w/ 15 levels "1","0","10","11",..: 9 9 8 8 1 1 9 1 1 11 ...
##  $ vehicle.value : num  21.4 17.1 17.3 17.3 25 ...
##  $ seats         : Factor w/ 5 levels "5","2","3","4",..: 1 3 1 1 2 2 2 2 2 2 ...
##  $ ccm           : num  1248 2476 1948 1948 1461 ...
##  $ hp            : num  70 94 90 90 85 85 70 85 85 65 ...
##  $ weight        : num  1285 1670 1760 1760 1130 ...
##  $ length        : num  4.32 4.79 4.91 4.91 4.04 ...
##  $ width         : num  1.68 1.74 1.81 1.81 1.67 ...
##  $ height        : num  1.8 1.97 1.75 1.75 1.82 ...
##  $ fuel.type     : Factor w/ 3 levels "Diesel","Gasoline",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ prior.claims  : num  0 0 0 0 0 0 4 0 0 0 ...
##  $ clm.count     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ clm.incurred  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ rnd           : num  0.9116 0.7073 0.0528 0.9932 0.6116 ...
##  $ train         : logi  TRUE TRUE TRUE FALSE FALSE TRUE ...
##  $ valid         : logi  FALSE FALSE FALSE TRUE TRUE FALSE ...
##  $ len.dm        : num  43.2 48 49.1 49.1 40.4 ...
##  $ hei.dm        : num  18 19.7 17.5 17.5 18.2 ...
##  $ wid.dm        : num  16.8 17.4 18.1 18.1 16.7 ...
##  $ drv.age.gr1   : Factor w/ 7 levels "44-50","18-33",..: 6 2 6 6 6 6 5 6 6 6 ...
##  $ drv.age.gr2   : Factor w/ 10 levels "38-42","18-22",..: 10 5 10 10 10 10 8 10 10 10 ...
##  $ region.g1     : Factor w/ 9 levels "R0","R1","R2",..: 4 5 7 7 4 4 4 4 4 1 ...
##  $ sev           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ veh.val.q15   : num  6.45 2.05 2.3 2.3 10 10 3.1 10 10 0 ...
##  $ veh.val.q35   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ driver.age.q35: num  28 0 33 33 33 33 18 33 33 30 ...
##  $ driver.age.q49: num  14 0 19 19 19 19 4 19 19 16 ...
##  $ driver.age.q59: num  4 0 9 9 9 9 0 9 9 6 ...
##  $ hp.cat        : Factor w/ 11 levels "(70,75]","(0,65]",..: 3 6 5 5 4 4 3 4 4 2 ...
##  $ efq           : num  0.05292 0.09897 0.1176 0.00941 0.11141 ...
##  $ esv           : num  584 961 610 610 580 ...
##  $ epp           : num  30.92 95.08 71.76 5.74 64.63 ...
```

2. Implementation

We can fit a regression using {rpart}. As we want to predict a frequency, we need to specify using the poisson deviance by calling the method "poisson" and setting the response as a two-column matrix including the exposure. For this first example, we take all the predictor available and leave all the hyperparameters at their default level.

```
# Select the train set
# We need to select only claims > 0 to predict severity
# The dataset is considerably reduced in size.
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
dta_trn <- dta %>% filter(train == TRUE)
dta_tst <- dta %>% filter(train == FALSE)
dim(dta)
```

```
## [1] 40760    46
```

```
dim(dta_trn)
```

```
## [1] 24495    46
```

```
dim(dta_tst)
```

```
## [1] 16265    46
```

```
# Remove the predictors that are not in use for the analysis
dta_trn <- subset(dta, select = -c(row.id, drv.age, clm.incurred, hp, length, width, height, sev, veh.va
                                   driver.age.q35, driver.age.q49, driver.age.q59,
                                   efq, esv, epp
                                   ))
```

Run of a single tree with {rpart}

