GBM_MTPL data

2022-10-26

Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors. In boosting, a random sample of data is selected, fitted with a model and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. With each iteration, the weak rules from each individual classifier are combined to form one, strong prediction rule.

1. Introduction

```
# 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\\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 ...
```

Quick data processing As some data have no predictive value by nature, like the contract number, we remove them from the database.

```r
dta <- subset(dta, select = -c(row.id, exposure ,drv.age, veh.age, clm.count))
```

```r
# Transformation into factors
dta$year<- as.factor(dta$year)
dta$nb.rb <- as.factor(dta$nb.rb)
dta$driver.gender <- as.factor(dta$driver.gender )
dta$marital.status <- as.factor(dta$marital.status)
dta$yrs.lic <- as.factor(dta$yrs.lic)
dta$ncd.level <- as.factor(dta$ncd.level)
dta$region <- as.factor(dta$region)
dta$body.code <- as.factor(dta$body.code)
dta$seats <- as.factor(dta$seats)
dta$fuel.type <- as.factor(dta$fuel.type)
```

```r
set.seed(54321) # reproducibility
# Create a stratified data partition
train_id <- caret::createDataPartition(
  y = dta$clm.incurred,
  # y = dta$clm.count/dta$exposure, # line used for claim frequency
  p = 0.7,
  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)
```

```
## 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
```

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

```
## numeric(0)
```

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

```
## numeric(0)
```

We distinguished 3 types of parameters.

1- the "Tree-specific", which affect each individual tree in the model 2- Boosting parameters, which drive the boosting method. 3- Parameter for the overall functionality

Basic Implementation

```r
library(gbm)
```

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

## Loaded gbm 2.1.8
```

```r
# for reproducibility
set.seed(1234)

# train GBM model
gbm.fit <- gbm(
```

```r
  formula = clm.incurred ~ .,
  distribution = "gaussian",
  data = dta_trn,
  n.trees = 10000,
  interaction.depth = 1, # a stump
  shrinkage = 0.001, # learning rate
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
  )
```

```r
# print results
print(gbm.fit)
```

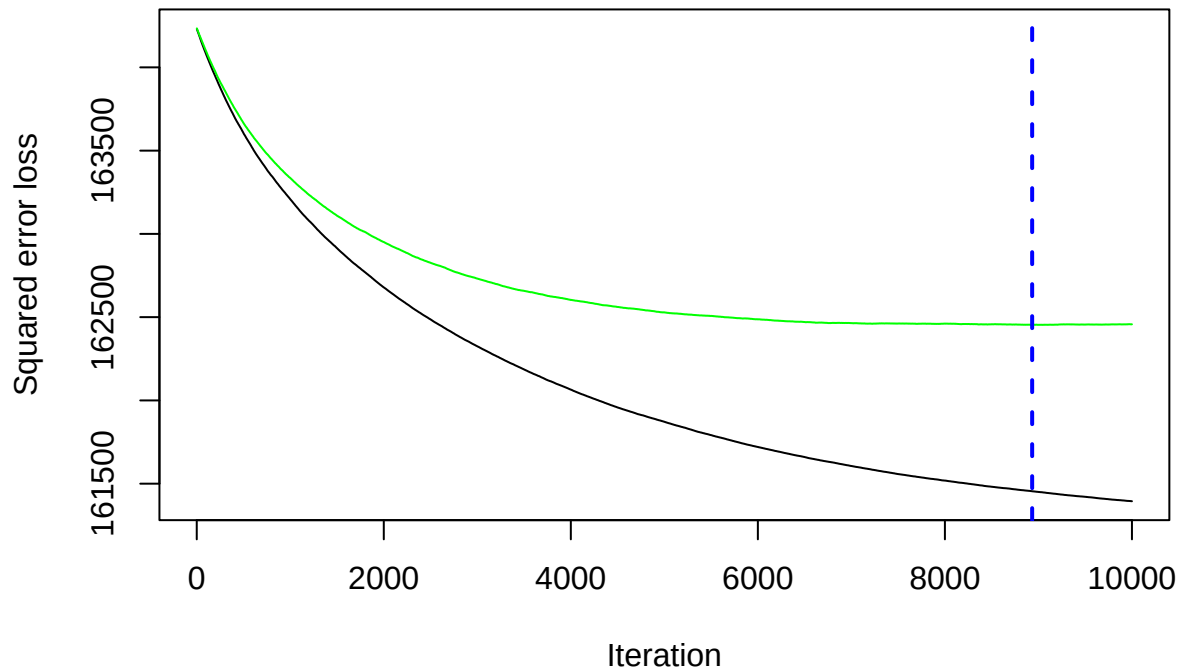
```
## gbm(formula = clm.incurred ~ ., distribution = "gaussian", data = dta_trn,
##     n.trees = 10000, interaction.depth = 1, shrinkage = 0.001,
##     cv.folds = 5, verbose = FALSE, n.cores = NULL)
## A gradient boosted model with gaussian loss function.
## 10000 iterations were performed.
## The best cross-validation iteration was 8933.
## There were 21 predictors of which 19 had non-zero influence.
```

The main idea of boosting is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.

