# Regression Trees in Car Insurance: Estimation of Claims Frequency

In this markdown, we will estimate the claims frequency of car drivers using a Regression Tree with the R package {caret}. The data in use come from the chapter one of the book “Predictive Modelling Applications in Actuarial Science, Vol.2”, Edited by 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. 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, they typically lack in predictive performance comparing to aggregation methods like Random Forest or GBM that we will explore in another markdown.

Classification And Regression Tree algorithm, aka CART, developed by Breiman et al. in 1984 works by 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

```r
# 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)
            "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)
)
```

```
# Define the data path and filename
data.path <- "C:\\Users\\William.Tiritilli\\Documents\\Project P\\Frees\\Tome 2 - Chapter 1\\"
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 ...
```

```
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]]
```

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

```
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
```

```
# 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
```

```
# 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.

We start by fitting a simple tree using our train set, taking the predictors that seem to be good candidate.

We calculate a frequency, but how to deal with a claim count in a decision tree? We use a Poisson Deviance as loss function ('method' parameter), keeping the exposure in a two-column matrix.

'maxdepth' represents the maxium depth of the tree 'cp' is the complexity parameter, that specify the proportion by which the overall error should improve for a split to be attempter.

```
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 4.1.2
```

```
fit <- rpart(formula =
               cbind(exposure,clm.count) ~
               driver.age  + hp
                + fuel.type + driver.gender + body.code + yrs.licensed,
                data = dta_trn,
                method = 'poisson',
                control = rpart.control(
                  maxdepth = 3,
                  cp = 0 )
)
print(fit)
```

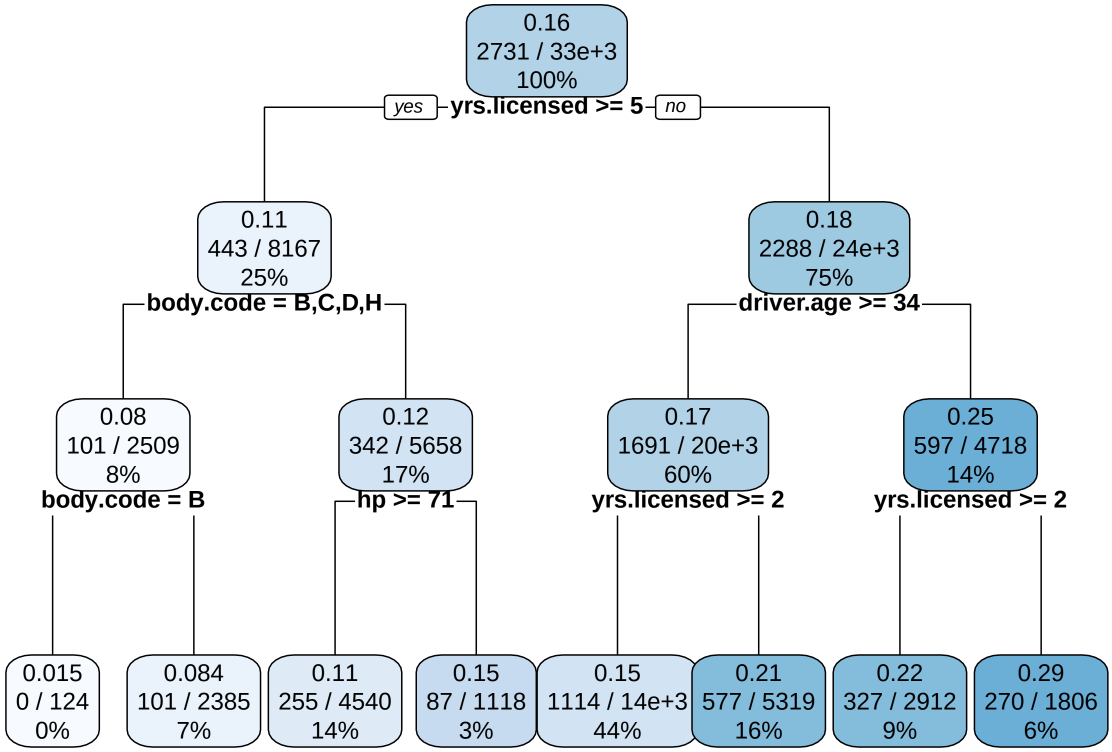
```
## n= 32610
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 32610 13257.680000 0.16469930
##    2) yrs.licensed>=4.5 8167  2491.723000 0.10910610
##      4) body.code=B,C,D,H 2509   640.461400 0.08026114
##        8) body.code=B 124     1.821845 0.01467100 *
##        9) body.code=C,D,H 2385   630.293000 0.08438388 *
##      5) body.code=A,E,F,G 5658  1836.083000 0.12229680
##       10) hp>=70.5 4540  1400.678000 0.11434330 *
##       11) hp< 70.5 1118   429.949900 0.15388610 *
##    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. To get a better idea, we print a graph using the package {rpartplot}:

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

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

```
rpart.plot(fit, cex = 0.75)
```



To check if the tree gives us a similar prediction, we select the criteria of the last branch of the tree, and there is a slight difference.

```
dta_trn %>%
  dplyr::filter(yrs.licensed < 5,
                driver.age < 34, yrs.licensed< 2) %>%
  dplyr::summarise(claim_freq =
                     sum(clm.count)/sum(exposure))
```

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

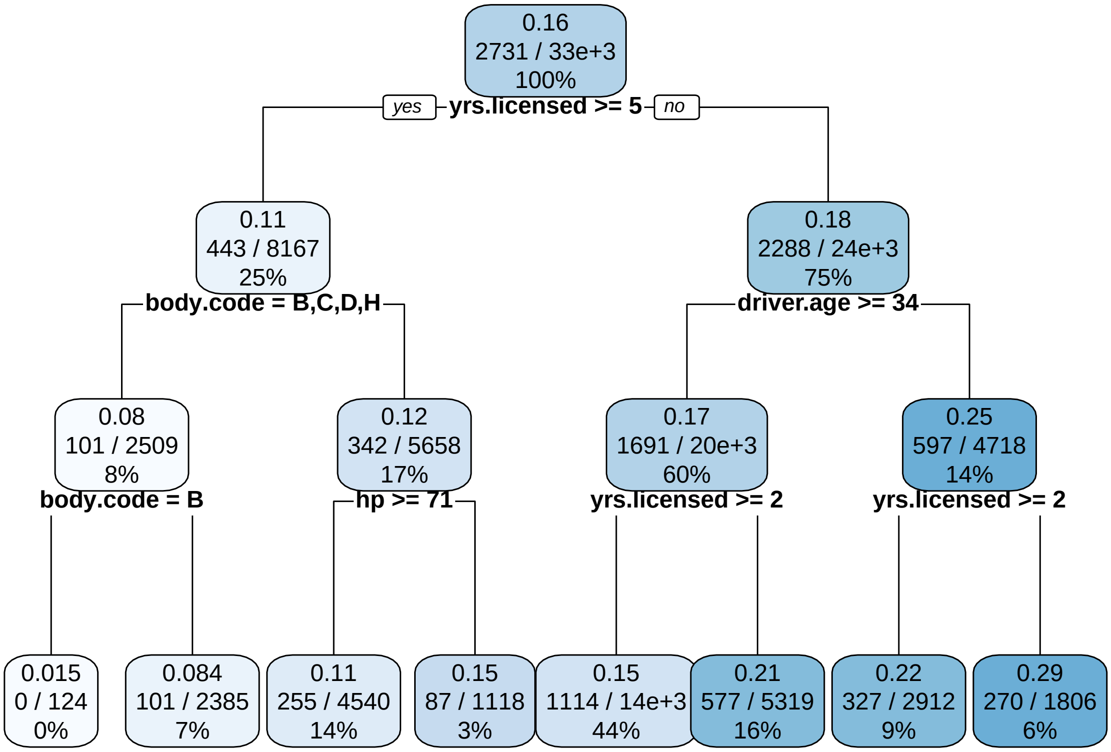
We apply a correction here:

```
rpart.fit <- rpart(formula =
               cbind(exposure,clm.count) ~
               driver.age  + hp
                + fuel.type + driver.gender + body.code + yrs.licensed,
                data = dta_trn,
                method = 'poisson',
                control = rpart.control(
                  maxdepth = 3,
                  cp = 0 ),
             #parms = list(shrink = 10^5)
)
print(rpart.fit)
```

```
## 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) hp>=70.5 4540  1400.678000 0.11434330 *
##       11) hp< 70.5 1118   429.949900 0.15388610 *
##    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 *
```