```
library(rpart)       # direct engine for decision tree application
```

```
## Warning: package 'rpart' was built under R version 4.1.3
```

```
library(caret)       # meta engine for decision tree application
```

```
## Warning: package 'caret' was built under R version 4.1.1

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
fit <- rpart(formula =
                cbind(exposure,clm.count) ~ . ,
                data = dta_trn,
                subset = train,
                method = 'poisson'
                )
print(fit)
```

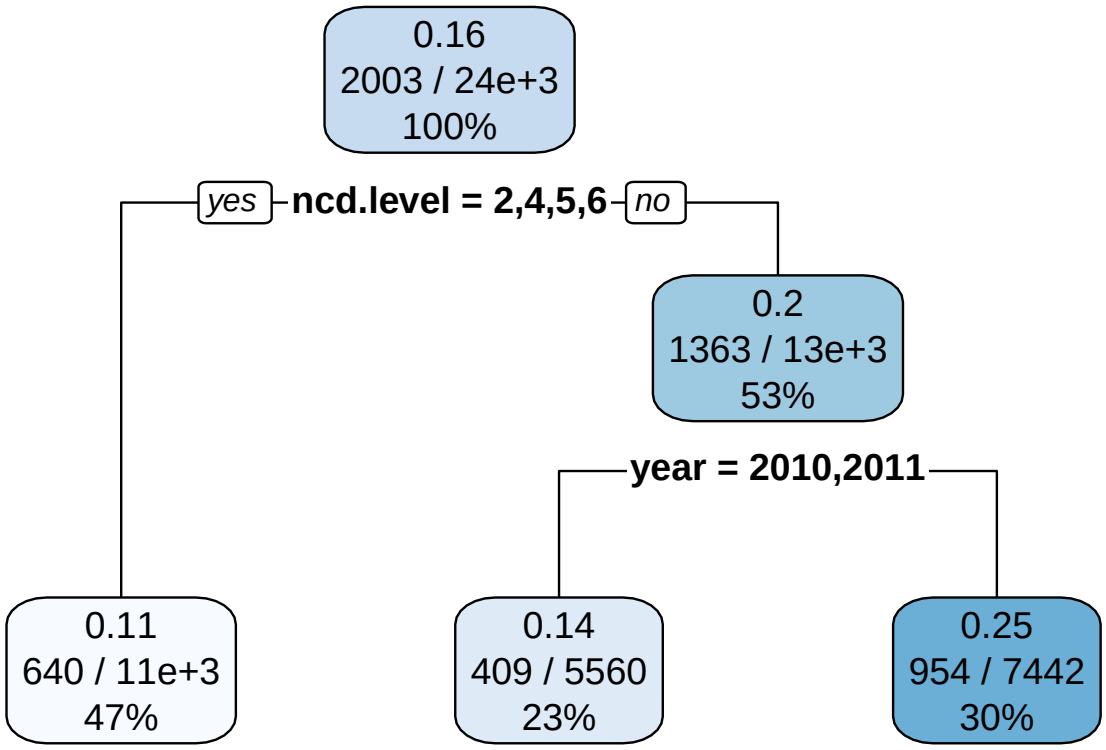
```
## n= 24495
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 24495 9848.918 0.1606008
##   2) ncd.level=2,4,5,6 11493 3598.377 0.1108765 *
##   3) ncd.level=1,3 13002 6079.696 0.2034859
##     6) year=2010,2011 5560 2112.325 0.1412178 *
##     7) year=2013,2012 7442 3866.075 0.2509145 *
```

The extract above explains the steps of the splits. Here we start with 24495 observations at the root node, and the first variable that optimize a reduction in deviance is the 'ncd.level', "No-Claim Discount", which is a discount applied on your Comprehensive Car Insurance premium that increases each year as long as you don't make an at-fault claim. Fortunately, we can get a visual representation of the tree with the package {rpart.plot}:

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.1.1
```

```
rpart.plot(fit, cex = 1.2)
```



3. Interpretation For each node, the first line in the box indicates the overall claim frequency of the sample. Here, it shows an overall 16% claims, confirmed by the computation below:

```
emp_freq = sum(dta$clm.count) / sum(dta$exposure)
print(emp_freq)
```

```
## [1] 0.1649933
```

The second line, shows the ratio between the number of claims and the sum of exposure. The third line is the exposure-proportion, where the sum of the split nodes always equals 100%.

The root node is split into 2 groups: a population having a No-Claim Discount at 2,4,5 and 6, representing 47% of the population and showing a claim frequency of 11%. The population that have the 'ncd' at 1 and 3 is shown on the right, representing 53% of the population with a claim rate of 20%. For this population, another split is done on the "year of analysis" variable. The observation from year of analysis between 2010 and 2011 represent 23% of the population with a claim rate at 14%, while the others show 25%, which is a big difference.

4. Hyperparameters Tuning

We start by fitting a simple tree using our train set, taking the predictors that look to be good candidates for our prediction.

In the formula, 'control' is an argument that provide a list of hyperparameter value. The parameter 'maxdepth' represents the maximum depth of the tree, set up at 3. 'cp' is the complexity parameter,