```r
# get MSE and compute RMSE
sqrt(min(gbm.fit$cv.error))
```

```
## [1] 403.0555
```

```r
# plot loss function as a result of n trees added to the ensemble
gbm.perf(gbm.fit,plot.it = TRUE, method = "cv")
```




```
## [1] 8933
```

The minimum cross-validation RMSE is 408, meaning that on average the model is about $329 off the actual severity. It was achieved with 8933 trees. In the plot, the test-error is in green and the OOB error in black. We observe that the test-error stabilizes earlier than the OOB error.

2. Tuning

We tried to tune one parameter at the time. - Here we increase the learning rate from 0.01 to 0.1, - Reduce the number of tree, - Increase the depth of each tree from a stump (1 split) to 3 splits,

```r
# for reproducibility
set.seed(1234)

# train GBM model
gbm.fit2 <- gbm(
  formula = clm.incurred ~ .,
  distribution = "gaussian",
  data = dta_trn,
  n.trees = 5000,
  interaction.depth = 3,
  shrinkage = 0.1,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
```

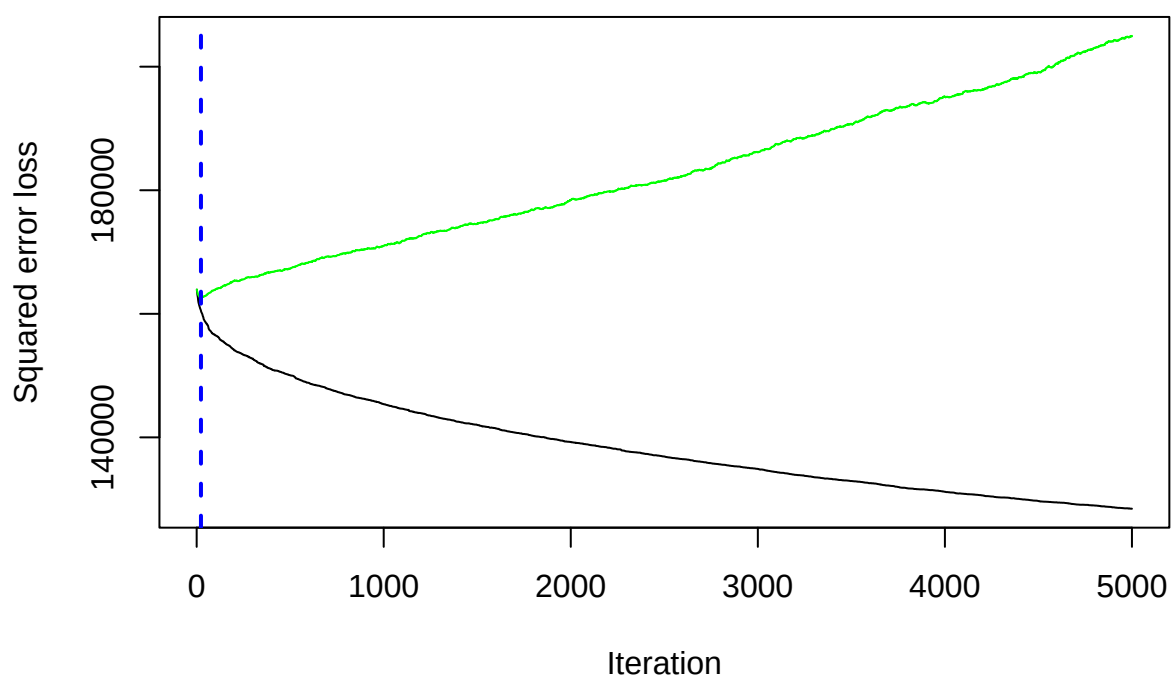
```
  verbose = FALSE
  )

# find index for n trees with minimum CV error
min_MSE <- which.min(gbm.fit2$cv.error)

# get MSE and compute RMSE
sqrt(gbm.fit2$cv.error[min_MSE])
```

```
## [1] 403.2263
```

```
# plot loss function as a result of n trees added to the ensemble
gbm.perf(gbm.fit2, method = "cv")
```




```
## [1] 23
```

We observe that the test error is increasing after 31 trees.

```
# for reproducibility
set.seed(1234)