Now we have the same value: 2.85941

```
rpart.plot(rpart.fit, cex = 0.75)
```



```r
# Tentative of prediction
# Generic prediction function
predict_model <- function(object, newdata) UseMethod('predict_model')
```

```r
tree_predict <-predict(rpart.fit, dta_tst)
```

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

## Pruning the tree

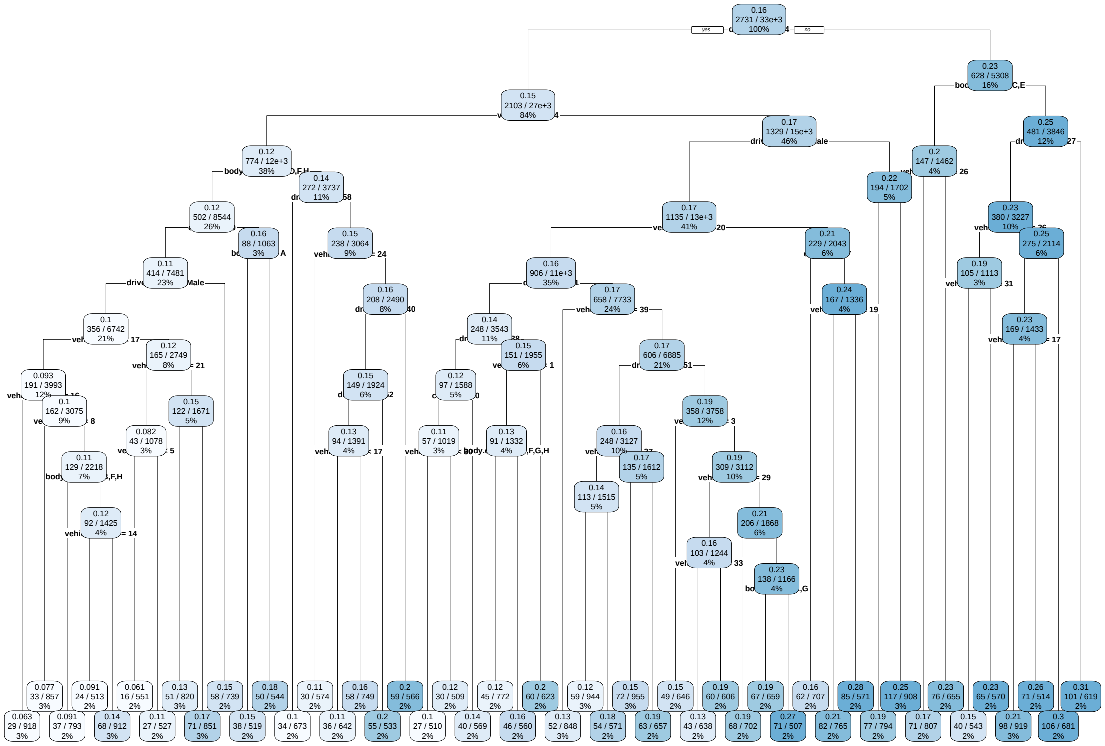
Now we want to follow a pruning strategy to develop a proper model.

We want to built a complex tree and prune it back to find an optimal subtree. To do this, we use the the complexity parameter that penalizes the loss function.

```r
set.seed(9753) # reproducibilty
fit2 <- rpart(formula =
               cbind(exposure,clm.count) ~
               driver.age + vehicle.age + vehicle.value + hp +
               fuel.type  + ccm + body.code + driver.gender,
             data = dta_trn,
             method = 'poisson',
             control = rpart.control(
               maxdepth = 20,
               minsplit = 1000,
               minbucket = 500,
               cp = 0,
               xval = 5
             )
)
```

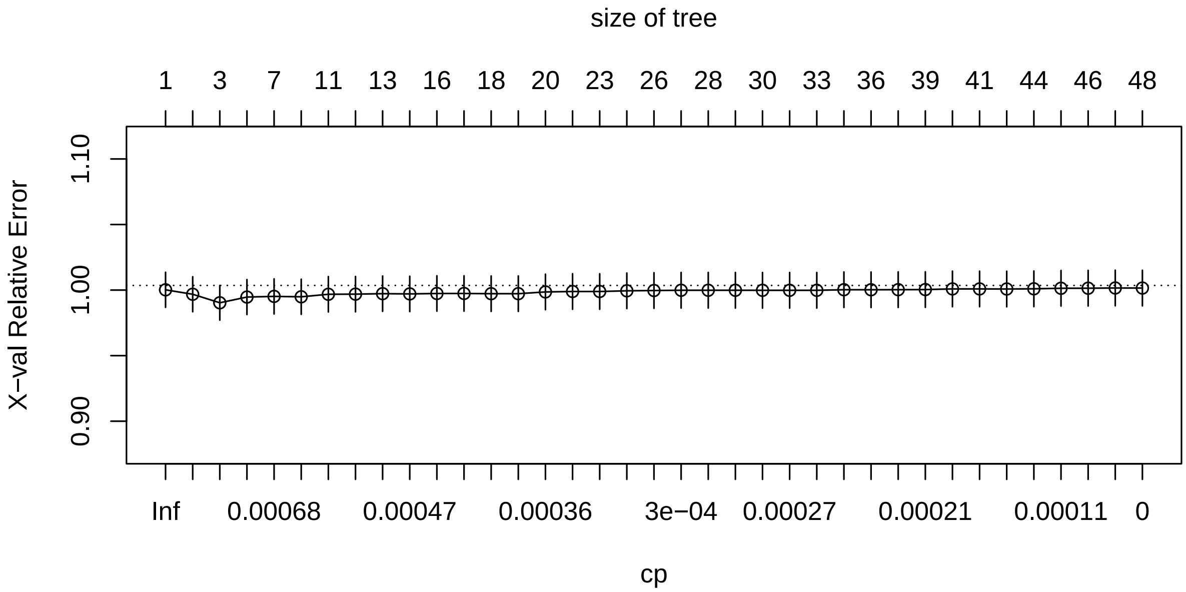
Visualization

```
rpart.plot(fit2, cex = 0.25)
```



We inspect the cross-validation results

```
plotcp(fit2)
```



The pruning cp plot shows the relative corss validation error (y-axis) for various cp value (x-axis).

