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 et 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 aggregation methods like Random Forest or GBM. The aim of this study is purelly educative and stands as an introduction to explore the other Tree-based methods.

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. Predictions are obtained by fitting a simpler model (e.g., a constant like the average response value) in each region.

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

1. Loading the data

```
# Define columnn class for dataset
colCls <- c("integer",         # row id
            "character",       # analysis year
            "numeric",         # exposure
            "character",       # new business / renewal business
            "numeric",         # driver age (continuous)
            "character",       # driver age (categorical)
            "character",       # driver gender
            "character",       # marital status
            "numeric",         # years licensed (continuous)
            "character",       # years licensed (categorical)
            "character",       # ncd level
            "character",       # region
            "character",       # body code
            "numeric",         # vehicle age (continuous)
            "character",       # vehicle age (categorical)
```

```r
            "numeric",          # vehicle value
            "character",        # seats
            rep("numeric", 6), # ccm, hp, weight, length, width, height (all continuous)
            "character",        # fuel type
            rep("numeric", 3)  # prior claims, claim count, claim incurred (all continuous)
)
```

```r
# Define the data path and filename
data.path <- "C:\\Users\\William.Tiritilli\\Documents\\Project P\\Book - Predictive Modeling vol1&2 - F
data.fn <- "sim-modeling-dataset2.csv"

# Read in the data with the appropriate column classes
dta <- read.csv(paste(data.path, data.fn, sep = "/"),
                colClasses = colCls)
str(dta)
```

```
## 'data.frame':    40760 obs. of  27 variables:
##  $ row.id        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ year          : chr  "2010" "2010" "2010" "2010" ...
##  $ exposure      : num  1 1 1 0.08 1 0.08 1 1 0.08 1 ...
##  $ nb.rb         : chr  "RB" "NB" "RB" "RB" ...
##  $ driver.age    : num  63 33 68 68 68 68 53 68 68 65 ...
##  $ drv.age       : chr  "63" "33" "68" "68" ...
##  $ driver.gender : chr  "Male" "Male" "Male" "Male" ...
##  $ marital.status: chr  "Married" "Married" "Married" "Married" ...
##  $ yrs.licensed  : num  5 1 2 2 2 2 5 2 2 2 ...
##  $ yrs.lic       : chr  "5" "1" "2" "2" ...
##  $ ncd.level     : chr  "6" "5" "4" "4" ...
##  $ region        : chr  "3" "38" "33" "33" ...
##  $ body.code     : chr  "A" "B" "C" "C" ...
##  $ vehicle.age   : num  3 3 2 2 1 1 3 1 1 5 ...
##  $ veh.age       : chr  "3" "3" "2" "2" ...
##  $ vehicle.value : num  21.4 17.1 17.3 17.3 25 ...
##  $ seats         : chr  "5" "3" "5" "5" ...
##  $ 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     : chr  "Diesel" "Diesel" "Diesel" "Diesel" ...
##  $ 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 ...
```

```r
set.seed(54321) # reproducibility
# Create a stratified data partition
train_id <- caret::createDataPartition(
  y = dta$clm.count/dta$exposure,
  p = 0.8,
  groups = 100
)[[1]]
```

```r
# Divide the data in training and test set
dta_trn <- dta[train_id,]
dta_tst <- dta[-train_id,]
```

```r
library(dplyr)
```

```
##
## 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
```

```r
# Proportions of the number of claims in train data
dta_trn$clm.count %>% table %>% prop.table %>% round(5)
```

```
## .
##       0       1       2       3       4       5
## 0.92257 0.07163 0.00537 0.00037 0.00003 0.00003
```

```r
# Proportions of the number of claims in test data
dta_tst$clm.count %>% table %>% prop.table %>% round(5)
```

```
## .
##       0       1       2       3
## 0.92098 0.07252 0.00613 0.00037
```

Proportions in train and test sets are well balanced.