# train GBM model
gbm.fit3 <- gbm(
  formula = clm.incurred ~ .,
```

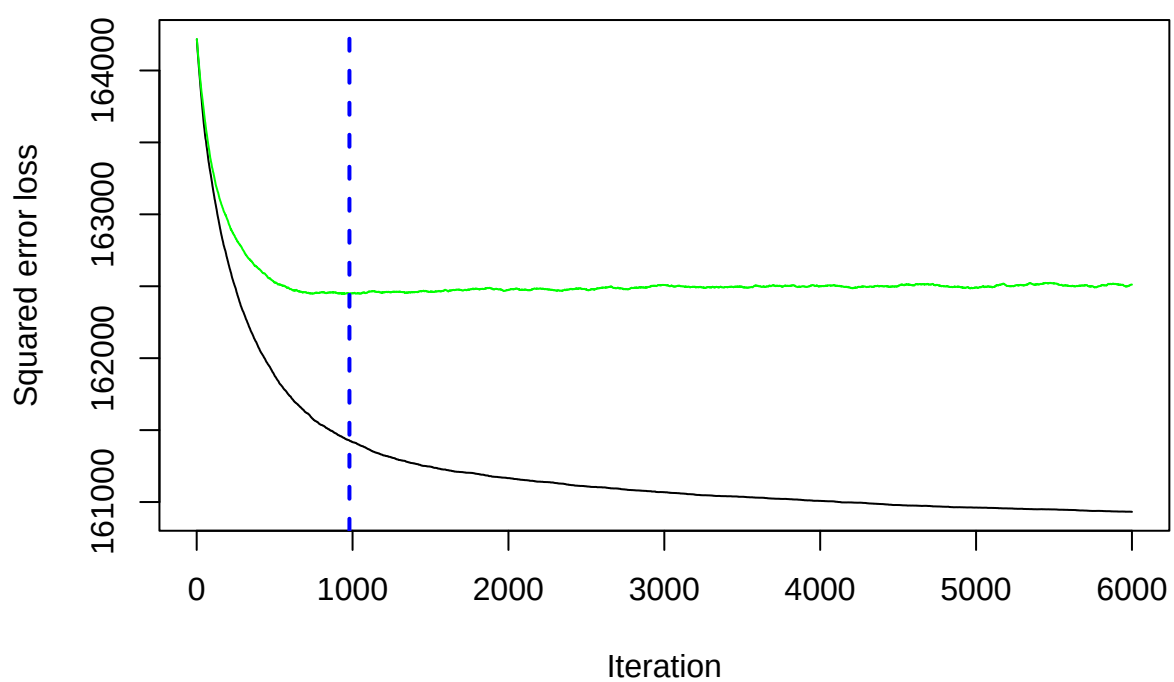
```r
  distribution = "gaussian",
  data = dta_trn,
  n.trees = 6000,
  interaction.depth = 1,
  shrinkage = 0.01,
  cv.folds = 5,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
  )
```

```r
# find index for n trees with minimum CV error
min_MSE <- which.min(gbm.fit3$cv.error)

# get MSE and compute RMSE
sqrt(gbm.fit3$cv.error[min_MSE])
```

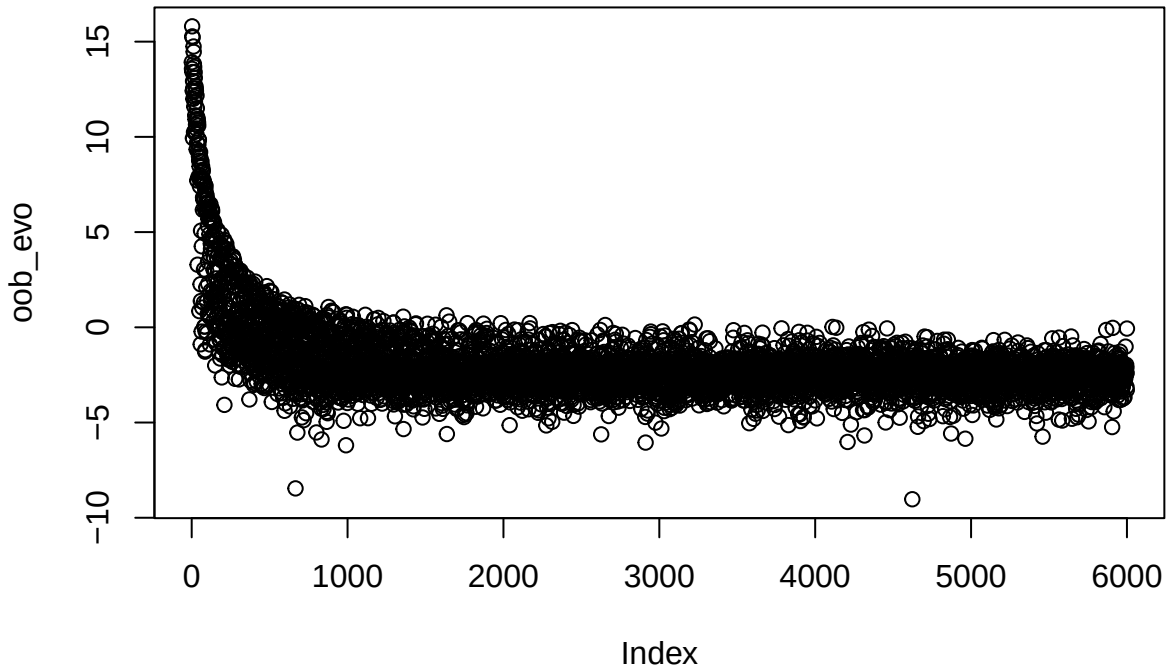
```
## [1] 403.0476
```