Breiman (1984) suggested that in actual practice, it's common to instead use the smallest tree within 1 standard error (SE) of the minimum CV error (this is called the 1-SE rule). Here it looks like taking a tree with 3 terminal nodes will give similar results within a small margin of error.

Now we chose the value for the complexity parameter that minimizes the error for pruning.

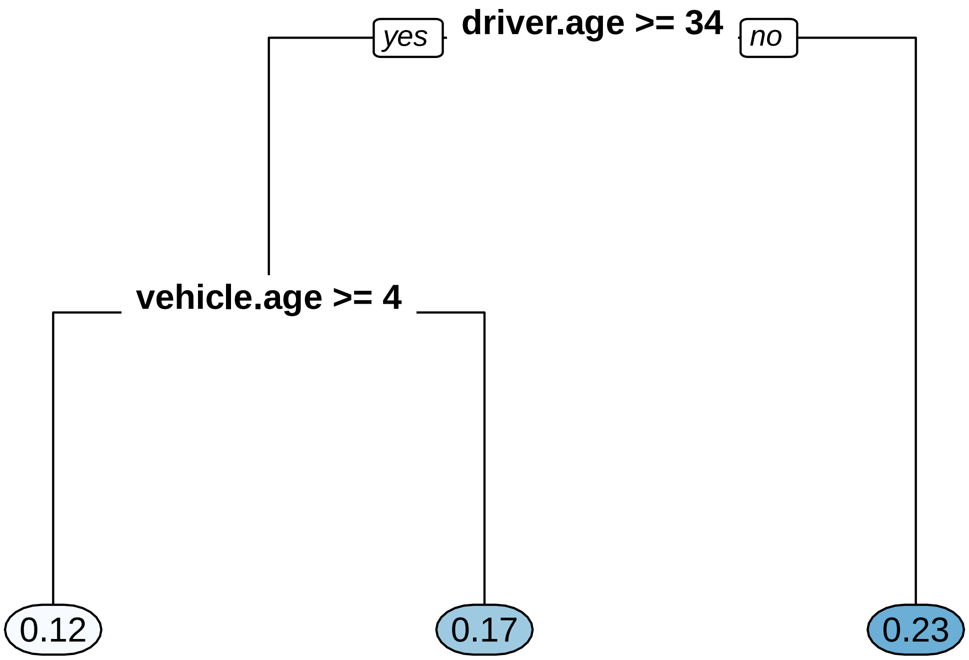
```
# Get the cross-validation results
cpt <- fit2$cptable

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

```
##          CP      nsplit   rel error      xerror        xstd
## 0.001048774 2.000000000 0.989635131 0.990310664 0.013240434
```

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

We can have a look at our new tree.

```
# Plot the tree
rpart.plot(fit_srt, type = 0, extra = 0, cex = 1.1)
```



The tree has been pruned pretty drastically. Two predictors have been retained. But how can we make sense of that.

## Interpretability

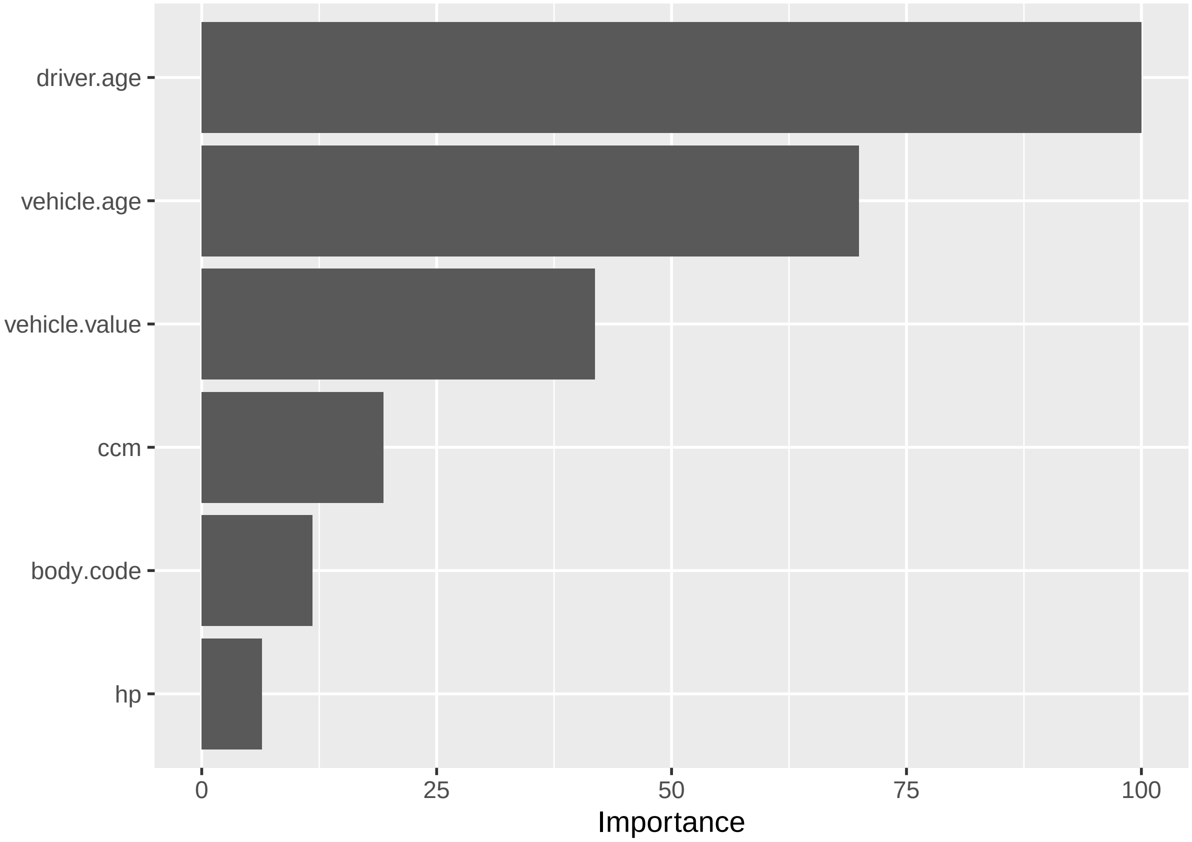
Feature importance is represented by the reduction in the loss function attributed to each variable at each split.

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 driver.age         81.3
## 2 vehicle.age        56.9
## 3 vehicle.value      34.0
## 4 ccm                15.7
## 5 body.code           9.57
## 6 hp                  5.23
```

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



Driver age has a non-linear relationship such that it has increasingly stronger effect on the frequency of claims