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.

```r
# Remove the predictors that are irrelevant for the analysis
dta_trn2 <- subset(dta_trn, select = -c(row.id, drv.age, clm.incurred))
```

```r
library(rpart)       # direct engine for decision tree application
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_trn2,
                method = 'poisson'
                )
print(fit)
```

```
## n= 32610
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 32610 13257.680 0.1646993
##   2) year=2010,2011 13834  4344.694 0.1088814 *
##   3) year=2012,2013 18776  8666.664 0.2065318
##     6) ncd.level=2,4,5,6 8895  3226.219 0.1407561 *
##     7) ncd.level=1,3 9881  5260.987 0.2641312 *
```

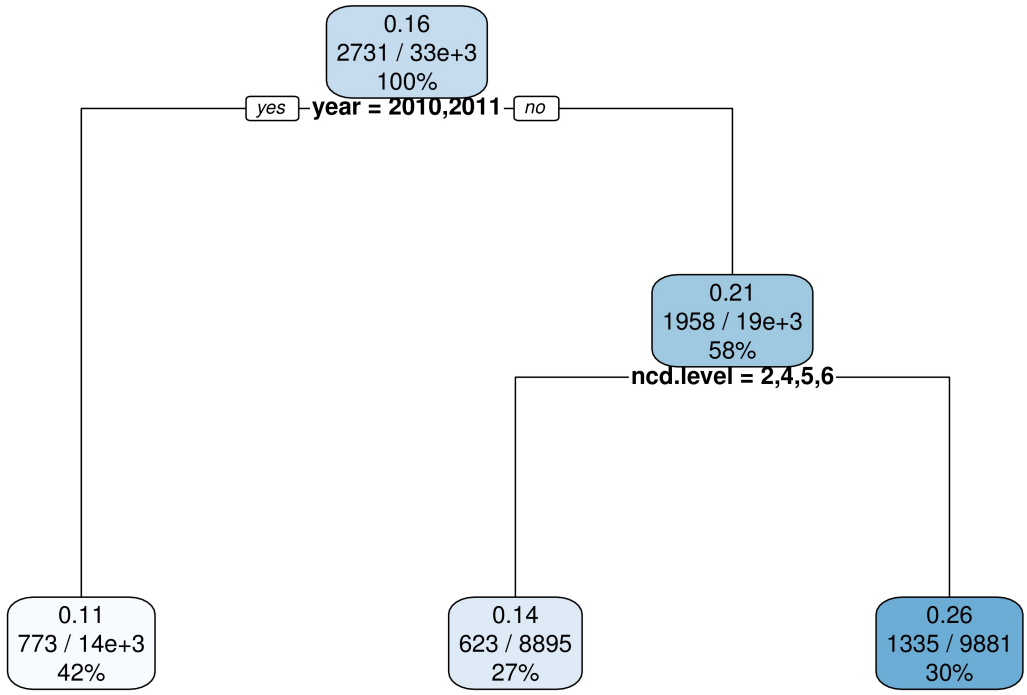
The extract above explains the steps of the splits. Here we start with 32610 observations at the root node, and the first variable that optimize a reduction in deviance is the year of analysis.

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 = 0.75)
```



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.

```
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: population from year 2010 and 2011, representing 42% of the total observations with a claims frequency rate at 11% and the other. The rest of the observation are on the right side of the graph. For this population, another split is done on the "no claim discount" variable. We observe that the difference in claims frequency between the two groups is pretty big: 14% vs 26%.

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.

