IFOA eXplainable AI Workstream - SHAP

Intro to case study

The aim of this document is to supply the reader of **All Clear** in The Actuary Magazine, with a practical, comprehensive example of applying SHAP to explain models' predictions.

The first section contains a short introduction to the dataset and EDA; in the next section I fit a Neural Network and a GLM (a tree based model will also be included soon for comparison); the third section contains output of SHAP and the final section holds additional technical information, with regards to both the used code and some mathematical background.

1 The dataset

1.1 Source

The dataset can be downloaded from: freMTPL2freq

Originally it was accessible through CASdatasets R package. Excerpt from the original documentation:

Description

In the two datasets freMTPL2freq, freMTPL2sev, risk features are collected for 677,991 motor third-part liability policies (observed mostly on one year). In addition, we have claim numbers by policy as well as the corresponding claim amounts. freMTPL2freq contains the risk features and the claim number while freMTPL2sev contains the claim amount and the corresponding policy ID.

Format

IDpol The policy ID (used to link with the claims dataset).
ClaimNb Number of claims during the exposure period.
Exposure The period of exposure for a policy, in years.
VehPower The power of the car (ordered values).
VehAge The vehicle age, in years.
DrivAge The driver age, in years (in France, people can drive a car at 18).
BonusMalus Bonus/malus, between 50 and 350: <100 means bonus, >100 means malus in France.
VehBrand The car brand (unknown categories).
VehGas The car gas, Diesel or regular.
Area The density value of the city community where the car driver lives in: from "A" for rural area to "F" for urban centre.
Density The density of inhabitants (number of inhabitants per square-kilometer) of the city where the car driver lives in.
Region The policy region in France (based on the 1970-2015 classification)

```r
if(FALSE){
  data_input = read.csv2("data/freMTPL2freq.csv",sep = ",") %>%
    as_tibble() %>%
    mutate(VehPower  = factor(VehPower, order = TRUE,levels = 4:15),
           across(c(Area,Region,VehBrand,VehGas),factor)) %>%
    rowwise() %>%
    mutate(ClaimNb = as.integer(min(ClaimNb,4)),
           VehAge = as.integer(min(VehAge,20)),
           DrivAge = as.integer(min(DrivAge,90)),
           Exposure = as.numeric(min(Exposure,1)),
           Density = log(Density)) %>%
    ungroup()
}
```

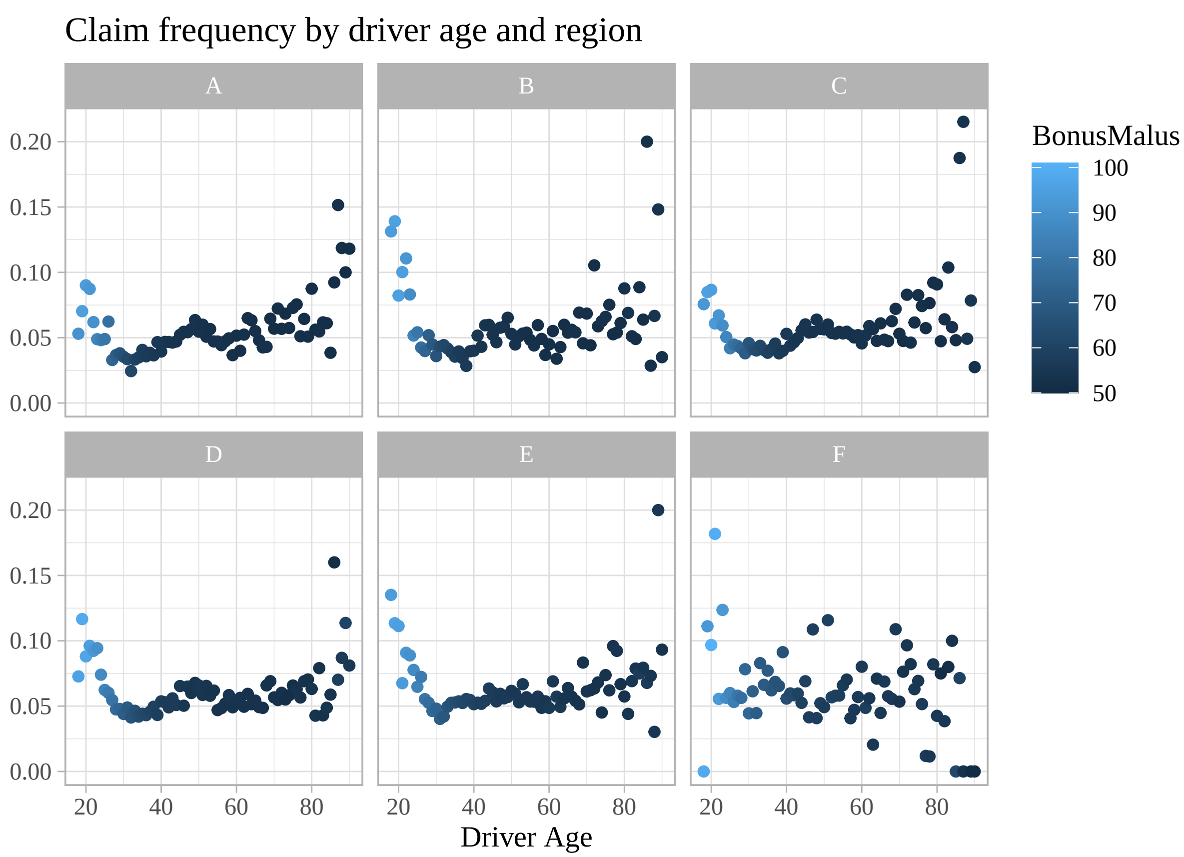
1.2 Preprocessing and Basic summary statistics