```r
# plot loss function as a result of n trees added to the ensemble
gbm.perf(gbm.fit3, method = "cv")
```

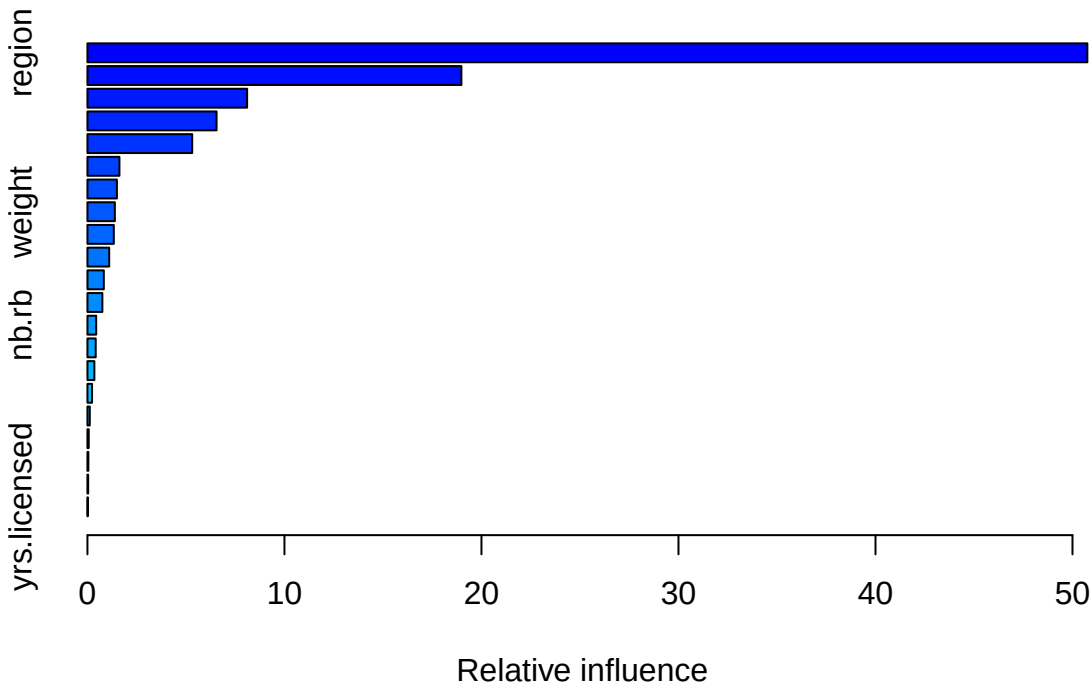



```
## [1] 980
```

```r
# Track the improvement of the OOB error
oob_evo <-gbm.fit3$oobag.improve
plot(oob_evo)
```




```r
summary(gbm.fit3)
```




```
##                            var     rel.inf
## region                  region 50.75468397
## driver.age          driver.age 18.98046769
## prior.claims      prior.claims  8.10455302
## year                      year  6.55423684
## ncd.level            ncd.level  5.31875460
## marital.status marital.status  1.62477185
## vehicle.value    vehicle.value  1.49511955
## weight                  weight  1.39157630
## height                  height  1.33823610
## width                    width  1.10492051
## yrs.lic                yrs.lic  0.83174441
## hp                          hp  0.75748668
## nb.rb                    nb.rb  0.44429224
## driver.gender    driver.gender  0.41952665
## body.code            body.code  0.35171339
## length                  length  0.23478699
## vehicle.age        vehicle.age  0.11978267
## ccm                        ccm  0.06529847
## fuel.type            fuel.type  0.04779190
## seats                    seats  0.03846115
## yrs.licensed      yrs.licensed  0.02179502
```

3. Creation of the hyperparameters grid search

Instead of tweaking the parameter manually, better to go with a grid search.

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(.01, .1),
  interaction.depth = c(1, 3),
  n.minobsinnode = c(5, 10),
  bag.fraction = c(.65, .8, 1),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0                     # a place to dump results
)

# total number of combinations
nrow(hyper_grid)
```

```
## [1] 24
```

Randomize data

```
# randomize data
random_index <- sample(1:nrow(dta_trn), nrow(dta_trn))
random_dta_train <- dta_trn[random_index, ]
```

Loop through the grid and store the results for each combination.

```
# grid search
for(i in 1:nrow(hyper_grid)) {

  # reproducibility
  set.seed(123)

  # train model
  gbm.tune <- gbm(
    formula =  clm.incurred~ .,
    distribution = "gaussian",
    data = random_dta_train,
    n.trees = 5000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = .75,
    n.cores = NULL, # will use all cores by default
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}
```

```r
hyper_grid %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

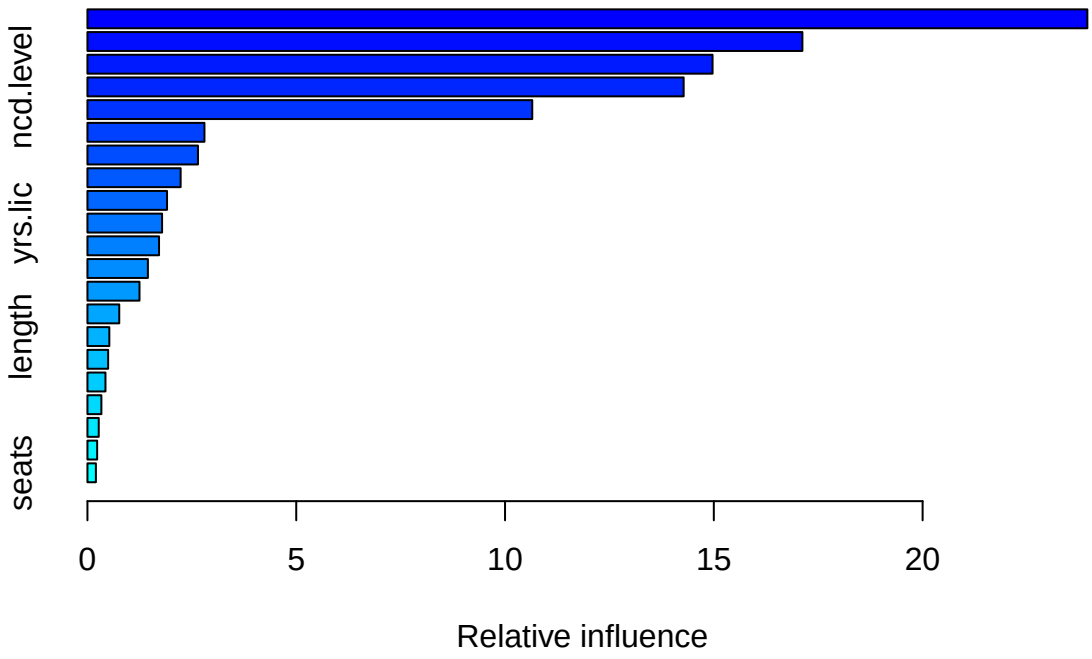
```
##    shrinkage interaction.depth n.minobsinnode bag.fraction optimal_trees
## 1       0.10                 3             10         1.00            52
## 2       0.01                 3             10         1.00           551
## 3       0.10                 1             10         0.80           252
## 4       0.10                 1              5         0.80           252
## 5       0.10                 1             10         0.65          3521
## 6       0.10                 1              5         1.00           204
## 7       0.10                 1             10         1.00           191
## 8       0.01                 1             10         1.00          2580
## 9       0.01                 1             10         0.80          4257
## 10      0.01                 3             10         0.80           434
##    min_RMSE
## 1  411.6794
## 2  411.7865
## 3  411.9252
## 4  411.9656
## 5  411.9773
## 6  411.9974
## 7  412.0055
## 8  412.0158
## 9  412.0169
## 10 412.0200
```

The best model has a high learning rate, 1 split and only 418 trees.