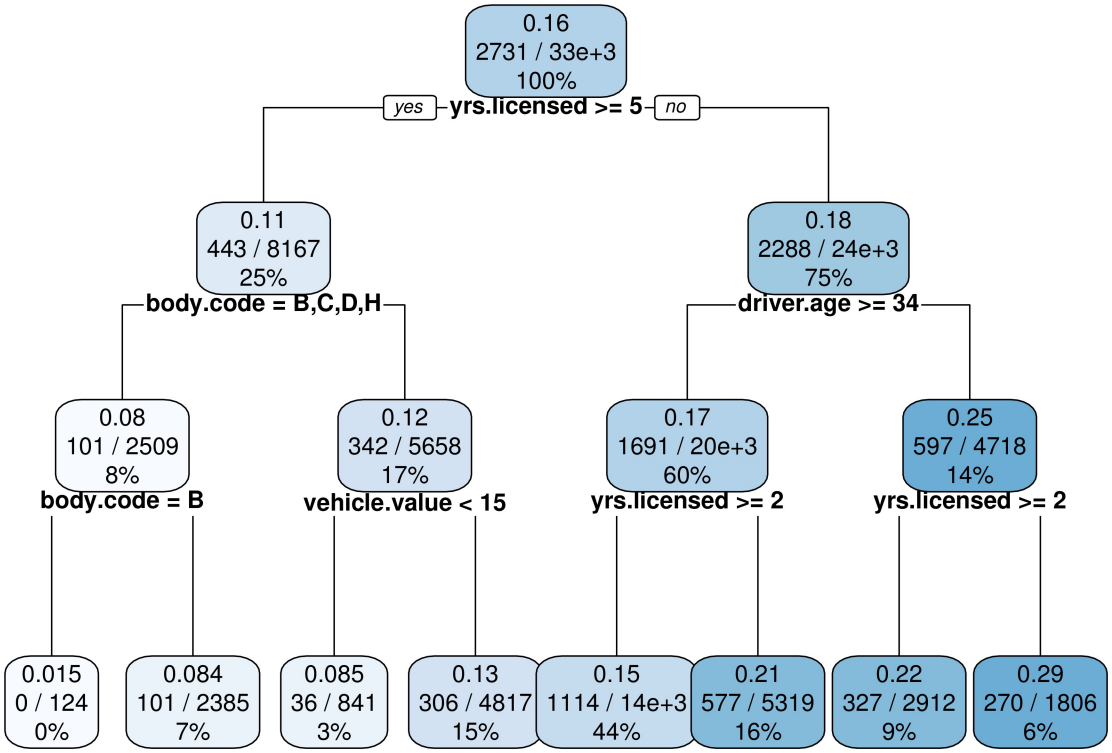
In the formula, 'control' is an argument that provide a list of hyperparameter value. The parameter 'maxdepth' represents the maximum depth of the tree, set up at 3. 'cp' is the complexity parameter, that specify the proportion by which the overall error should improve for a split to be attempted. We 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
                + 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= 32610
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 32610 13257.680000 0.16469930
##    2) yrs.licensed>=4.5 8167  2491.723000 0.10910610
##      4) body.code=B,C,D,H 2509   640.461400 0.08026114
##        8) body.code=B 124     1.821845 0.01467100 *
##        9) body.code=C,D,H 2385   630.293000 0.08438388 *
##      5) body.code=A,E,F,G 5658  1836.083000 0.12229680
##       10) vehicle.value< 15.225 841   217.963900 0.08507866 *
##       11) vehicle.value>=15.225 4817  1611.439000 0.12921790 *
##    3) yrs.licensed< 4.5 24443 10655.050000 0.18276260
##      6) driver.age>=33.5 19725  8136.286000 0.16741700
##       12) yrs.licensed>=1.5 14406  5570.771000 0.15211440 *
##       13) yrs.licensed< 1.5 5319  2529.809000 0.20772230 *
##      7) driver.age< 33.5 4718  2456.388000 0.24669850
##       14) yrs.licensed>=1.5 2912  1415.248000 0.22165730 *
##       15) yrs.licensed< 1.5 1806  1031.732000 0.28516660 *
```

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.

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

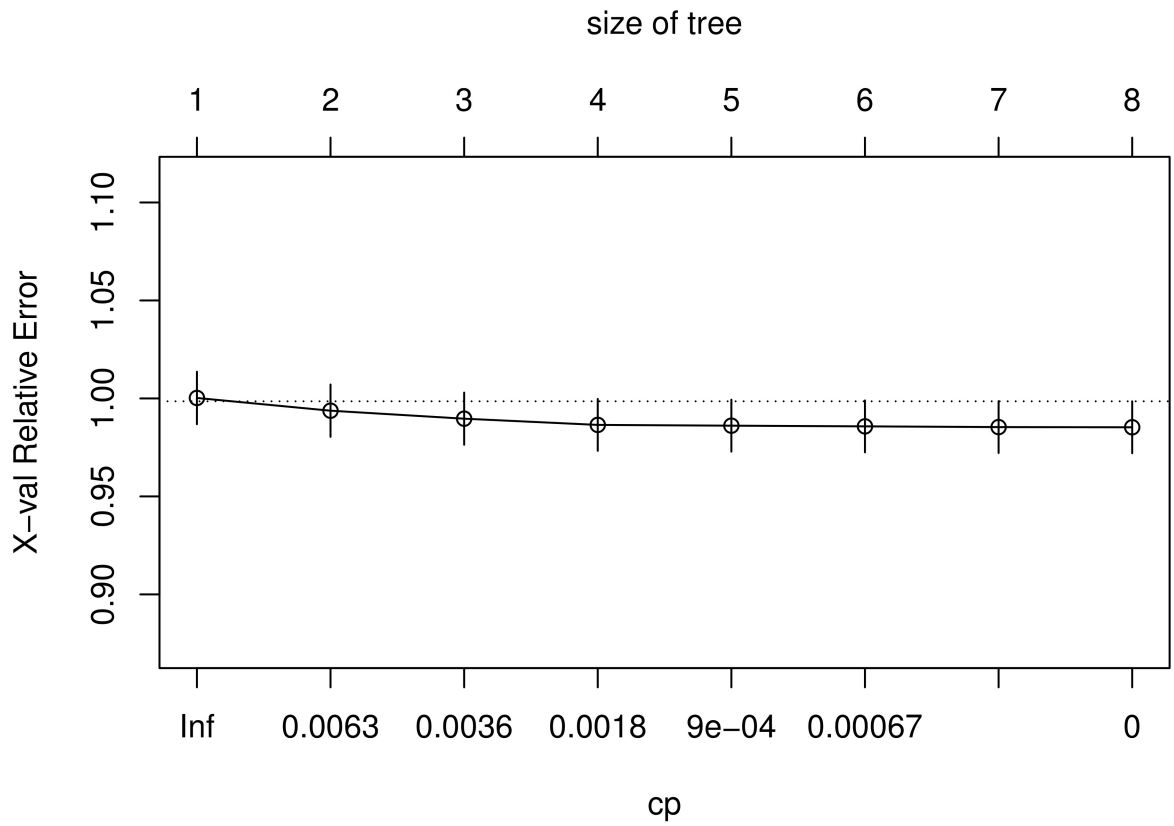
```
##   claim_freq
## 1  0.2859412
```

We obtain 0.285, pretty 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 = 0.0018 and a size tree of 4. This can be confirmed directly by printing the results:

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

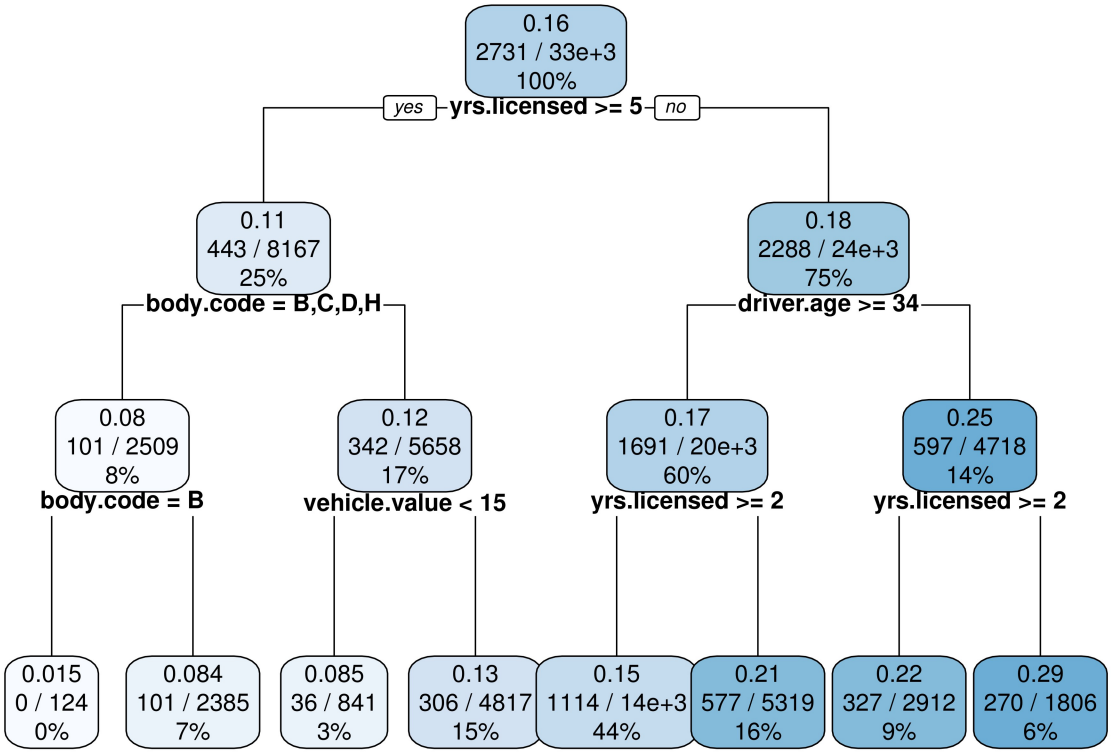
```
##             CP nsplit rel error    xerror       xstd
## 1 0.0083648088      0 1.0000000 1.0002428 0.01339655
## 2 0.0047052172      1 0.9916352 0.9937514 0.01336911
## 3 0.0026932283      2 0.9869300 0.9896825 0.01332146
## 4 0.0011448803      3 0.9842367 0.9865031 0.01325028
## 5 0.0007096248      4 0.9830919 0.9861091 0.01326929
## 6 0.0006295653      5 0.9823822 0.9857480 0.01326556
## 7 0.0005038772      6 0.9817527 0.9853842 0.01325924
## 8 0.0000000000      7 0.9812488 0.9852928 0.01325848
```

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