ClaimNb	Area	VehPower	VehAge	DrivAge	BonusMalus	VehBrand	VehGas	Density	Region
1	D	5	0	55	50	B12	Regular	7.104144	R82
1	D	5	0	55	50	B12	Regular	7.104144	R82
1	B	6	2	52	50	B12	Diesel	3.988984	R22
1	B	7	0	46	50	B12	Diesel	4.330733	R72
1	B	7	0	46	50	B12	Diesel	4.330733	R72
1	E	6	2	38	50	B12	Regular	8.007367	R31

1.3 EDA

```
data_input %>%
  group_by(DrivAge,Area) %>%
  summarise(Freq = mean(ClaimNb),
            BonusMalus = mean(BonusMalus)) %>%
  ggplot(aes(x = DrivAge, y=Freq, color = BonusMalus))+
  facet_wrap(~Area)+
  geom_point()+
  ggtitle("Claim frequency by driver age and region")+
  xlab("Driver Age")+
  ylab("")+
  theme_light()+
  theme(text=element_text(family="serif"),
        legend.justification = c("right", "top"))
```

```
## `summarise()` has grouped output by 'DrivAge'. You can override using the `.groups` argument.
```



2. Models - fitting, output comparison

In this section we build three models - a Neural Network, Gradient Boosted Trees (XGBoost), and a Poisson GLM.

We examine the models' outputs using a hold out test dataset (15% of available observations) and applying a model_evalulation function for additional statistics. The function lets us view: the obtained loss (MSE, MAE, Poisson Deviance); prediction statistics grouped by the actual claim number and raw predictions and more. Further details on model_evaluation function along with the source code can be found in section 4.1

Note: Poisson deviance loss is not symmetrical around the actual value we compare the prediction against (in contrast to e.g. MSE and MAE). It puts more emphasis on differentiating between zero and non-zero claim instances than say observations with 1 and 2 claims (see 4.4 for additional information).

2.1 NN

```
# notrun
if(FALSE){

# don't enter next epoch if there are no significant performance gains
early_stop = callback_early_stopping(monitor = "val_loss", patience = 3)

# Neural network structure
Neural_Net = keras_model_sequential(input_shape = c(ncol(train))) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'relu')

# Fitting the model
Neural_Net %>%
  compile(
    loss = "poisson",
    optimizer = optimizer_rmsprop()
  ) %>%
  fit(
    callbacks = list(early_stop),
    train %>% as.matrix(),
    (data[sample_TTS,"ClaimNb"]/data_input$Exposure[sample_TTS]) %>% as.matrix(),
    batch_size = 2^13,
    epochs = 10,
    validation_split = 0.1,
    shuffle = TRUE ,
    sample_weight = data_input$Exposure[sample_TTS] %>% as.matrix()
  )

eval_list$Neural_Net = model_evalulation(Neural_Net,
                                     cbind(test,data[-sample_TTS,"ClaimNb"]),
                                     type = "NN",
                                     ClaimNBadj = FALSE)
}

summary(Neural_Net)
```

```
## Model: "sequential_13"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense_56 (Dense)                    (None, 64)                      3712
## ________________________________________________________________________________
## dense_55 (Dense)                    (None, 64)                      4160
## ________________________________________________________________________________
## dense_54 (Dense)                    (None, 32)                      2080
## ________________________________________________________________________________
## dense_53 (Dense)                    (None, 1)                       33
## ================================================================================
## Total params: 9,985
## Trainable params: 9,985
## Non-trainable params: 0
## ________________________________________________________________________________
```

2.2 XGB

To be done

2.3 GLM

```
if(TRUE){

  GLM = glm(formula = ClaimNb/Exposure ~ Area + VehPower + VehAge + DrivAge + BonusMalus + VehBrand + Ve
            family = poisson(link = log),
            weights = Exposure,
            data = data_input[sample_TTS,] %>% select(-IDpol))  %>% suppressWarnings()

}

GLM
```

```
##
## Call:  glm(formula = ClaimNb/Exposure ~ Area + VehPower + VehAge + DrivAge +
##     BonusMalus + VehBrand + VehGas + Density + Region, family = poisson(link = log),
##     data = data_input[sample_TTS, ] %>% select(-IDpol), weights = Exposure)
##
## Coefficients:
##   (Intercept)         AreaB         AreaC         AreaD         AreaE
##    -4.0114467     0.0161407     0.0005032     0.0339460     0.0184801
##         AreaF    VehPower.L    VehPower.Q    VehPower.C    VehPower^4
##    -0.0737318    -0.0502667    -0.1834956    -0.0409452    -0.0923431
##    VehPower^5    VehPower^6    VehPower^7    VehPower^8    VehPower^9
##     0.0140356    -0.2462388    -0.1482789     0.0890247     0.0124442
##   VehPower^10   VehPower^11        VehAge       DrivAge    BonusMalus
##    -0.0867967     0.1070066    -0.0404711     0.0063530     0.0225429
##   VehBrandB10   VehBrandB11   VehBrandB12   VehBrandB13   VehBrandB14
##     0.0173827     0.1143393     0.1727967     0.0193933    -0.0694485
##    VehBrandB2    VehBrandB3    VehBrandB4    VehBrandB5    VehBrandB6
##    -0.0132280    -0.0146900    -0.0223253     0.0444672    -0.0211478
## VehGasRegular       Density     RegionR21     RegionR22     RegionR23
##     0.0668401     0.0378664     0.1720110     0.0322257    -0.0821447
##     RegionR24     RegionR25     RegionR26     RegionR31     RegionR41
##     0.0738775     0.0304650    -0.0142948    -0.1117022    -0.2737856
##     RegionR42     RegionR43     RegionR52     RegionR53     RegionR54
##    -0.0228858    -0.0797646    -0.0230057     0.0716439    -0.0464823
##     RegionR72     RegionR73     RegionR74     RegionR82     RegionR83
##    -0.0833502    -0.0988507     0.1921085     0.0682227    -0.2679160
##     RegionR91     RegionR93     RegionR94
##    -0.0254702     0.0077099     0.1421034
##
## Degrees of Freedom: 576310 Total (i.e. Null);  576258 Residual
## Null Deviance:       190400
## Residual Deviance: 184100    AIC: Inf
```