```r
# for reproducibility
set.seed(123)

# train GBM model
gbm.fit.final <- gbm(
  formula = clm.incurred ~ .,
  distribution = "gaussian",
  data = dta_trn,
  n.trees = 418,
  interaction.depth = 1,
  shrinkage = 0.1,
  n.minobsinnode = 10,
  bag.fraction = 0.8,
  train.fraction = 1,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
  )
```

```r
summary(gbm.fit.final)
```




```
##                            var    rel.inf
## region                  region 23.9449462
## year                      year 17.1211307
## driver.age          driver.age 14.9696873
## ncd.level            ncd.level 14.2772741
## prior.claims      prior.claims 10.6521708
## vehicle.value    vehicle.value  2.8017836
## width                    width  2.6466197
## marital.status marital.status  2.2318014
## weight                  weight  1.9062027
## yrs.lic                yrs.lic  1.7871893
## height                  height  1.7144628
## nb.rb                    nb.rb  1.4478046
## hp                          hp  1.2466836
## driver.gender    driver.gender  0.7615151
## length                  length  0.5245231
## fuel.type            fuel.type  0.4952281
## body.code            body.code  0.4314855
## vehicle.age        vehicle.age  0.3336953
## yrs.licensed      yrs.licensed  0.2707253
## ccm                        ccm  0.2325557
## seats                    seats  0.2025153
```

Save the final model

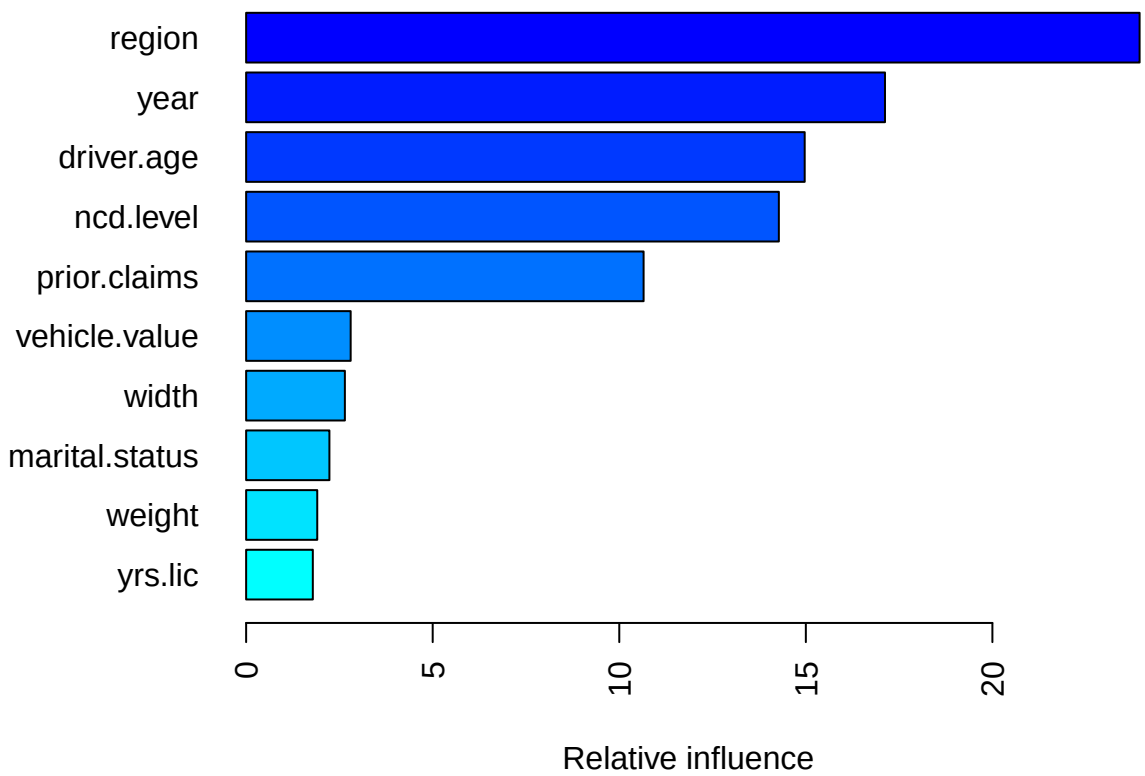
```
# save the model to disk
saveRDS(gbm.fit.final, "C:\\Users\\William.Tiritilli\\Documents\\Project P\\GBM\\GBM Tuning\\GBM_418_fi
```

Visualizing

A. Variable importance

We want to understand the variables that ahve the largest influence on the claim severity. The method "relative.influence" where at each split in each tree, the pacakge computes the improvement in the split-criterion and average the improvement mace by each variable across all the tree where the variable is used.