Partial dependence plot

Might remove these lines

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

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

```
## [1] 0.1646118
```

Function

### PDPs

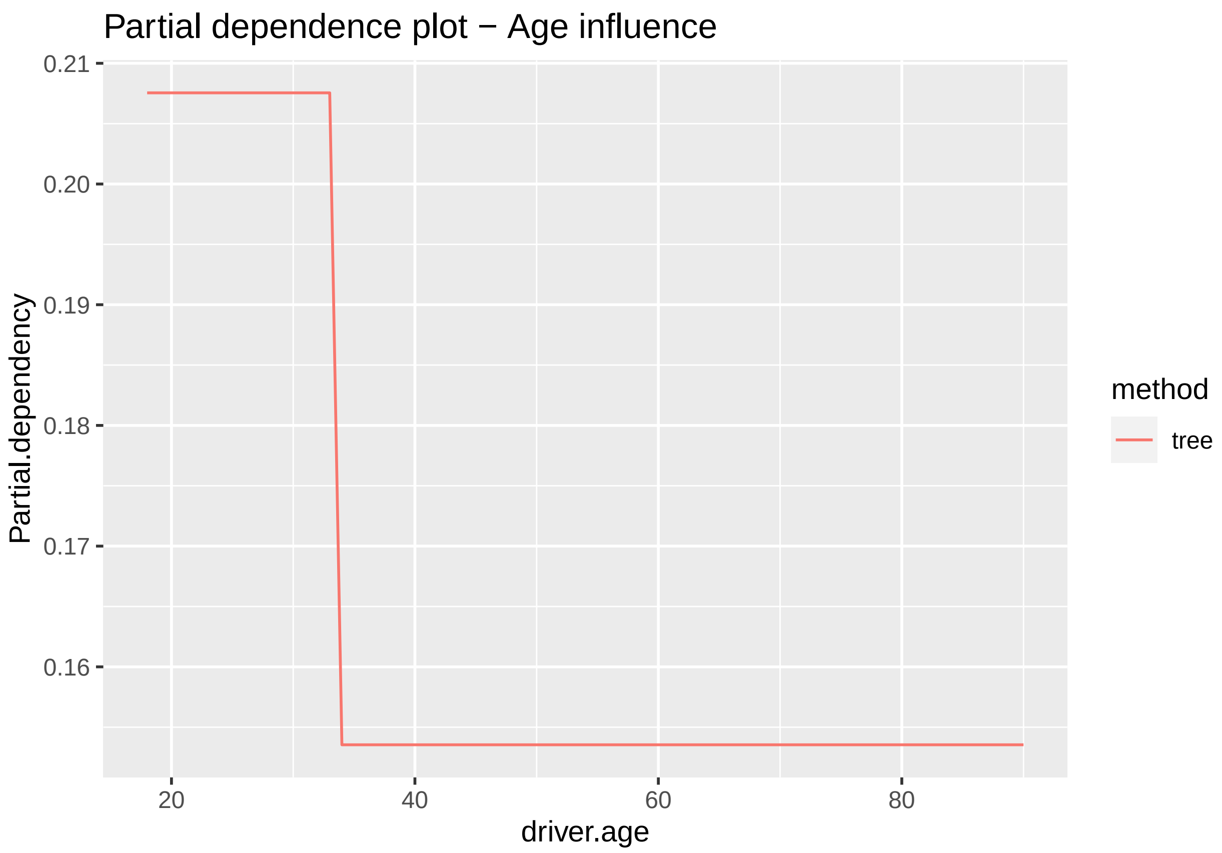
We use partial dependence plots (PDPs) to get an insight on the relation between a feature and the target. The function `par_dep` performs the essential steps to generate such a PD effect. The following steps are performed for each value in a predefined grid of the variable of interest:

. use the original training data (or a subset to speedup calculations) . change the value of the variable of interest to the current value in the grid for all observations . predict the model on this altered data set . calculate the mean of all these predictions to get the PD effect for the current grid value

```r
par_dep <- function(object, data, grid) {
  # Initialize a vector to save the effect
  pd_effect <- rep(0, nrow(grid))
  # Iterate over the grid values to calculate the effect
  for (i in seq_len(length(pd_effect))) {
    pd_effect[i] <-
      data %>%
      dplyr::mutate(!! names(grid) := grid[i, ]) %>%
      predict_model(object, newdata = .) %>% mean() # use of the function
      # predict_model to calculate a prediction on one parameter

  }
  return(pd_effect)
}
```

```r
# Use a random sample of the training observations
set.seed(54321)
dta_trn_sample <- dta_trn[sample(seq_len(nrow(dta_trn)), size = 10000), ]
# Define the grid for the ages
grid_ageph <- data.frame('driver.age' = 18:90)
# Calculate the PD effect for each ML model
grid_ageph <- grid_ageph %>%
  dplyr::mutate(tree = rpart.fit %>% par_dep(data = dta_trn_sample,
                                             grid = grid_ageph))

                #rf = rf_freq %>% par_dep(data = mtpl_trn_sample,
                                         #grid = grid_ageph), # error here
                # no applicable method for 'predict' applied to an object of class "c('rforest', 'list'
                # need to change the predict function for the RF model...

                #gbm = gbm_freq %>% par_dep(data = mtpl_trn_sample,
                                           #grid = grid_ageph))
```

```r
grid_ageph %>% reshape2::melt(id.vars = 'driver.age',
                              value.name = 'Partial.dependency',
                              variable.name = 'method') %>%
  ggplot(aes(x = driver.age, y = Partial.dependency)) +
  geom_line(aes(group = method, colour = method)) +
  ggtitle("Partial dependence plot - Age influence")
```



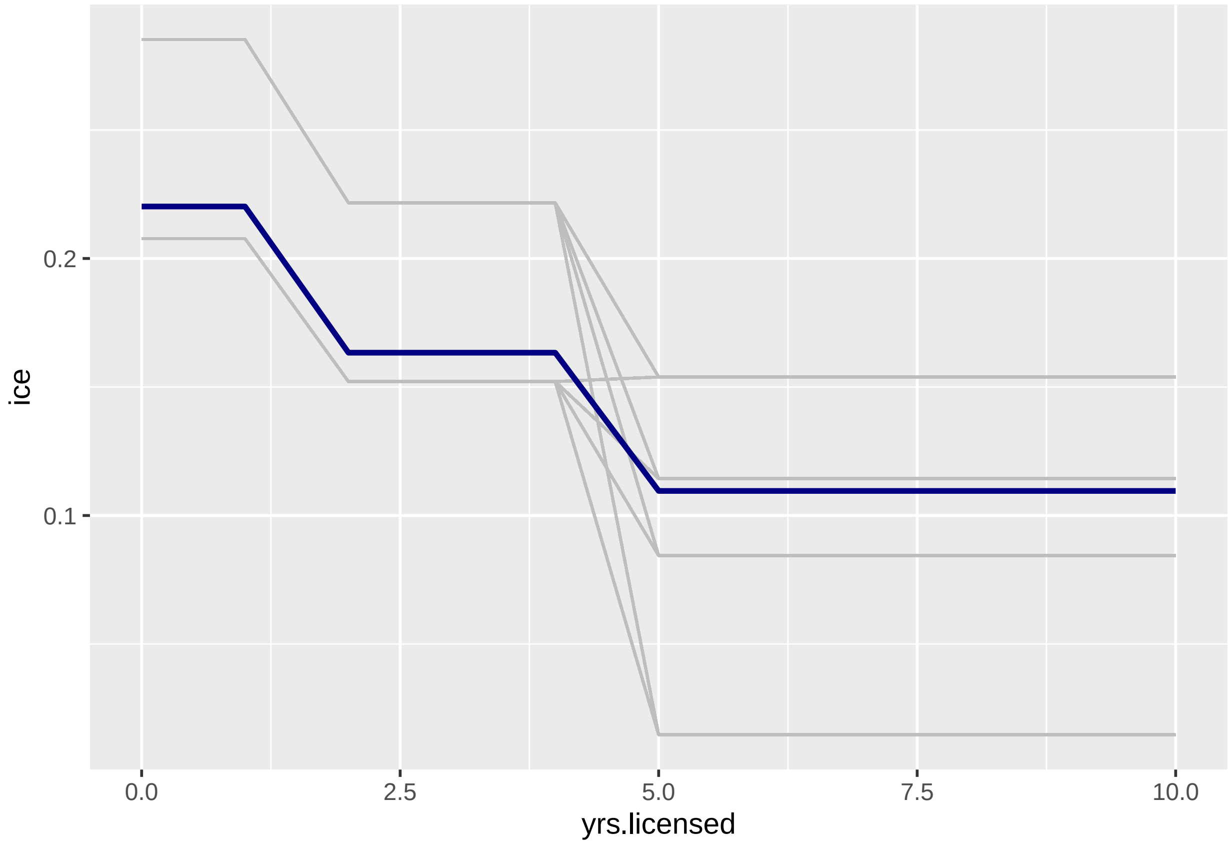
We observe that the risk of filling a claim is high for the young driver and decrease from 33years old.