```
##         CP     nsplit  rel error     xerror       xstd
## 0.00000000 7.00000000 0.98124880 0.98529283 0.01325848
```

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

```r
# 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

```r
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
```

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

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

Perform a FOR loop across the grid parameter

```r
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
                + 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.

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

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

```r
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 0.008364809     0
## 2     1010        3      1000 0.008364809     0
## 3     1020        3      1000 0.008364809     0
## 4     1030        3      1000 0.008364809     0
## 5     1040        3      1000 0.008364809     0
## 6     1050        3      1000 0.008364809     0
```

5. Interpretability: making sense of a model

Variable Importance

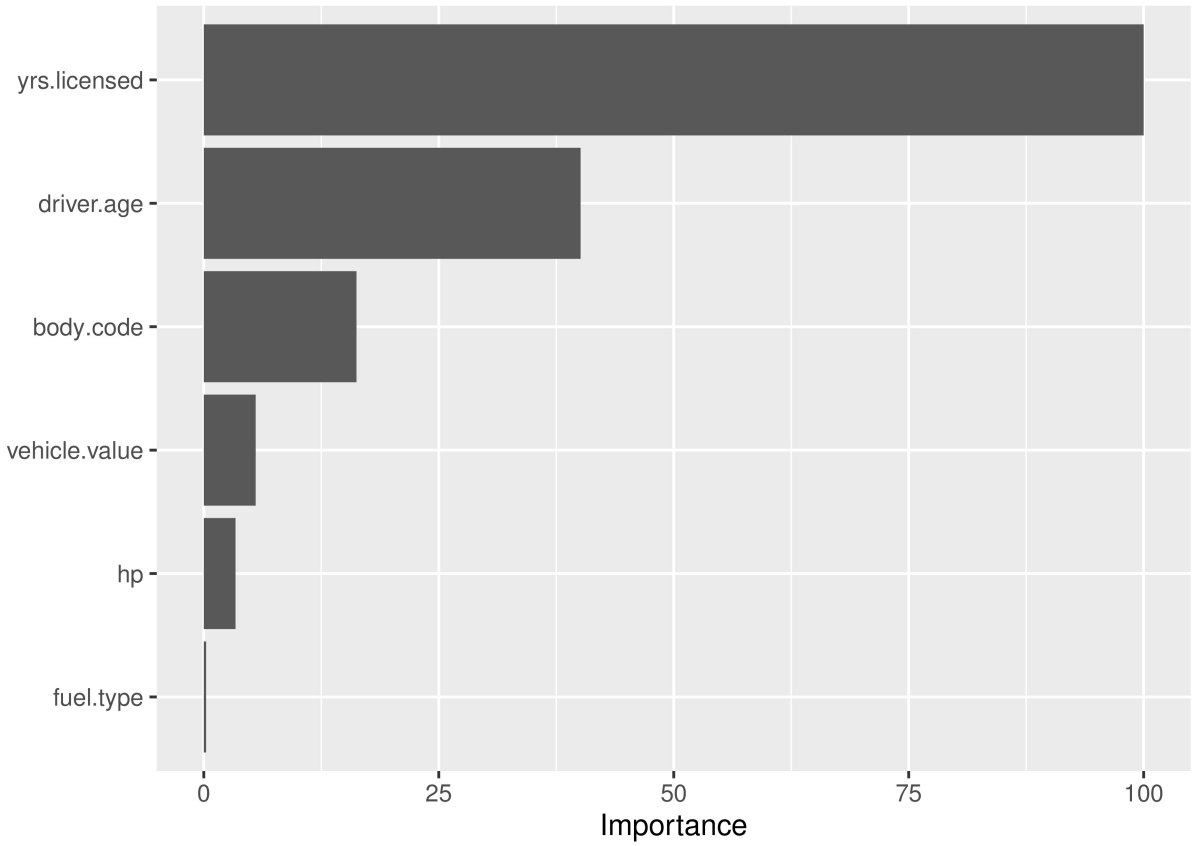
Feature importance is represented by the reduction in the loss function attributed to each variable at each split.

The function vi from the package vip is helpful here.

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