2.4 Model comparison

Here we examine the model_evalulation outputs, starting with basic loss functions and then turning to more in-depth view and predictions statistics.

2.4.1 Loss Metrics

Neural Network

absolute_error	mean_squared_error	ExposureWeightedPoissonDevianceLoss
0.136935	0.0613371	34750.78

XGB

GLM

absolute_error	mean_squared_error	ExposureWeightedPoissonDevianceLoss
0.1493149	0.0619182	36053.18

2.4.2 Broader summary statistics

As an alternative to comparing the loss metrics, we can assess the models performance using summary statistics grouped by the actual claim number. It can be noted that the average predictions for the Network are higher than for the GLM for policies with ClaimNb>0 and lower for ClamNb=0 which would suggest a better fit. Furthermore, the

Neural Network

Actual	count	mean_pred	sd_pred	min	max	Q1	Q2	Q3
0	96495	0.095	0.078	0.000	1.016	0.050	0.074	0.111
1	4924	0.138	0.131	0.000	0.978	0.061	0.090	0.163
2	271	0.159	0.145	0.006	0.734	0.067	0.104	0.197
3	9	0.242	0.236	0.062	0.775	0.098	0.139	0.226
4	3	0.122	0.036	0.083	0.154	0.107	0.130	0.142

XGB

GLM

Actual	count	mean_pred	sd_pred	min	max	Q1	Q2	Q3
0	96495	0.107	0.058	0.023	3.381	0.073	0.094	0.123
1	4924	0.122	0.075	0.034	1.504	0.079	0.103	0.139
2	271	0.149	0.133	0.042	1.182	0.088	0.116	0.158
3	9	0.215	0.172	0.055	0.600	0.112	0.155	0.244
4	3	0.113	0.074	0.058	0.197	0.071	0.084	0.141

3. XAI

Note that the current implementation of SHAP is insensitive to One-Hot Encoding, meaning that explanation of an instance might contain a non-zero contribution for both VehGas=Diesel AND VehGas!=Regular at the same time. This is caused by the lack of interface in the SHAP function to indicate the range of OHE variables.

Currently to apply SHAP in R, one has to access a Python environment that has SHAP package installed. Here, DALEX package is used as a wrapper for SHAP. Alternatively a work in progress version of a pure R implementation is available - shapr package v0.2.0 .

For the full code producing graphs below go to: scripts/Prep_SHAP_Graphs.R, where you will find a wrapper for the original plotting function, to allow for selecting only the most significant variables. It will prove helpful in case the models use a large number of variables and/or including OHE.

The code below is an example of how to call DALEX and SHAP in R. Here we estimate the SHAP values on a subset of 1000 observations. SHAP for the XGB and GLM are produced analogously.

```
if(FALSE){
  explainer = list()
  residuals = list()
  SHAP = list()

  row_subset = sample(1:nrow(test),1000)

  explainer$Neural_Net = explain(
    model = Neural_Net,
    data = test[row_subset,],
    y = (data[-sample_TTS,"ClaimNb"] %>% as.matrix())[row_subset],
    predict_function = predict_wrapper,
    label = "Neural_Net"
  )

  SHAP$Neural_Net_1 = shap(explainer$Neural_Net,
                           new_observation = test[678,],
                           method = "KernelSHAP"
  )

  SHAP$Neural_Net_1_plot = CustomSHAPplot(dalex_output = SHAP$Neural_Net_1)
}
```

3.1 Individual SHAP

Neural Network Individual SHAP value plots for observations: with 1 claim and low error for NN; an instance with 0 claims and very high error
An instance with 2 claims

For this observation ClaimNb = 1, we can see (as in most cases) that the BonusMalus variable along with vehicle age and brand/gas type have the most impact on the final prediction.