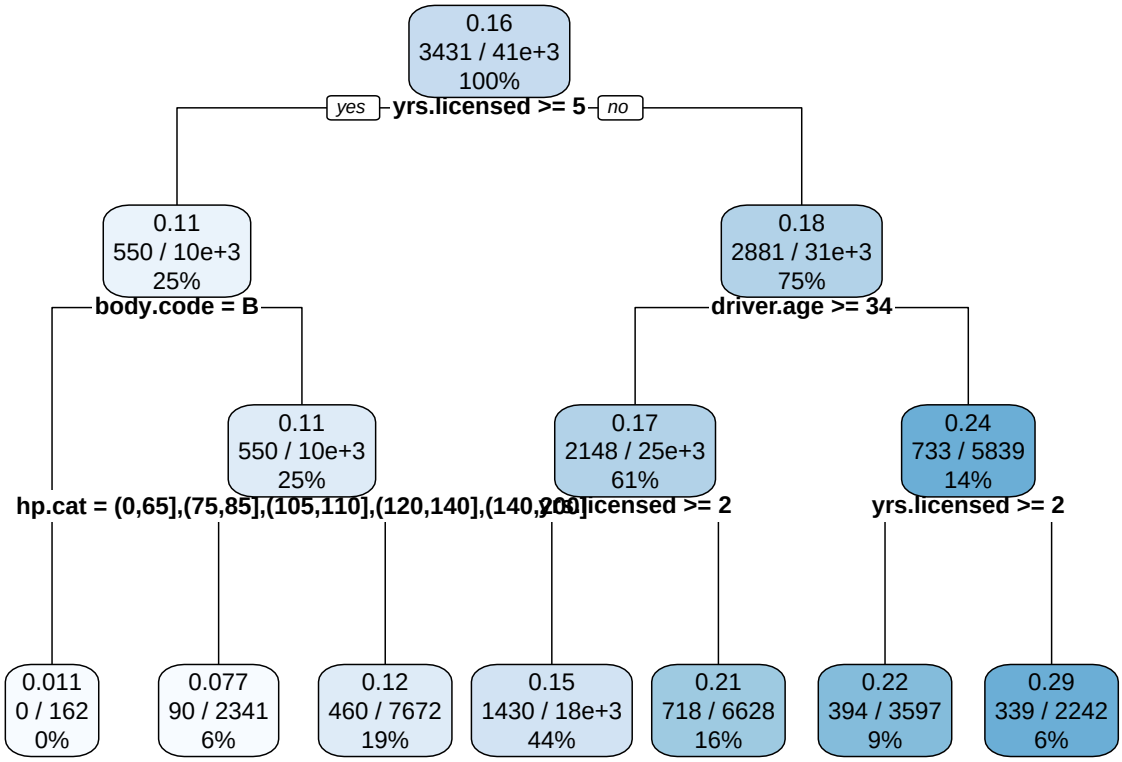
that specifies the proportion by which the overall error should improve for a split to be attempted. We force rpart to generate a full tree by setting that parameter at 0.

```
# A second model
fit2 <- rpart::rpart(formula =
                cbind(exposure, clm.count) ~
                driver.age  + hp.cat
                 + fuel.type + driver.gender + body.code + yrs.licensed + vehicle.value,
                 data = dta_trn,
                 method = 'poisson',
                 control = rpart.control(
                   maxdepth = 3,
                   cp = 0 )
)
print(fit2)
```

```
## n= 40760
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 40760 16616.020000 0.16499330
##    2) yrs.licensed>=4.5 10175  3092.637000 0.10828700
##      4) body.code=B 162     1.863403 0.01126877 *
##      5) body.code=A,C,D,E,F,G,H 10013  3074.595000 0.11007560
##       10) hp.cat=(0,65],(75,85],(105,110],(120,140],(140,200] 2341   566.838800 0.07733158 *
##       11) hp.cat=(70,75],(65,70],(85,90],(90,100],(100,105],(110,120] 7672  2491.222000 0.12020980 *
##    3) yrs.licensed< 4.5 30585 13379.240000 0.18334990
##      6) driver.age>=33.5 24746 10271.200000 0.16912110
##       12) yrs.licensed>=1.5 18118  7085.352000 0.15486790 *
##       13) yrs.licensed< 1.5 6628  3146.832000 0.20703750 *
##      7) driver.age< 33.5 5839  3040.018000 0.24322590
##       14) yrs.licensed>=1.5 3597  1725.477000 0.21539140 *
##       15) yrs.licensed< 1.5 2242  1300.002000 0.28572370 *
```

We get the information on the nodes. Let's print the graph:

```
library(rpart.plot)
rpart.plot(fit2, cex = 0.75)
```



We can try to compute the results manually for quick check.

```
library(dplyr)
dta_trn %>% dplyr::filter(yrs.licensed <5 , driver.age<34, yrs.licensed < 2) %>% dplyr::summarise( clai
```