```
par(mar = c(5, 8, 1, 1))
summary(
  gbm.fit.final,
  cBars = 10,
  method = relative.influence, # also can use permutation.test.gbm
  las = 2
  )
```

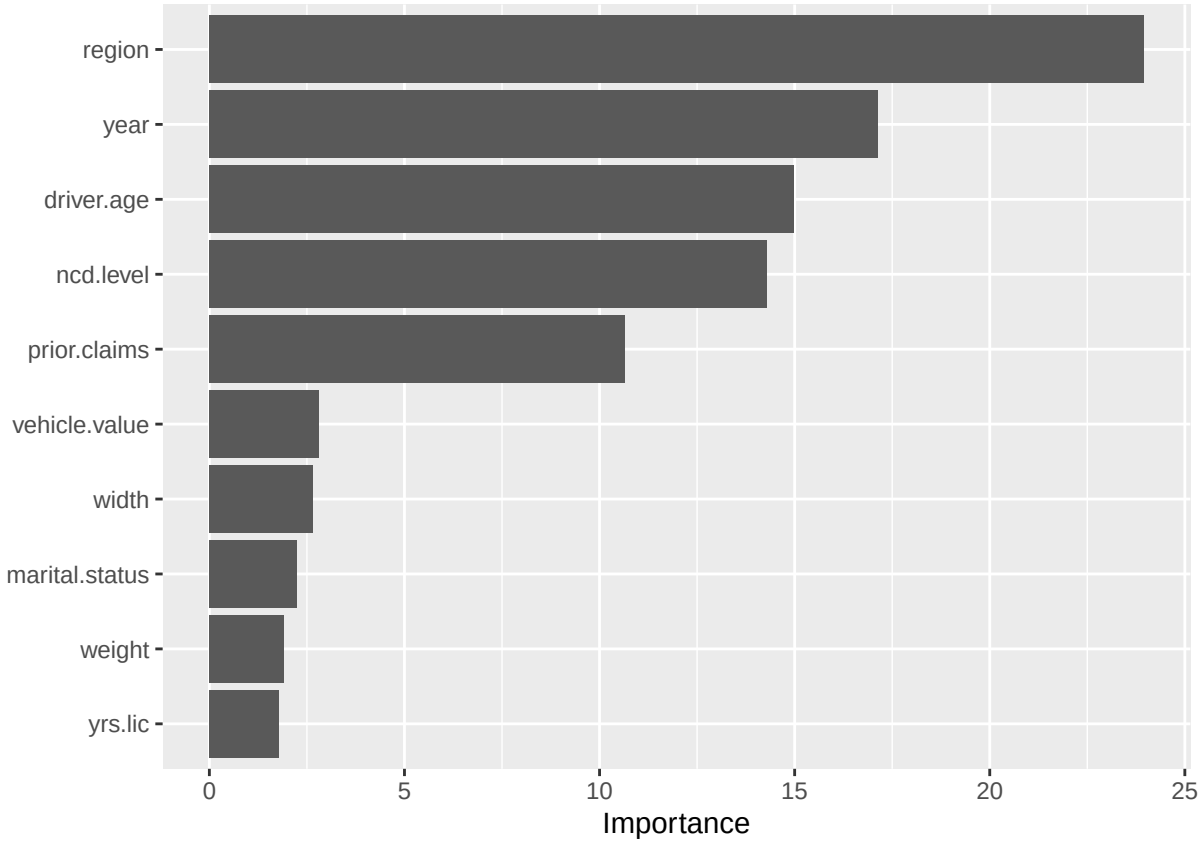



```
##                          var    rel.inf
## region                region 23.9449462
## year                    year 17.1211307
```

```
## driver.age         driver.age 14.9696873
## ncd.level           ncd.level 14.2772741
## prior.claims     prior.claims 10.6521708
## vehicle.value   vehicle.value  2.8017836
## width                   width  2.6466197
## marital.status marital.status  2.2318014
## weight                 weight  1.9062027
## yrs.lic               yrs.lic  1.7871893
## height                 height  1.7144628
## nb.rb                   nb.rb  1.4478046
## hp                         hp  1.2466836
## driver.gender   driver.gender  0.7615151
## length                 length  0.5245231
## fuel.type           fuel.type  0.4952281
## body.code           body.code  0.4314855
## vehicle.age       vehicle.age  0.3336953
## yrs.licensed     yrs.licensed  0.2707253
## ccm                         ccm  0.2325557
## seats                   seats  0.2025153
```

Region is the most important. It means that it got the largest average decrease in MSE among all the other variables.