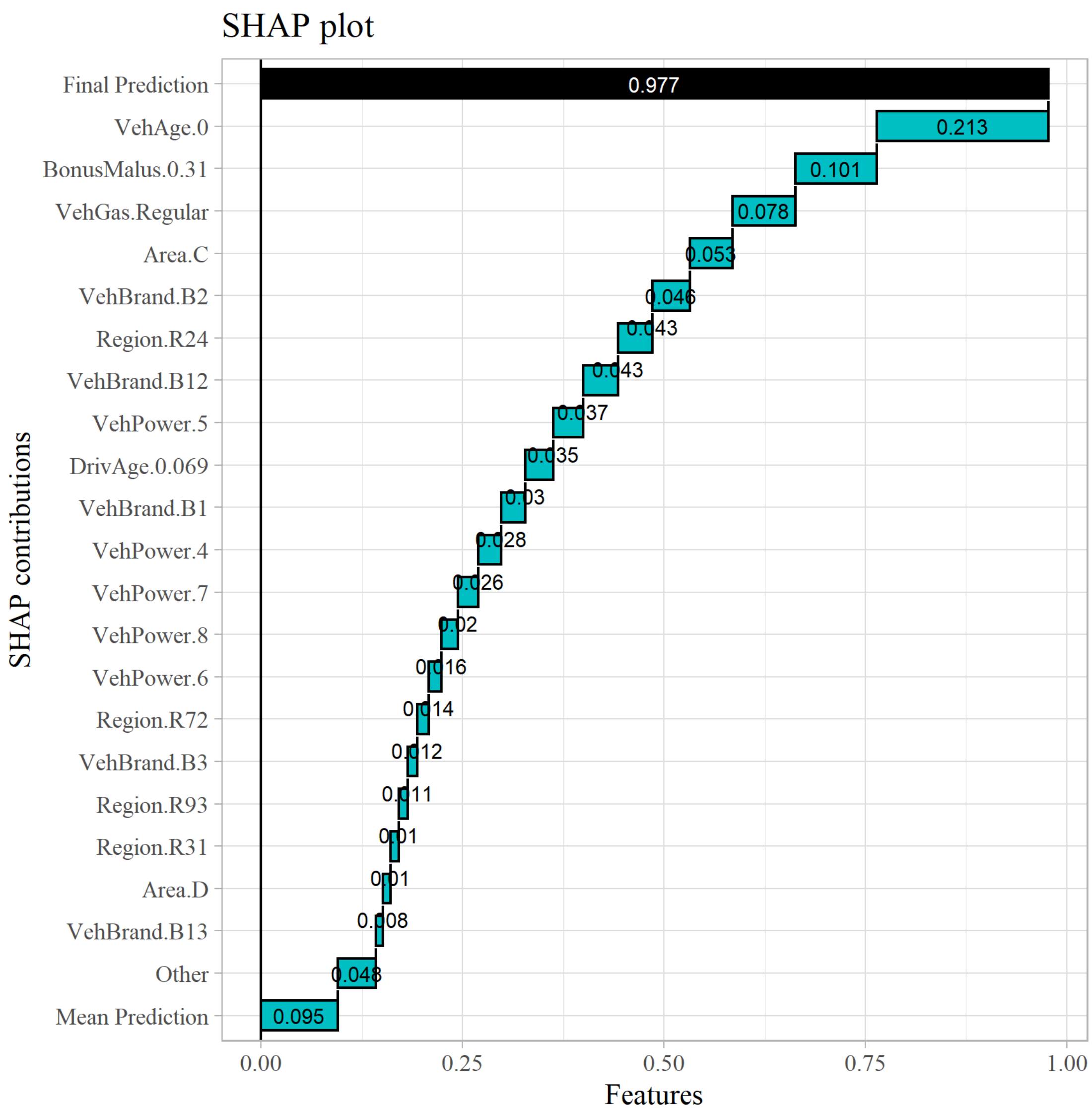


Figure 1: Individual SHAP

This instance, in reality has ClaimNb of 0 and Networks prediction for it, has the highest error from all observations with actual ClaimNb=0. Interestingly, if it wasn't for the BonusMalus value in case of this instance, the model could have output a much lower prediction. This could be confirmed using an ICE (Individual Conditional Expectation) plot for this observation.