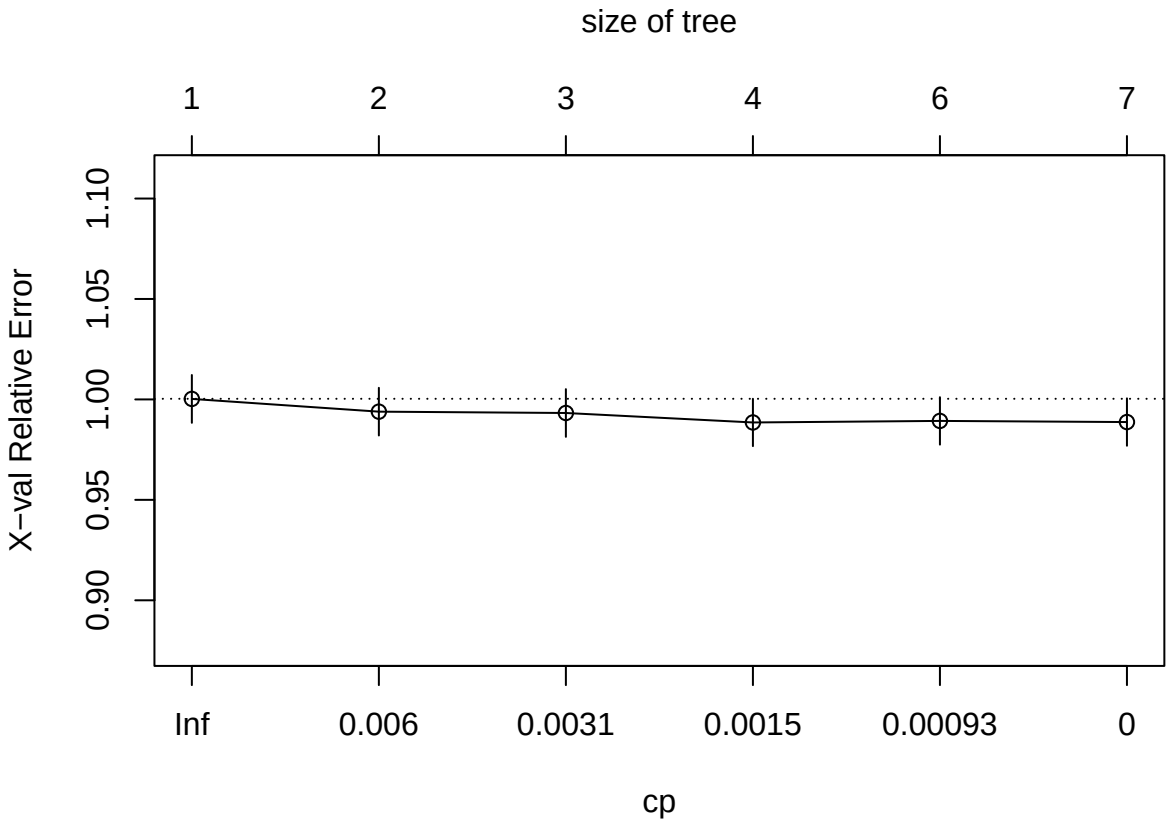
```
##   claim_freq
## 1  0.2863418
```

We obtain 0.286 which is close to the 0.29 showing on the box in the right side of the graph.

Behind the scenes, {rpart} is automatically applying a range of cost complexity values to prune the tree. To compare the error for each value, it performs a 10-fold cross validation so that the error associated with a given value is computed on the hold-out validation data.

The results are summarized in the below graph where the y-axis represents the cross-validation error, the x-axis the cost-complexity value and the upper-x is the number of terminal nodes for the tree.

```
plotcp(fit2)
```



We observe that the error stabilizes for cp = 0.0015 and a size tree of 4 nodes. This can be confirmed directly by printing the results:

```
# Get the cross-validation results
cpt<-fit2$cptable
print(cpt)
```

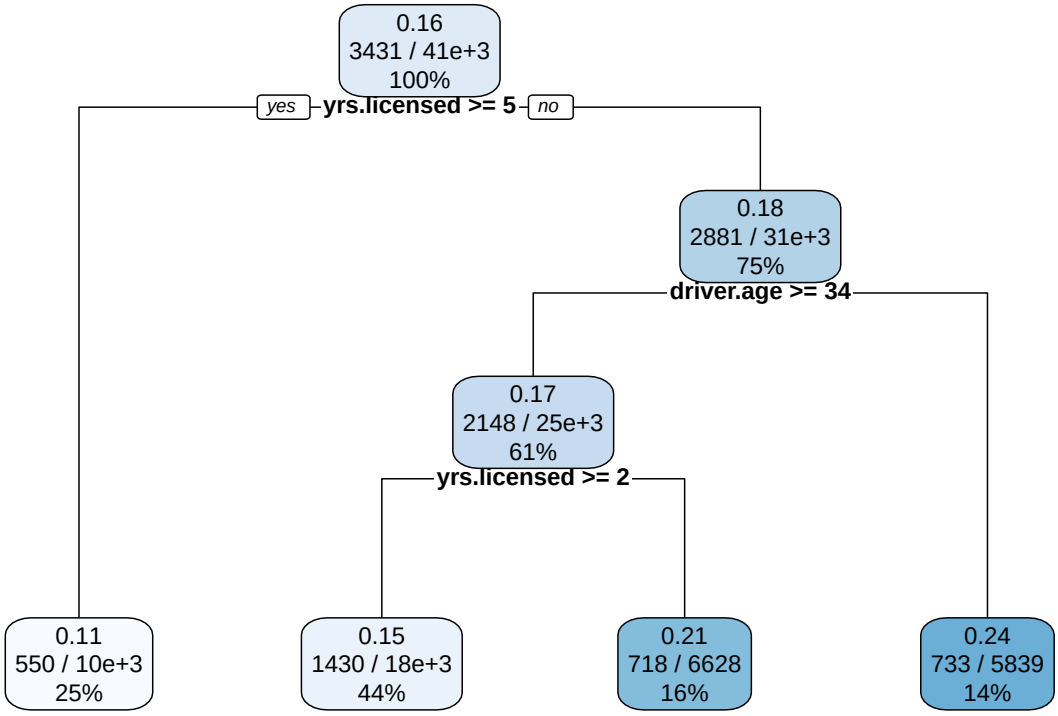
```
##             CP nsplit rel error    xerror       xstd
## 1 0.0086751647      0 1.0000000 1.0002489 0.01190681
## 2 0.0040938157      1 0.9913248 0.9939044 0.01186895
## 3 0.0023481539      2 0.9872310 0.9932463 0.01188049
## 4 0.0009843870      3 0.9848829 0.9885264 0.01178481
## 5 0.0008749926      5 0.9829141 0.9892975 0.01181210
## 6 0.0000000000      6 0.9820391 0.9887286 0.01180780
```

```
# Look for the minimal xerror
min_xerr <- which.min(cpt[,'xerror'])
cpt[min_xerr,]
```

```
##          CP      nsplit   rel error      xerror        xstd
## 0.000984387 3.000000000 0.984882866 0.988526368 0.011784812
```

```
# Prune the tree with an automatic method
fit_srt <- prune(fit2,
cp = cpt[min_xerr, 'CP'])
```

```
# The new tree looks like below
rpart.plot(fit_srt, cex = 0.75)
```