```
## # A tibble: 6 x 2
##   Variable      Importance
##   <chr>              <dbl>
## 1 yrs.licensed     156.
## 2 driver.age        62.5
## 3 body.code         25.3
## 4 vehicle.value      8.61
## 5 hp                 5.27
## 6 fuel.type          0.385
```

```r
# 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 shows the marginal effect one or two features have on the predicted outcome of a machine learning model (J. H. Friedman 200130). 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.

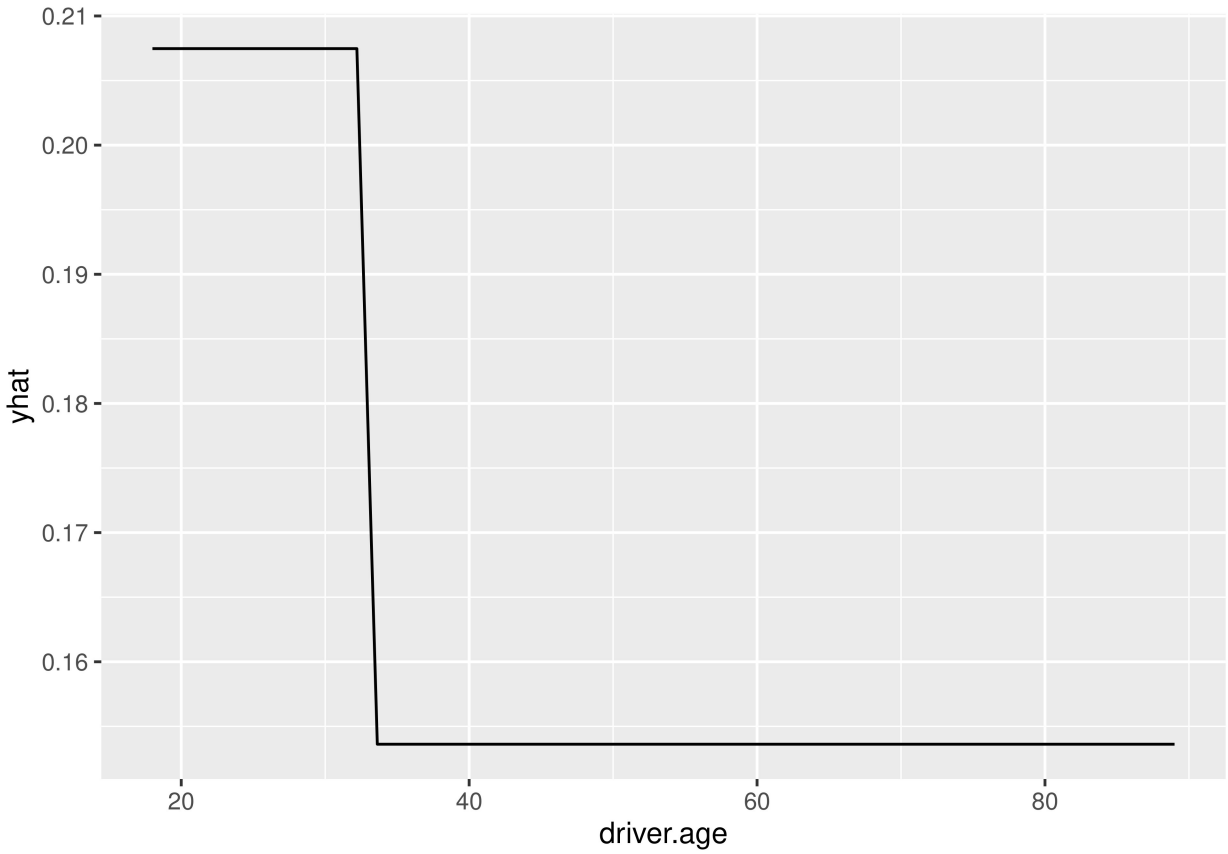
```
# Need to define this helper function for Poisson
pred.fun <- function(object,newdata){
  mean(predict(object, newdata))
}
```