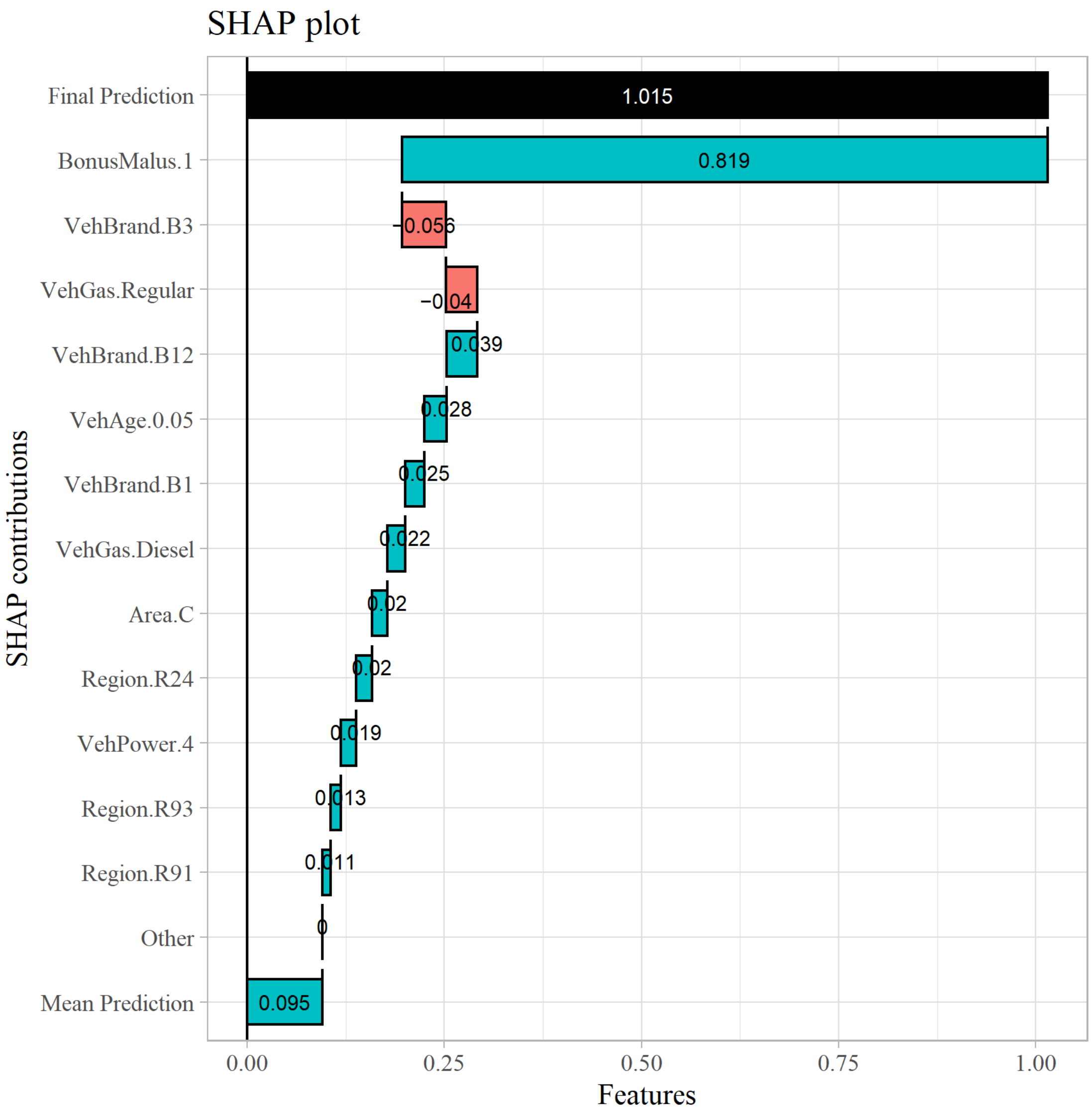


Figure 2: Individual SHAP

In case of this observation, the actual number of claims=2, and here again BonusMalus plays a siginficant role in the final prediction.

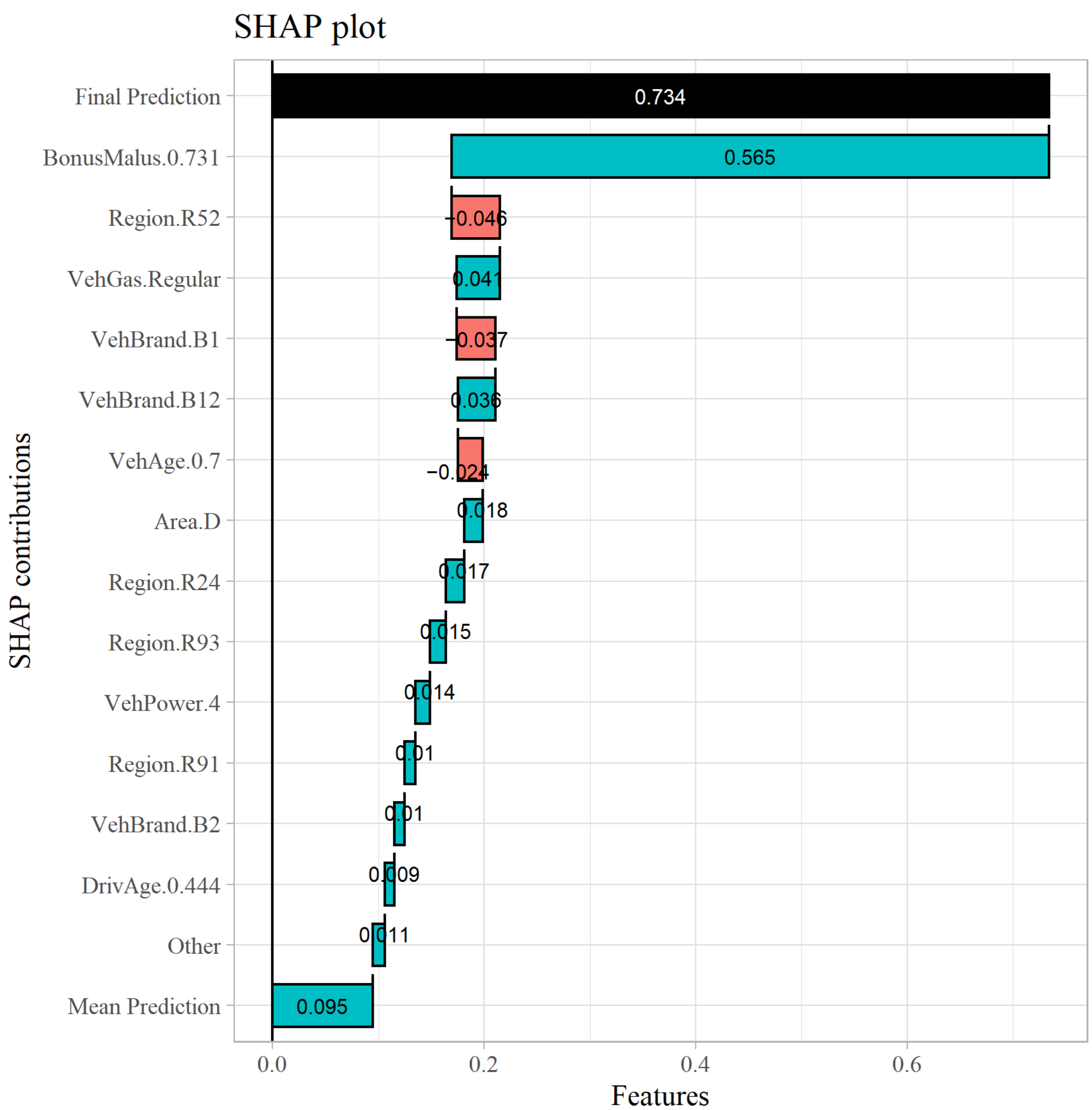


Figure 3: Individual SHAP

3.2 Model - level SHAP

to be done

placeholder:

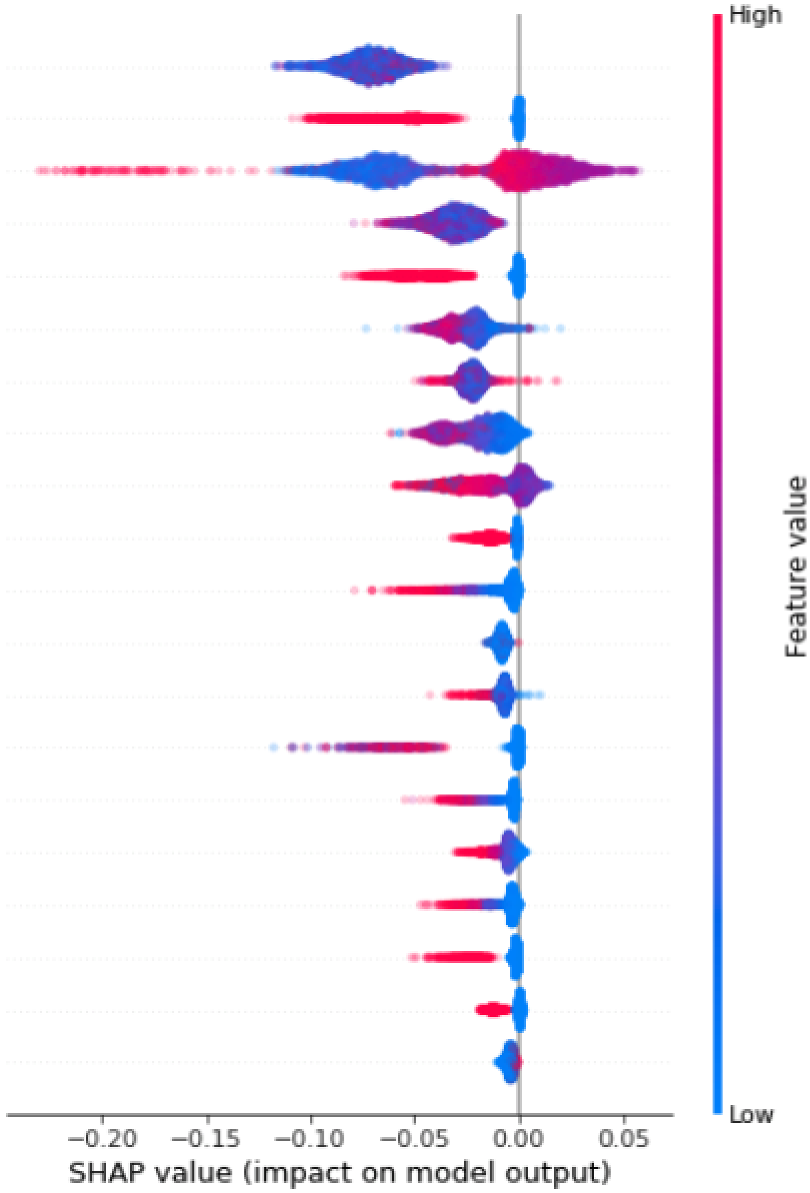


Figure 4: Individual SHAP

source

4. Additional Information

4.1 model_evaluation utility function

Utility function for analysing model performance.

Input:
model: KERAS Neural Network, XGBoost, GLM
type: specify the type of model
data: named dataframe to assess models' performance on
ClaimNBadj: We might want to fit a model on transformed y h(y) which would require h^-1 transformation before obtianing comparable outputs

Output:
list(Evaluation = Evaluation, Predictions = Predictions, AvE = AvE, Sorted_by_MSE = Sorted_by_MSE))

```
if (FALSE){


  # utility function for analysing model performance
  model_evalulation = function(model,
                               data = test,
                               type = "NN",
                               ClaimNBadj = FALSE){
    if (type=="NN"){

      if( ClaimNBadj==TRUE){

        # rescaled Predictions
        Predictions = data.frame(Predicted = (model %>% predict(data %>% select(-ClaimNb) %>% as.matrix
                                 Actual = data$ClaimNb)
      }else{

        Predictions = data.frame(Predicted = (model %>% predict(data %>% select(-ClaimNb) %>% as.matrix
                                 Actual = data$ClaimNb)
      }

    }else if(type=="GLM"){

      Predictions = data.frame(Predicted = predict(model,newdata = data %>% select(-ClaimNb,-Exposure,-
                               Actual = data$ClaimNb)

    }else if(type == "XGB"){

      Predictions = data.frame(Predicted = predict(model,newdata = data %>% select(-ClaimNb) %>% as.mat
                               Actual = data$ClaimNb)
    }

    Evaluation = data.frame(
      absolute_error = mean(abs(as.matrix(Predictions$Predicted - data$ClaimNb))),
      mean_squared_error = mean(as.matrix((Predictions$Predicted - data$ClaimNb)^2)),
      ExposureWeightedPoissonDevianceLoss = ExposureWeightedPoissonDevianceLoss(Predictions$Predicted,da
    )
```

```
    AvE = Predictions %>% mutate(Actual = as.factor(Actual)) %>%
      group_by(Actual) %>%
      summarise(count = n(),
                mean_pred = mean(Predicted),
                sd_pred = sd(Predicted),
                min = min(Predicted),
                max = max(Predicted),
                Q1 = quantile(Predicted,probs = 0.25),
                Q2 = quantile(Predicted,probs = 0.5),
                Q3 = quantile(Predicted,probs = 0.75),
                IQR = (Q3-Q1)/Q2,
                Negative_Pred = sum(Predicted<0))


    Sorted_by_MSE = data %>% mutate(Predictions = Predictions$Predicted,
                                    SquaredError = (Predictions - ClaimNb)^2) %>% arrange(-SquaredError)

    return(list(Evaluation = Evaluation,
                Predictions = Predictions,
                AvE = AvE,
                Sorted_by_MSE = Sorted_by_MSE))
  }

  # for DALEX and SHAP explainer objects
  predict_wrapper=function(model,new_data){
    return(model %>% predict(new_data %>% as.matrix()))
  }

  predict_wrapper_GLM=function(model,new_data){
    return(predict(model,newdata = new_data,type="response"))
  }



}
```