We can perform additional tuning to try to improve the model's performance. 3 other parameters are involved:

- minsplit: the minimum number of data points required to attempt a split before it is forced to create a terminal node,
- maxdepth: the maximum number of internal nodes between the root node and the terminal nodes,
- minbucket: the minimum number of observations in any terminal node (aka "leaf").

Moreover, instead of manually looking at all the possible values for the parameters, we want to set up a grid search to automatically search across a range of differently tuned models to identify the optimal hyerparameter setting.

Grid Search method

Create the hyper-parameter grid

```
hyper_grid <- expand.grid(
  minsplit = seq(1000, 2000, 10),
  maxdepth = seq(3, 8, 1),
  minbucket = seq(1000, 1000, 1)
)

print(head(hyper_grid))
```

```
##   minsplit maxdepth minbucket
## 1     1000        3      1000
## 2     1010        3      1000
## 3     1020        3      1000
## 4     1030        3      1000
## 5     1040        3      1000
## 6     1050        3      1000
```

```
print(nrow(hyper_grid))
```

```
## [1] 606
```

Then we perform a FOR loop across the grid parameter

```
models <- list()

for (i in 1:nrow(hyper_grid)) {

  # get minsplit, maxdepth values at row i
  minsplit <- hyper_grid$minsplit[i]
  maxdepth <- hyper_grid$maxdepth[i]

  # train a model and store in the list
  models[[i]] <- rpart(formula =
                cbind(exposure, clm.count) ~
                driver.age  + hp.cat
                 + fuel.type + driver.gender + body.code + yrs.licensed + vehicle.value,
                 data = dta_trn,
                 method = 'poisson',
    control = list(minsplit = minsplit, maxdepth = maxdepth , xval = 5)
    # xval is the number of cross-validation

    )
}
```

We can now create a function to extract the minimum error associated with the optimal cost complexity value for each model.

```
# function to get optimal cp
get_cp <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  cp <- x$cptable[min, "CP"]
}
```

```
# function to get minimum error
get_min_error <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  xerror <- x$cptable[min, "xerror"]
}
```

```
head(hyper_grid %>%
  mutate(
    cp    = purrr::map_dbl(models, get_cp),
    error = purrr::map_dbl(models, get_min_error)
    ) %>%
  arrange(error) %>%
  top_n(-5, wt = error))
```

```
##   minsplit maxdepth minbucket          cp error
## 1     1000        3      1000 0.008675165     0
## 2     1010        3      1000 0.008675165     0
## 3     1020        3      1000 0.008675165     0
## 4     1030        3      1000 0.008675165     0
## 5     1040        3      1000 0.008675165     0
## 6     1050        3      1000 0.008675165     0
```