## Individual conditional expectations (ICEs)

```r
### ICEs
# An individual conditional expectation (ICE) curve is generated in a very comparable way to a PDP.
# The same steps as listed above are followed, only the last step is not performed.
# An ICE curve shows the individual predictions instead of averaging all the predictions (like in a PDP
# The function `ice` to generate an ICE curve is therefore very similar to `par_dep`:

ice <- function(object, data, grid) {
  # Initialize a matrix to save the effect
  ice_effect <- matrix(0, nrow = nrow(grid), ncol = nrow(data))
  # Iterate over the grid values to calculate the effect
  for (i in seq_len(nrow(ice_effect))) {
    ice_effect[i, ] <-
      data %>%
      dplyr::mutate(!! names(grid) := grid[i, ]) %>%
      predict_model(object, newdata = .)
  }
  return(cbind(grid, ice_effect))
}
```

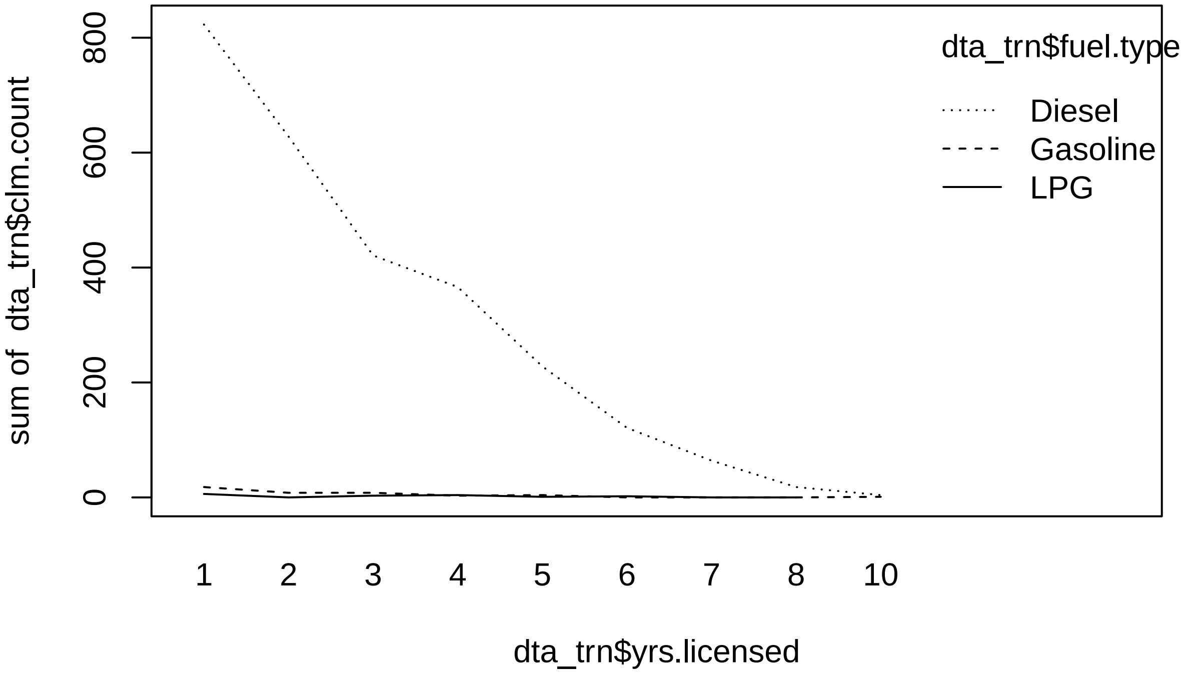
```
# Use a random sample of the training observations
set.seed(54321)
mtpl_trn_sample <- dta_trn[sample(seq_len(nrow(dta_trn)), size = 1000), ]
# Define the grid for the ages
grid_yrs_lic <- data.frame('yrs.licensed' = 0:10)
# Calculate the ICE effect
ice_tree <- rpart.fit %>% ice(data = dta_trn_sample,
                              grid = grid_yrs_lic)
# ice_gbm <- gbm_freq %>% ice(data = mtpl_trn_sample,
#                              grid = grid_bm)
```

```
#After some reshaping we plot these ICE curves with the PD effect on top, as in Figure 8 of the paper:
gridExtra::grid.arrange(
  ice_tree %>% reshape2::melt(id.vars = 'yrs.licensed',
                              value.name = 'ice',
                              variable.name = 'observation') %>%
  dplyr::group_by(yrs.licensed) %>%
  dplyr::mutate(pd = mean(ice)) %>%
  ggplot(aes(x = yrs.licensed)) +
  geom_line(aes(y = ice, group = observation), color = 'grey', alpha = 0.1) +
  geom_line(aes(y = pd), size = 1, color = 'navy')
)
```