4.2

Data wrangling was not the main intention of this exercise, thus the dataset in Prep_RMD/XAI_data.rda has been already processed. The script below takes the raw freMTPL data as in the source and transforms it to our needs.

```
if (FALSE){

  data_input=read.csv2("data/freMTPL2freq.csv",sep = ",") %>%
    as_tibble() %>%
    mutate(VehPower  = factor(VehPower, order = TRUE,levels = 4:15),
           across(c(Area,Region,VehBrand,VehGas),factor)) %>%
    rowwise() %>%
    mutate(ClaimNb = as.integer(min(ClaimNb,4)),
           VehAge = as.integer(min(VehAge,20)),
           DrivAge = as.integer(min(DrivAge,90)),
```

```
          Exposure = as.numeric(min(Exposure,1)),
          Density = log(Density)) %>%
    ungroup()

  # remove ID's
  data = data_input %>% select(-c(IDpol,Exposure))

  # train/test split
  sample_TTS = sample(x = 1:nrow(data),size = round(0.85 * nrow(data)),replace = FALSE)

  train = data[sample_TTS,] %>% as.data.frame()
  test = data[-sample_TTS,] %>% as.data.frame()

  # Normalization of numeric variables
  data_prep = function(data){

    Normalized = apply(data %>% select(-ClaimNb) %>% select_if(Negate(is.factor)),
                       2,
                       FUN = function(x){return((x-min(x))/(max(x)-min(x)))}) %>%
      as_tibble()

    data = data %>% select_if(is.factor) %>% cbind(Normalized)

    data$VehPower = factor(data$VehPower, order = FALSE)

    # OHE of factor variables
    OHE = dummyVars("~.", data = data %>% select_if(is.factor)) %>%
      predict(newdata = data %>% select_if(is.factor)) %>%
      as_tibble()

    data = data %>% select_if(Negate(is.factor)) %>% cbind(OHE) %>% as_tibble()

    return(data)
  }

  # data prep separately for train and test
  train = data_prep(train)
  test = data_prep(test)

}
```

4.3

text

```
if (FALSE){

  # Customized and corrected SHAP plot
  CustomSHAPplot=function(dalex_output,
                          epsilon = 0.007){

    colnames(dalex_output) = str_replace_all(colnames(dalex_output),pattern = "_",replacement = "")
```

```
    # supply values in names for continuous variables
    dalex_output$vname[dalex_output$vname=="VehAge"]=paste("VehAge",round(dalex_output$VehAge[1],3),sep
    dalex_output$vname[dalex_output$vname=="DrivAge"]=paste("DrivAge",round(dalex_output$DrivAge[1],3),
    dalex_output$vname[dalex_output$vname=="BonusMalus"]=paste("BonusMalus",round(dalex_output$BonusMal

    dalex_output = dalex_output %>% as_tibble()

    avg_pred = dalex_output$yhatmean
    total_pred = dalex_output$yhat
    other_pred = sum(dalex_output$attribution[abs(dalex_output$attribution)<=epsilon])

    plot_data = rbind(data.frame(values = c(avg_pred[1],other_pred),
                                 labels = c("Mean Prediction","Other")),
                      dalex_output %>%
                        as_tibble() %>%
                        filter(abs(attribution) > epsilon) %>%
                        mutate(attribution = attribution) %>%
                        transmute(values = attribution,
                                  labels = vname) %>%
                        arrange(abs(values))) %>%
      mutate(values = round(values,3))

    plt = plot_data %>%
      waterfall(fill_by_sign = TRUE,
                calc_total = TRUE,
                total_axis_text = "Final Prediction")+
      coord_flip()+
      ggtitle("SHAP plot")+
      xlab("SHAP contributions")+
      ylab("Features")+
      theme_light()+
      theme(text=element_text(family="serif"),
            legend.justification = c("right", "top"))

    return(plt)
  }

}
```

4.4 Poisson Deviance Loss

```
ExposureWeightedPoissonDevianceLoss = function(predicted,
                                               actual,
                                               exposure=NULL){

  output = rep(NA,length(actual))

  if (is.null(exposure)){
    exposure = rep(1,length(actual))
  }

  # actual zero
```