5. Interpretability: making sense of a model

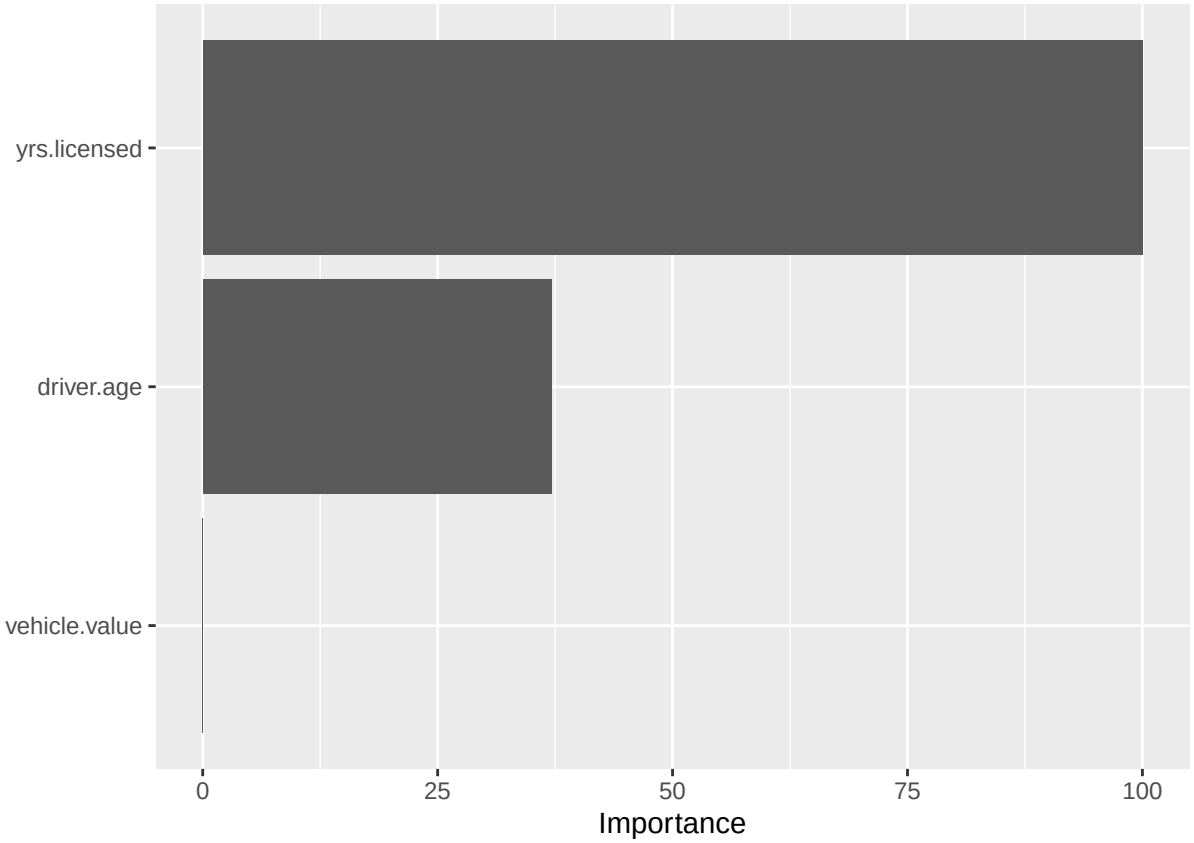
Variable Importance

Feature importance is represented by the reduction in the loss function attributed to each variable at each split. The function vi from the package vip is helpful here.

```
# Use of the package vip
var_imp <- vip::vi(fit_srt)
print(var_imp)
```

```
## # A tibble: 3 x 2
##   Variable      Importance
##   <chr>              <dbl>
## 1 yrs.licensed    183.
## 2 driver.age       68.0
## 3 vehicle.value     0.0800
```

```
# Function vip makes the plot
vip::vip(fit_srt, scale = TRUE)
```



Partial dependence plot

We use partial dependence plots (PDPs) to get an insight on the relation between a feature and the target. It shows the marginal effect that one or two features have on the predicted outcome of a machine learning model (J. H. Friedman). The plot can show whether the relationship between the target and a feature is linear, monotonic or more complex. In our case, the function 'partial' from the {pdp} package performs the essential steps to generate such a PD effect.

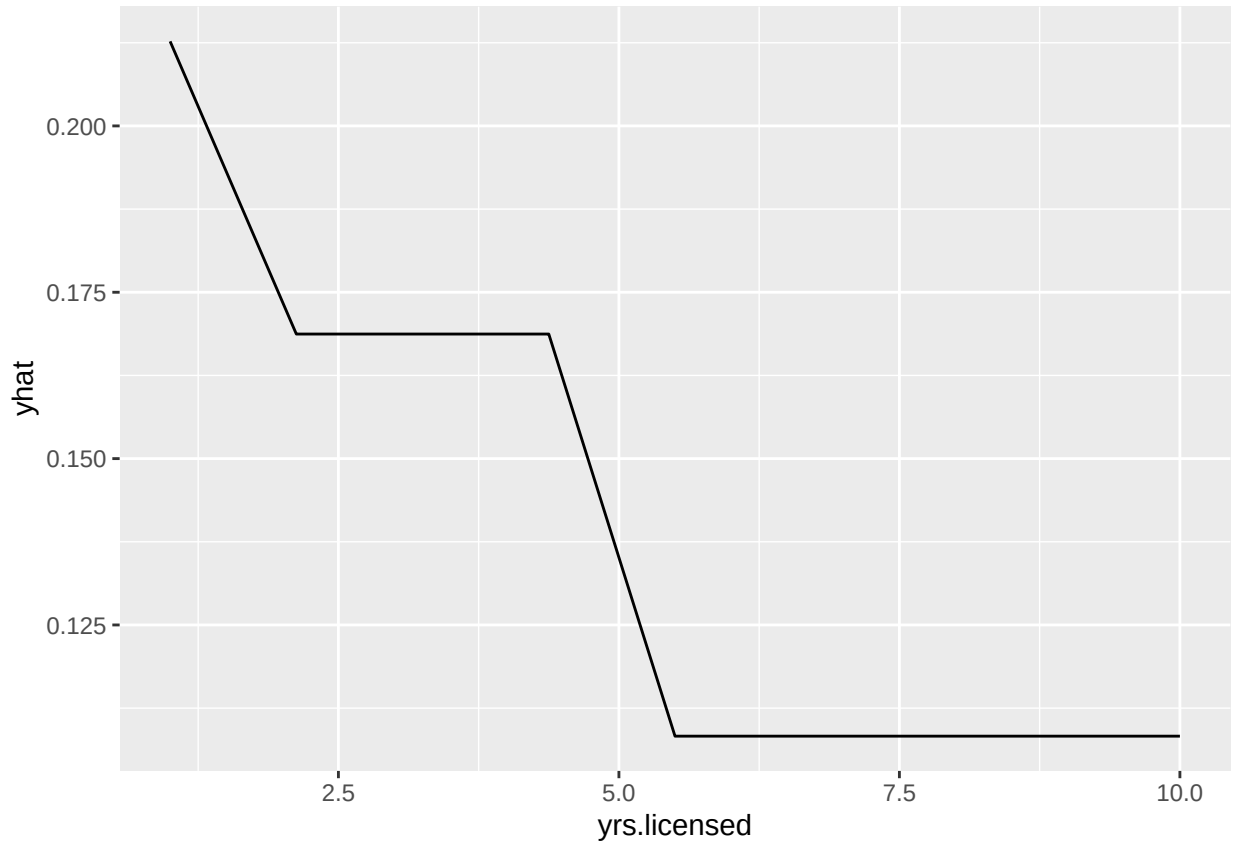
```
# Need to define this helper function for Poisson
pred.fun <- function(object,newdata){
  mean(predict(object, newdata))
}
```

```
pred.fun(fit_srt,dta_trn)
```