## Interaction Effect

```
interaction.plot(dta_trn$yrs.licensed,dta_trn$fuel.type ,dta_trn$clm.count,sum) # No
```



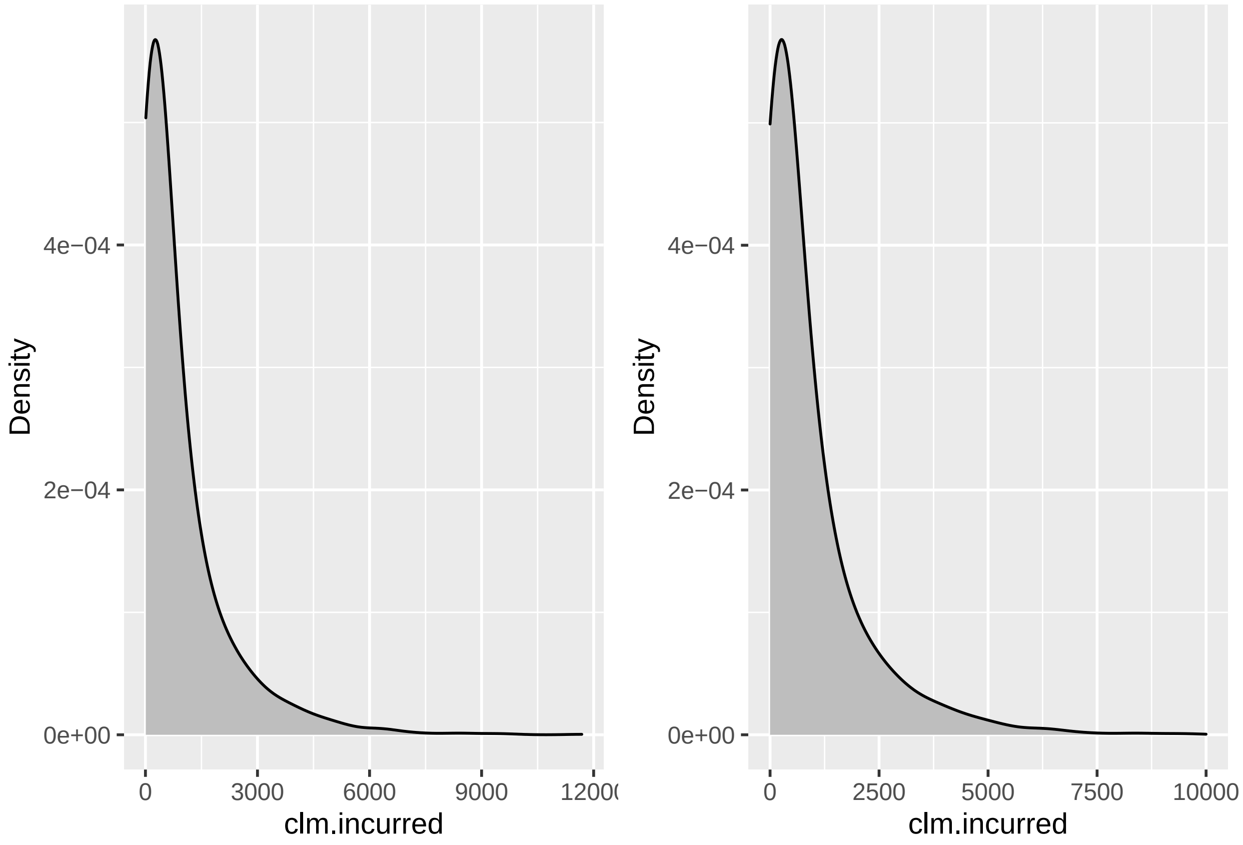
No interaction between the number of years licensed and the fuel type.

## Statistical performance

Quickly we fit a tree model for the claim severity

```
# Only retain the claims
dta_trn_claims <- dta_trn %>% dplyr::filter(clm.count > 0)
# Plot the density of all observations and those below 10 000 Euro
gridExtra::grid.arrange(
      ggplot(dta_trn_claims, aes(x = clm.incurred )) +
      geom_density(adjust = 3, col = 'black', fill = 'gray') +
      labs(y = 'Density'),
      ggplot(dta_trn_claims, aes(x = clm.incurred )) +
      geom_density(adjust = 3, col = 'black', fill = 'gray') +
      labs(y = 'Density') + xlim(0, 1e4),
      ncol = 2
        )
```

```
## Warning: Removed 1 rows containing non-finite values (stat_density).
```



```
features <- c('driver.age', 'hp',
                'fuel.type ', 'driver.gender', 'body.code', 'yrs.licensed')
```

```
tree_sev <- distRforest::rpart(
  formula =  clm.incurred ~
               driver.age  + hp
                + fuel.type + driver.gender + body.code + yrs.licensed,
  data = dta_trn_claims,
  weights = clm.count,
  method = 'gamma',
  control = rpart.control(cp = 3.7e-3, # cp in Table 3
                          minbucket = 0.01 * nrow(dta_trn_claims), # kappa in Table 1
                          xval = 0,
                          maxcompete = 0,
                          maxsurrogate = 0)
  )
```

```
## Registered S3 methods overwritten by 'distRforest':
##   method             from
##   labels.rpart       rpart
##   model.frame.rpart rpart
##   plot.rpart         rpart
##   predict.rpart      rpart
##   print.rpart       rpart
##   residuals.rpart   rpart
##   summary.rpart     rpart
##   text.rpart        rpart
```

```
#Transfo of predictor in factor for gbm
dta_trn$fuel.type <-as.factor(dta_trn$fuel.type)
dta_trn$driver.gender <-as.factor(dta_trn$driver.gender)
dta_trn$body.code <-as.factor(dta_trn$body.code)
dta_trn$yrs.licensed <-as.factor(dta_trn$yrs.licensed)

dta_trn_claims$fuel.type <-as.factor(dta_trn_claims$fuel.type)
dta_trn_claims$driver.gender <-as.factor(dta_trn_claims$driver.gender)
dta_trn_claims$body.code <-as.factor(dta_trn_claims$body.code)
dta_trn_claims$yrs.licensed <-as.factor(dta_trn_claims$yrs.licensed)
```

```
library(gbm)
```

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

```
## Loaded gbm 2.1.8
```

```
set.seed(54321)
gbm_freq <- gbm(
  formula = as.formula(paste('clm.count ~ offset(log(exposure)) +', paste(features, collapse = ' + ')))
  data = dta_trn,
  distribution = 'poisson',
  n.trees = 1400, # T in Table 3
  interaction.depth = 5, # d in Table 3
  shrinkage = 0.01, # lambda in Table 1
  bag.fraction = 0.75, # delta in Table 1
  n.minobsinnode = 0.01 * 0.75 * nrow(dta_trn), # kappa * delta in Table 1
  verbose = FALSE
  )

set.seed(54321)
gbm_sev <- gbm(
  formula = as.formula(paste('clm.incurred ~', paste(features, collapse = ' + '))),
  data = dta_trn_claims,
  weights = clm.count,
  #distribution = 'gamma', # Gamma distribution not supported by the package?
  distribution = 'gaussian',

  n.trees = 500, # T in Table 3
  interaction.depth = 1, # d in Table 3
  shrinkage = 0.01, # lambda in Table 1
  bag.fraction = 0.75, # delta in Table 1
  n.minobsinnode = 0.01 * 0.75 * nrow(dta_trn_claims), # kappa * delta in Table 1
  verbose = FALSE
  )
```

```
# Prediction function for a GBM
predict_model.gbm <- function(object, newdata) {
  predict(object, newdata, n.trees = object$n.trees, type = 'response')
}
```

```
oos_pred <- tibble::tibble(
  tree_freq = rpart.fit %>% predict_model(newdata = dta_tst),
  tree_sev = tree_sev %>% predict_model(newdata = dta_tst),
  gbm_freq = gbm_freq %>% predict_model(newdata = dta_tst),
  gbm_sev = gbm_sev %>% predict_model(newdata = dta_tst)
)
```

```
## Warning in predict.gbm(object, newdata, n.trees = object$n.trees, type =
## "response"): NAs introduced by coercion
```

```
## Warning in predict.gbm(object, newdata, n.trees = object$n.trees, type =
## "response"): predict.gbm does not add the offset to the predicted values.
```

```
## Warning in predict.gbm(object, newdata, n.trees = object$n.trees, type =
## "response"): NAs introduced by coercion
```

```
# These predictions are compared to the observed values in `mtpl_tst` with the Poisson/gamma
# deviance for frequency/severity models respectively:

# Poisson deviance
dev_poiss <- function(ytrue, yhat) {
  -2 * mean(dpois(ytrue, yhat, log = TRUE) - dpois(ytrue, ytrue, log = TRUE), na.rm = TRUE)
}
# Gamma deviance
dev_gamma <- function(ytrue, yhat, wcase) {
  -2 * mean(wcase * (log(ytrue/yhat) - (ytrue - yhat)/yhat), na.rm = TRUE)
}
```

```
# The out-of-sample deviances are calculated below.
# These are the values for data fold 3 in Figure 10 of the paper.

# Calculate the Poisson deviance for the frequency models
oos_pred %>% dplyr::select(ends_with('_freq')) %>%
  purrr::map(~ dev_poiss(dta_tst$clm.count, .x * dta_tst$exposure))
```

```
## $tree_freq
## [1] 0.4060604
##
## $gbm_freq
## [1] 0.4084019
```

```
# Calculate the gamma deviance for the severity models
oos_pred %>% dplyr::select(ends_with('_sev')) %>%
  purrr::map(~ dev_gamma(dta_tst$clm.incurred, .x, dta_tst$clm.count))
```

```
## $tree_sev
## [1] 2.226507
##
## $gbm_sev
## [1] 2.206478
```

```
## Economic lift

#After comparing the ML models for frequency and severity,
#we now turn to a comparison at the premium level.
#We calculate the predicted premiums for the test data `mtpl_tst`
#by multiplying the frequency and severity:

oos_pred <- oos_pred %>% dplyr::mutate(
  tree_prem = tree_freq * tree_sev, #wt: freq*sev = pure premium
  # rf_prem = rf_freq * rf_sev,
  gbm_prem = gbm_freq * gbm_sev
)
```

```
oos_pred %>% dplyr::select(ends_with('_prem')) %>%
  dplyr::summarise_all(~ sum(.x * dta_tst$exposure))
```