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

```
## [1] 0.164176
```

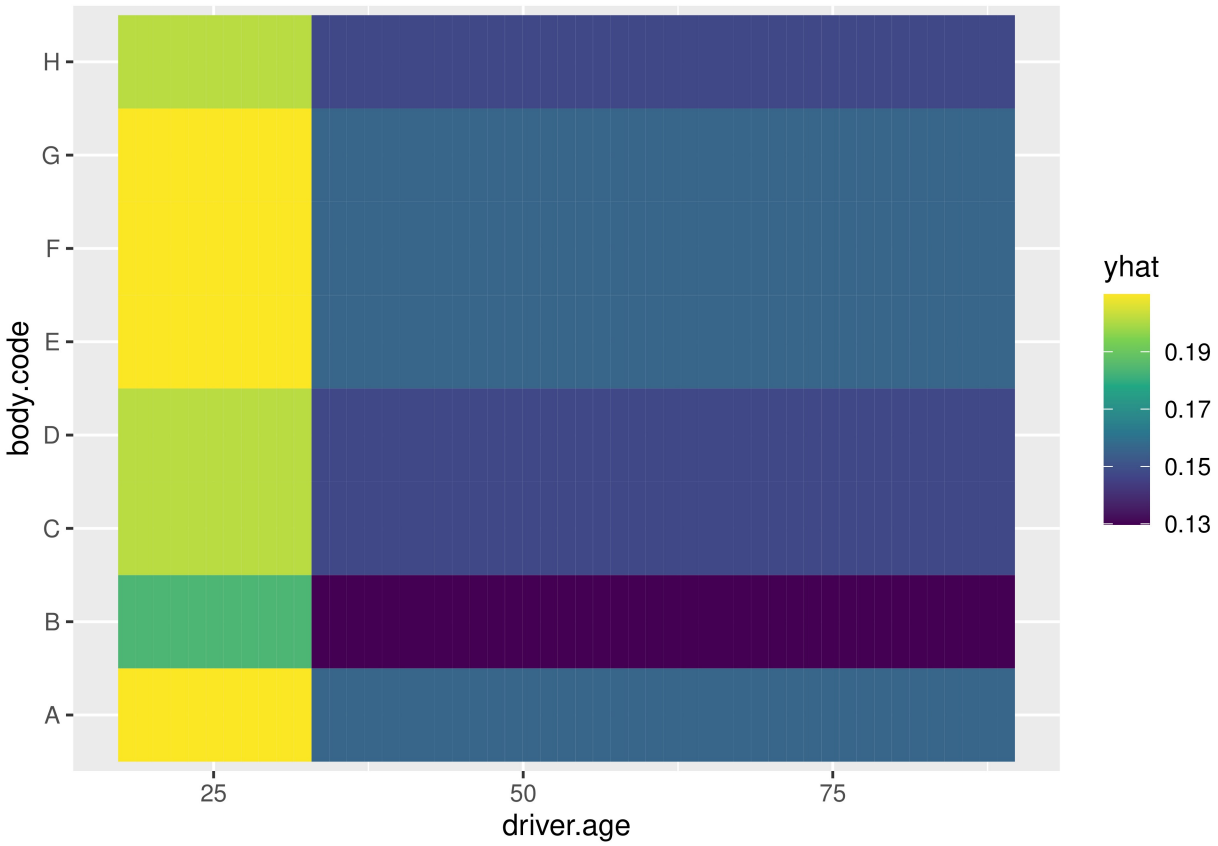
```
set.seed(4525)
pdp_ids <- dta_trn2 %>% nrow %>% sample(size = 5000)
```

```
# Partial Dependence plot
fit_srt %>% pdp::partial(pred.var = 'driver.age',
                     pred.fun = pred.fun,
                     train = dta_trn2[pdp_ids,]) %>% autoplot()
```



Partial Dependence plot in two dimensions

```
fit_srt %>% pdp::partial(pred.var = c('driver.age','body.code'),
                         pred.fun = pred.fun,
                         train = dta_trn2[pdp_ids,]) %>% autoplot()
```


H
G
F
E
D
C
B
A
body.code
25
50
75
driver.age
yhat
0.19
0.17
0.15
0.13