```
# devtools::install_github("koalaverse/vip")
vip::vip(gbm.fit.final)
```



B. Partial Dependence Plot (PDP)

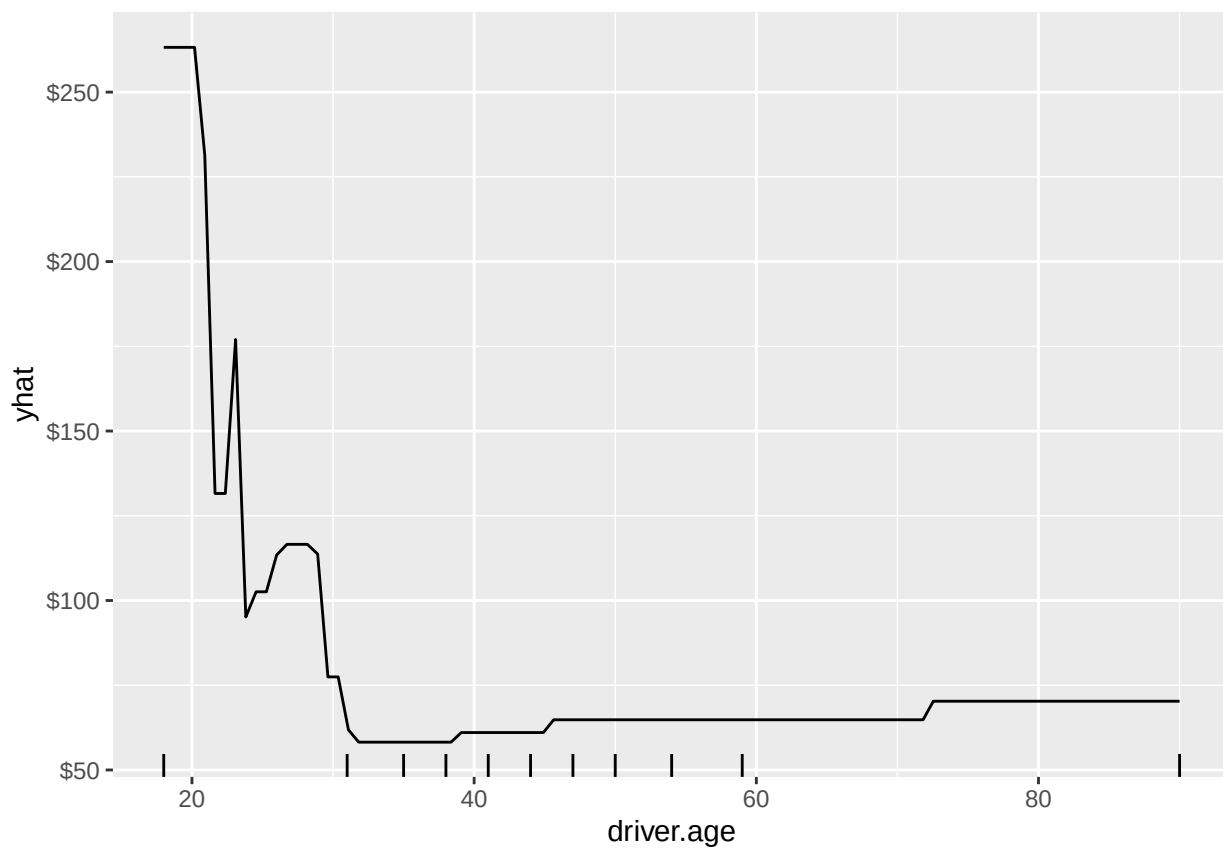
By definition, PDPs plot the change in the average predicted value as specified feature(s) vary over their marginal distribution.

Say differently, it shows the average change in predicted claim severity as we vary a variable, while holding everything else the same.