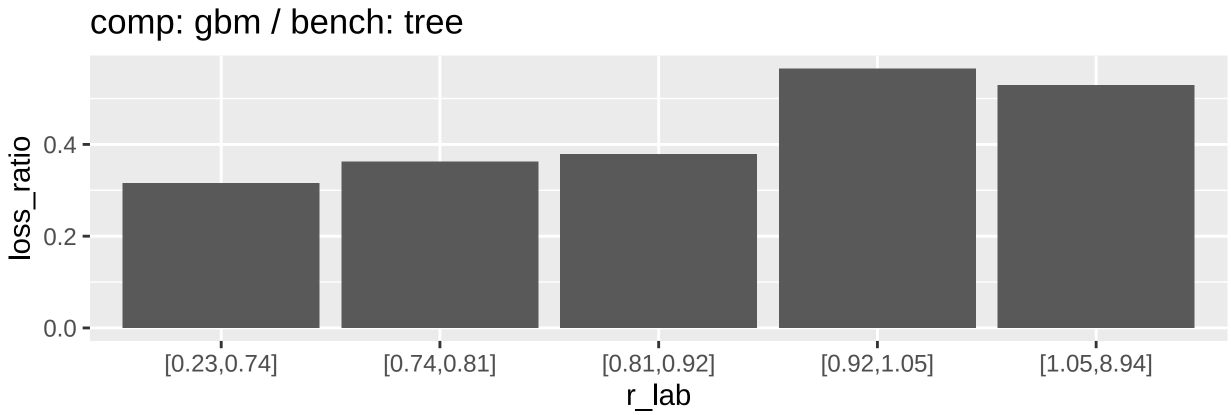
```
## # A tibble: 1 x 2
##   tree_prem gbm_prem
##       <dbl>    <dbl>
## 1   643524.  548532.
```

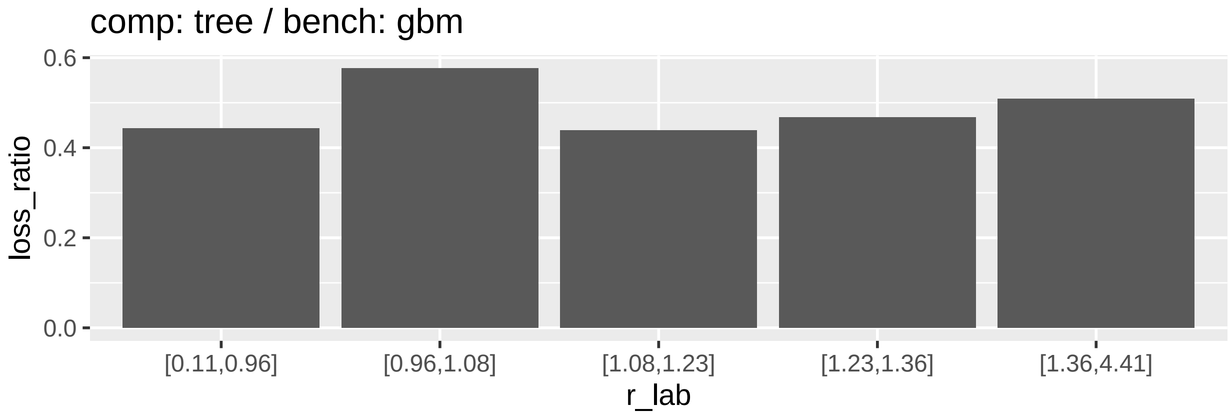
```
#We now focus on some model lift measures, which are introduced
#and analyzed in Sections 5.1 and 5.2 of the paper.
#To streamline the coding we add the observed target values from
#the test data `mtpl_tst` to the predictions data:

oos_pred <- oos_pred %>% dplyr::mutate(
  nclaims = dta_tst$clm.count,
  expo = dta_tst$exposure,
  amount = dta_tst$clm.incurred
)
```

```
### Loss ratio lift
# The loss ratio lift is assessed by applying the following steps:
#
#   + sort policies from smallest to largest relativity
# + bin the policies in groups of equal exposure
# + calculate the loss ratio in each bin using the benchmark premium

loss_ratio_lift <- function(data, bench, comp, ngroups) {

  # Calculate relativity and sort from small to large
  data %>% dplyr::mutate(r = get(paste0(comp, '_prem')) / get(paste0(bench, '_prem'))) %>%
    dplyr::arrange(r) %>%
    # Bin in groups of equal exposure
    dplyr::mutate(bin = cut(cumsum(expo),
                            breaks = sum(expo) * (0:ngroups) / ngroups,
                            labels = FALSE)) %>%
    dplyr::group_by(bin) %>%
    dplyr::mutate(r_lab = paste0('[', round(min(r), 2), ',', round(max(r), 2), ']')) %>%
    # Calculate loss ratio per bin
    dplyr::summarise(r_lab = r_lab[1],
                     loss_ratio = sum(amount) / sum(get(paste0(bench, '_prem'))),
                     sum_expo = sum(expo))

}
```

```
lrl_gbm_tree <- oos_pred %>% loss_ratio_lift(bench = 'tree',
                                             comp = 'gbm',
                                             ngroups = 5)
lrl_tree_gbm <- oos_pred %>% loss_ratio_lift(bench = 'gbm',
                                             comp = 'tree',
                                             ngroups = 5)
```

```
gridExtra::grid.arrange(
  lrl_gbm_tree %>% ggplot(aes(x = r_lab, y = loss_ratio)) +
    geom_bar(stat = 'identity') +
    ggtitle('comp: gbm / bench: tree'),

  lrl_tree_gbm %>% ggplot(aes(x = r_lab, y = loss_ratio)) +
     geom_bar(stat = 'identity') +
     ggtitle('comp: tree / bench: gbm')

  #ncol = 2
)
```