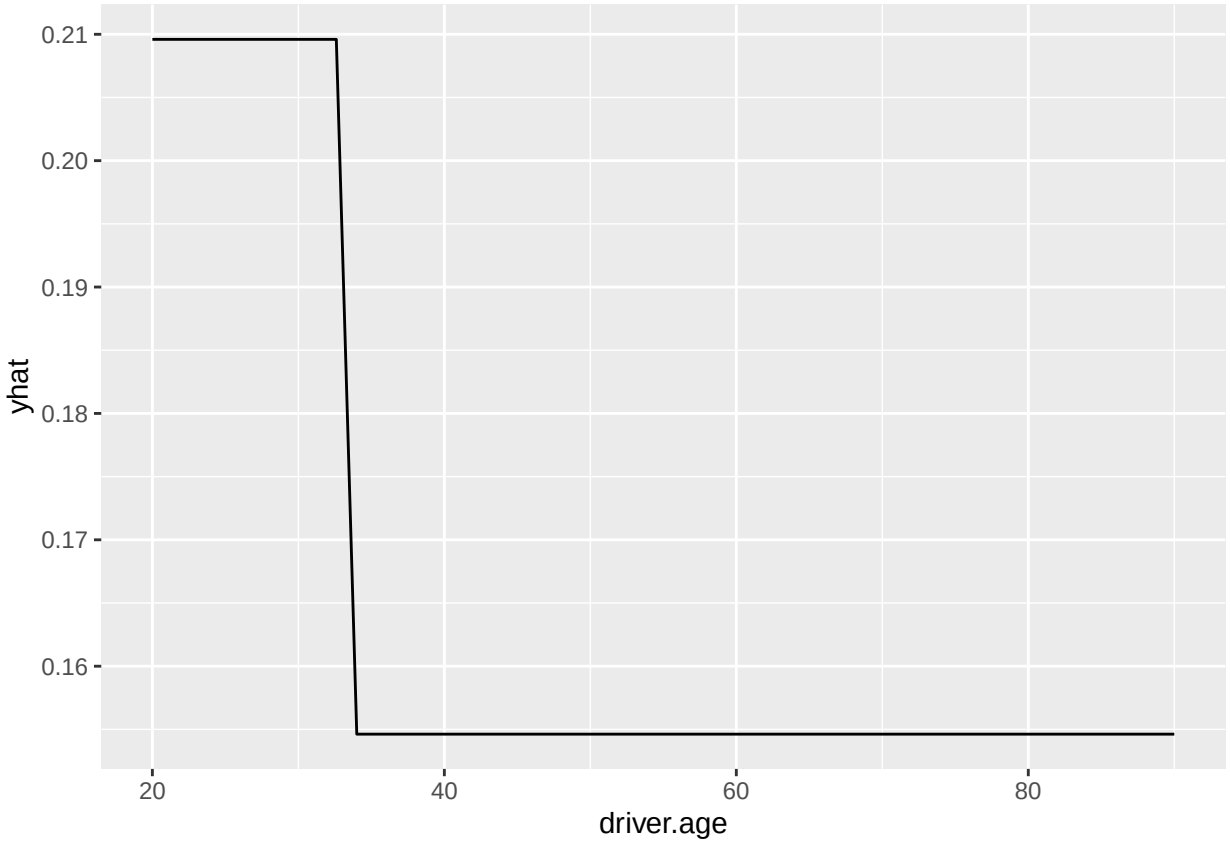
```
## [1] 0.1643807
```

```
set.seed(4525)
pdp_ids <- dta_trn %>% nrow %>% sample(size = 5000)
```

```
# Partial Dependence plot
fit_srt %>% pdp::partial(pred.var = 'yrs.licensed',
                     pred.fun = pred.fun,
                     train = dta_trn[pdp_ids,]) %>% autoplot()
```