```
library(ggplot2)
```

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

```
gbm.fit.final %>%
  pdp::partial(pred.var = "driver.age", n.trees = gbm.fit.final$n.trees, grid.resolution = 100) %>%
  autoplot(rug = TRUE, train = dta_trn) +
  scale_y_continuous(labels = scales::dollar)
```



The average change in severity decreases as the driver's age increases.

C. Individual Conditional Expectation (ICE)

ICE curves are an extension of PDP plots but, rather than plot the average marginal effect on the response variable, we plot the change in the predicted response variable for each observation as we vary each predictor variable.

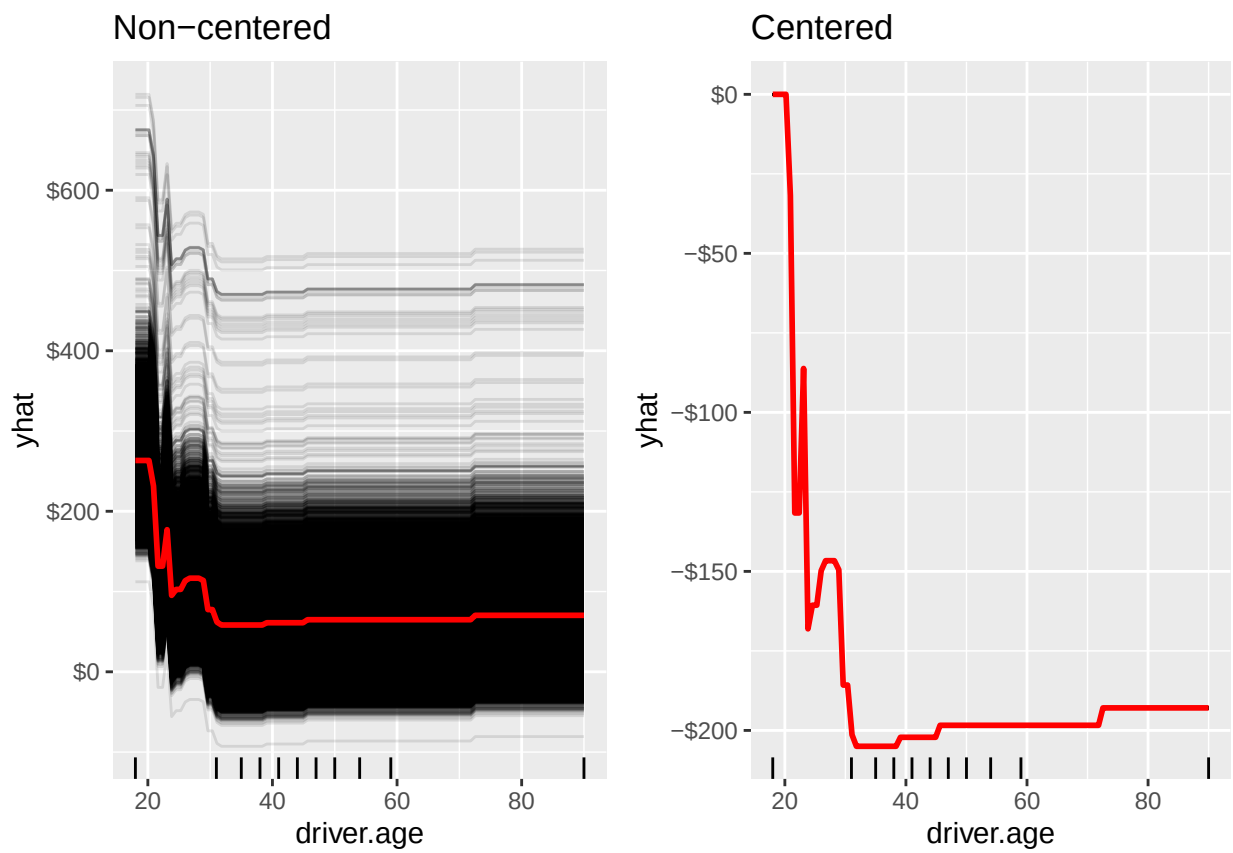
```r
ice1 <- gbm.fit.final %>%
  pdp::partial(
    pred.var = "driver.age",
    n.trees = gbm.fit.final$n.trees,
    grid.resolution = 100,
    ice = TRUE
    ) %>%
  autoplot(rug = TRUE, train = dta_trn, alpha = .1) +
  ggtitle("Non-centered") +
  scale_y_continuous(labels = scales::dollar)
```

```
## Warning in partial.default(., pred.var = "driver.age", n.trees =
## gbm.fit.final$n.trees, : Recursive method not available for "gbm" objects when
## `ice = TRUE`. Using brute force method instead.
```

```r
ice2 <- gbm.fit.final %>%
  pdp::partial(
    pred.var = "driver.age",
    n.trees = gbm.fit.final$n.trees,
    grid.resolution = 100,
    ice = TRUE
    ) %>%
  autoplot(rug = TRUE, train = dta_trn, alpha = .1, center = TRUE) +
  ggtitle("Centered") +
  scale_y_continuous(labels = scales::dollar)
```

```
## Warning in partial.default(., pred.var = "driver.age", n.trees =
## gbm.fit.final$n.trees, : Recursive method not available for "gbm" objects when
## `ice = TRUE`. Using brute force method instead.
```

```r
gridExtra::grid.arrange(ice1, ice2, nrow = 1)
```



D. LIME

```
model_type.gbm <- function(x, ...) {
  return("regression")
}

predict_model.gbm <- function(x, newdata, ...) {
  pred <- predict(x, newdata, n.trees = x$n.trees)
  return(as.data.frame(pred))
}
```

```
#install.packages("lime")
library(lime)
```

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

##
## Attaching package: 'lime'

## The following object is masked from 'package:dplyr':
##
##     explain
```

```r
# get a few observations to perform local interpretation on
local_obs <- dta_tst[56:57, ]

# apply LIME
explainer <- lime(dta_trn, gbm.fit.final)
```

```
## Warning: clm.incurred does not contain enough variance to use quantile binning.
## Using standard binning instead.
```

```r
explanation <- lime::explain(local_obs, explainer, n_features = 5)
plot_features(explanation)
```

The extract shows the prediction, \$26.6 and \$26.4, local model fit at 40%, and the most influential variable driving the predicted value for each.

```r
# predict values for test data
pred <- gbm::predict.gbm(object = gbm.fit.final, n.trees = gbm.fit.final$n.trees, dta_tst)

# results - Root Mean Square Error
caret::RMSE(pred, dta_tst$clm.incurred)
```

```
## [1] 406.6322
```

```r
# load the model
gbm.fit.final <- readRDS("C:\\Users\\William.Tiritilli\\Documents\\Project P\\GBM\\GBM Tuning\\GBM_418_
#print(super_model)
# make a predictions on "new data" using the final model
final_predictions <- gbm::predict.gbm(gbm.fit.final, dta_tst)
```

```
## Using 418 trees...
```