comp: gbm / bench: tree

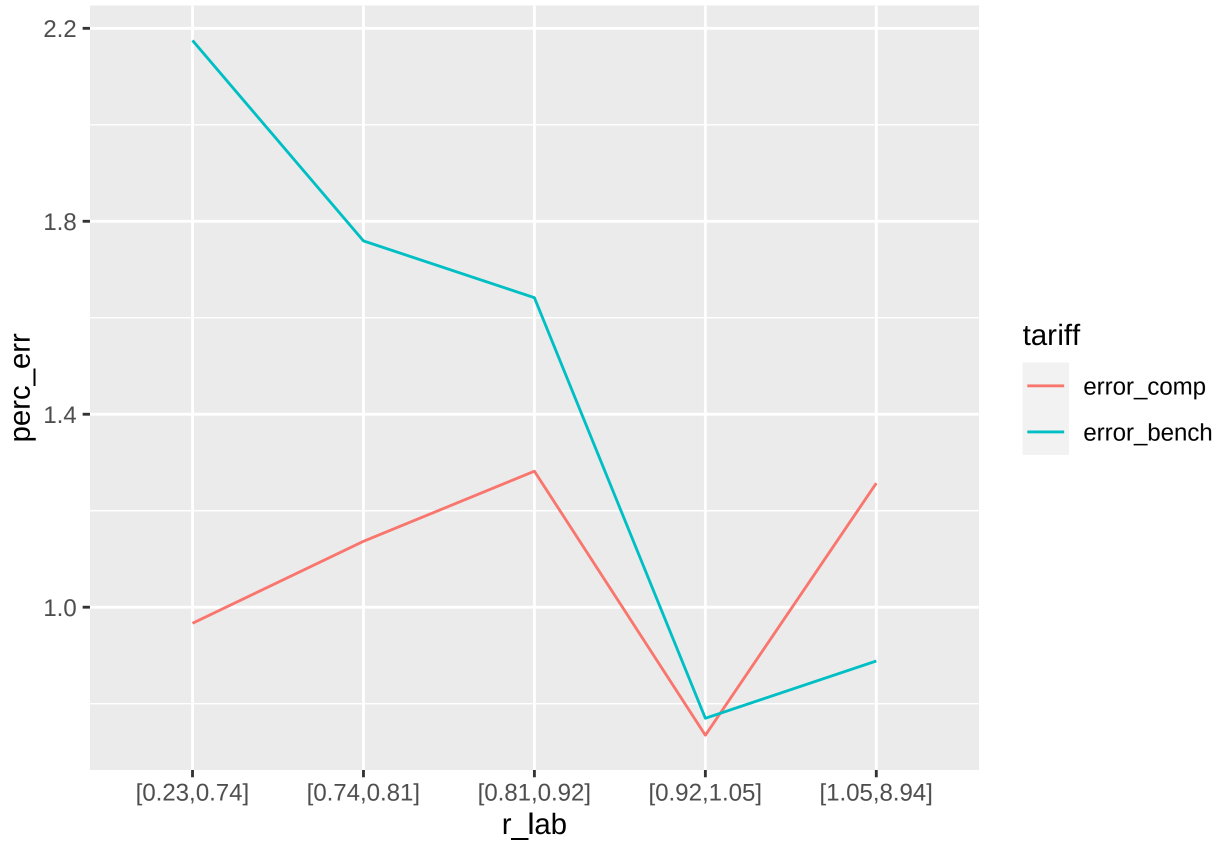

comp: tree / bench: gbm

```
### Double lift
#The double lift is assessed by applying the following steps:

#   + sort policies from smallest to largest relativity
# + bin the policies in groups of equal exposure
# + calculate the average loss amount and average premiums (comp & bench) in each bin
# + calculate the percentage error of premium (comp & bench) to loss in each bin

double_lift <- function(data, bench, comp, ngroups) {

  # Calculate relativity and sort from small to large
  data %>% dplyr::mutate(r = get(paste0(comp, '_prem')) / get(paste0(bench, '_prem'))) %>%
    dplyr::arrange(r) %>%
    # Bin in groups of equal exposure
    dplyr::mutate(bin = cut(cumsum(expo),
                            breaks = sum(expo) * (0:ngroups) / ngroups,
                            labels = FALSE)) %>%
    dplyr::group_by(bin) %>%
    dplyr::mutate(r_lab = paste0('[', round(min(r), 2), ',', round(max(r), 2), ']')) %>%
    # Calculate percentage errors for both tariffs
    dplyr::summarise(r_lab = r_lab[1],
                     error_comp = mean(get(paste0(comp, '_prem'))) / mean(amount) - 1,
                     error_bench = mean(get(paste0(bench, '_prem'))) / mean(amount) - 1,
                     sum_expo = sum(expo))

}
```

```
dbl_gbm_tree <- oos_pred %>% double_lift(bench = 'tree',
                                         comp = 'gbm',
                                         ngroups = 5)
dbl_gbm_tree
```

```
## # A tibble: 5 x 5
##     bin r_lab       error_comp error_bench sum_expo
##   <int> <chr>            <dbl>       <dbl>    <dbl>
## 1     1 [0.23,0.74]      0.967       2.17      842.
## 2     2 [0.74,0.81]      1.14        1.76      842.
## 3     3 [0.81,0.92]      1.28        1.64      843.
## 4     4 [0.92,1.05]      0.735       0.770     843.
## 5     5 [1.05,8.94]      1.26        0.888     843.
```

```
dbl_gbm_tree %>% reshape2::melt(id.vars = c('r_lab', 'bin' , 'sum_expo'),
                                value.name = 'perc_err',
                                variable.name = 'tariff') %>%
  ggplot(aes(x = r_lab, y = perc_err)) +
  geom_line(aes(group = tariff, colour = tariff))
```



```
### Gini index

#The last measure for economic lift that we analyze is the Gini index
#obtained from an ordered Lorenz curve. The function `gini()` from the `cplm` package allows to calcula
```

```
library(cplm)
```

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

## Loading required package: coda

## Warning: package 'coda' was built under R version 4.1.1

## Loading required package: Matrix

## Loading required package: splines
```

```
gini(loss = 'amount',
     score = paste0(c('tree',  'gbm'), '_prem'),
     data = as.data.frame(oos_pred))
```

```
##
## Call:
## gini(loss = "amount", score = paste0(c("tree", "gbm"), "_prem"),
##     data = as.data.frame(oos_pred))
##
## Gini indices:
##            tree_prem  gbm_prem
## tree_prem   0.000     12.138
## gbm_prem    1.475      0.000
##
## Standard errors:
##            tree_prem  gbm_prem
## tree_prem  0.000      3.677
## gbm_prem   3.731      0.000
##
## The selected score is gbm_prem.
```

```
#We can program the mini-max strategy explicitly as follows,
#which gives the ranking `gbm > rf > tree`:
gini(loss = 'amount',
     score = paste0(c('tree', 'gbm'), '_prem'),
     data = as.data.frame(oos_pred)) %>%
  slot('gini') %>%
  as.data.frame() %>%
  dplyr::mutate(max_gini = pmax(tree_prem, gbm_prem)) %>%
  dplyr::mutate(bench = c('tree', 'gbm')) %>%
  dplyr::arrange(max_gini)
```

```
##           tree_prem gbm_prem  max_gini bench
## gbm_prem   1.475497   0.0000  1.475497   gbm
## tree_prem  0.000000  12.1379 12.137903  tree
```