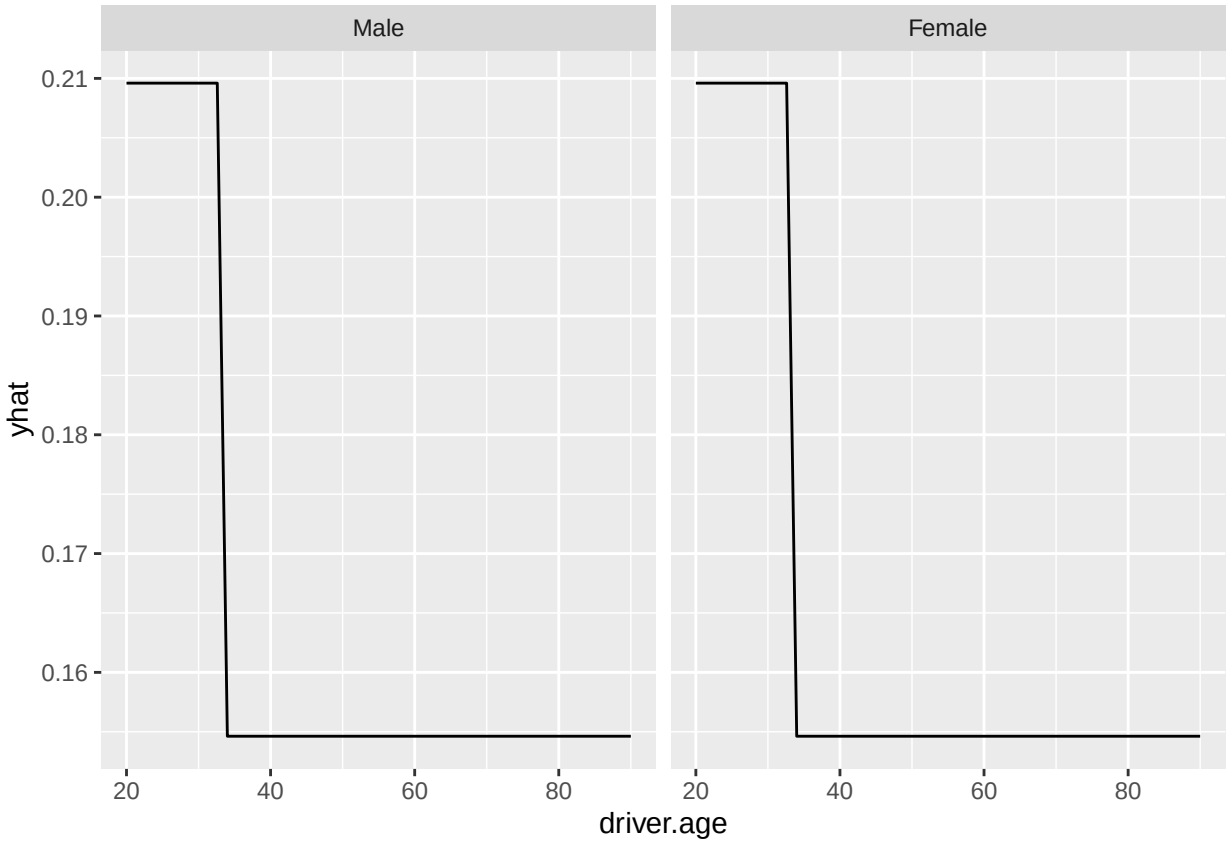



```
# Partial Dependence plot
fit_srt %>% pdp::partial(pred.var = 'driver.age',
                    pred.fun = pred.fun,
                    train = dta_trn[pdp_ids,]) %>% autoplot()
```



Partial Dependence plot in two dimensions

```
fit_srt %>% pdp::partial(pred.var = c('driver.age','driver.gender'),
                         pred.fun = pred.fun,
                         train = dta_trn[pdp_ids,]) %>% autoplot()
```



6. Saving the model

```
# saving the model
saveRDS(fit2, file = "Model_output/Tree_Regression_clm_freq.rda")
```

```
#loading the model
single_tree_freq = readRDS("Model_output/Tree_Regression_clm_freq.rda")
```

7. Prediction on the model

```
# Statistical performance
# Generic prediction function
predict_model <- function(object, newdata) UseMethod('predict_model')
# Prediction function for a regression tree
predict_model.rpart <- function(object, newdata) {
  predict(object, newdata, type = 'vector')
}

# Prediction
tree_freq = single_tree_freq %>% predict_model(newdata = dta_tst)
head(tree_freq)
```

```
##         1         2         3         4         5         6
## 0.1548679 0.1548679 0.1548679 0.2070375 0.2070375 0.1548679
```