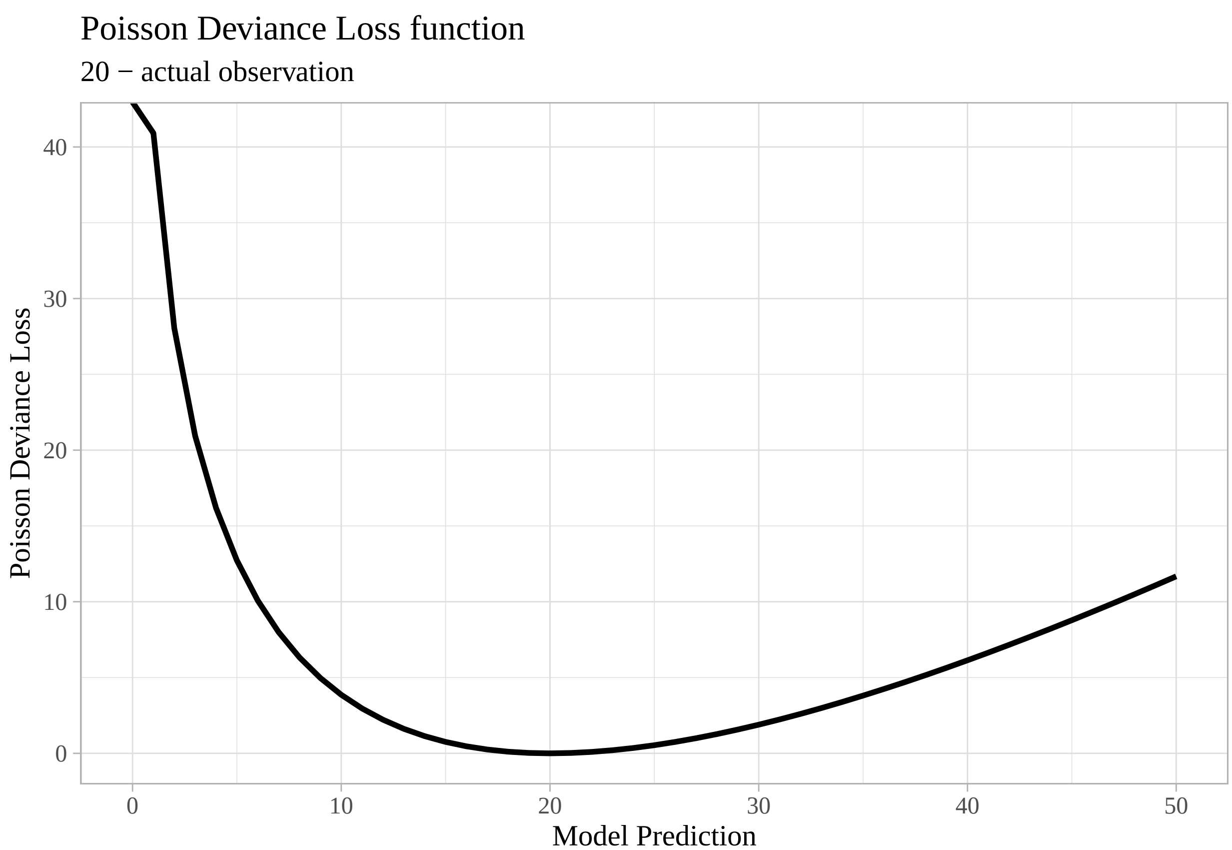
```
  k = (actual == 0)
  output[k] = 2*predicted[k]*exposure[k]

  # actual non zero and predicted non-zero
  m = (actual != 0 & predicted != 0 )
  output[m] = 2*actual[m]*(predicted[m]*exposure[m]/actual[m] - 1 - log(predicted[m]*exposure[m]/actual

  # actual non-zero and predicted zero
  n =  (actual != 0 & predicted == 0 )
  output[n] = 2*actual[n]

  output = sum(output)

  return(output)
}
```



4.5 Session information

Hardware, R and Python configuration

```
sessionInfo()
```

```
## R version 4.0.4 (2021-02-15)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
## system code page: 1250
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] xgboost_1.5.0.2  skimr_2.1.3      keras_2.7.0      tensorflow_2.7.0
##  [5] reticulate_1.18  shapper_0.1.3    DALEX_2.2.0      caret_6.0-86
##  [9] lattice_0.20-41  forcats_0.5.1    stringr_1.4.0    dplyr_1.0.7
## [13] purrr_0.3.4      readr_1.4.0      tidyr_1.1.4      tibble_3.1.6
## [17] ggplot2_3.3.5    tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-152        fs_1.5.0            lubridate_1.7.10
##  [4] httr_1.4.2          repr_1.1.3          tools_4.0.4
##  [7] backports_1.2.1     utf8_1.2.1          R6_2.5.0
## [10] rpart_4.1-15        DBI_1.1.1           colorspace_2.0-0
## [13] nnet_7.3-15         withr_2.4.1         tidyselect_1.1.0
## [16] compiler_4.0.4      cli_3.1.0           rvest_1.0.0
## [19] xml2_1.3.2          labeling_0.4.2      scales_1.1.1
## [22] tfruns_1.5.0        rappdirs_0.3.3      digest_0.6.27
## [25] rmarkdown_2.7       base64enc_0.1-3     pkgconfig_2.0.3
## [28] htmltools_0.5.1.1   highr_0.8           dbplyr_2.1.1
## [31] rlang_0.4.10        readxl_1.3.1        rstudioapi_0.13
## [34] farver_2.1.0        generics_0.1.0      jsonlite_1.7.2
## [37] ModelMetrics_1.2.2.2 magrittr_2.0.1     Matrix_1.3-2
## [40] Rcpp_1.0.6          munsell_0.5.0       fansi_0.4.2
## [43] lifecycle_1.0.0     stringi_1.5.3       whisker_0.4
## [46] pROC_1.17.0.1       yaml_2.2.1          MASS_7.3-53
## [49] plyr_1.8.6          recipes_0.1.17      grid_4.0.4
## [52] crayon_1.4.1        haven_2.3.1         splines_4.0.4
## [55] hms_1.0.0           zeallot_0.1.0       knitr_1.31
## [58] pillar_1.6.4        reshape2_1.4.4      codetools_0.2-18
## [61] stats4_4.0.4        reprex_2.0.0        glue_1.4.2
## [64] evaluate_0.14       data.table_1.14.0   modelr_0.1.8
## [67] vctrs_0.3.8         foreach_1.5.1       cellranger_1.1.0
## [70] gtable_0.3.0        assertthat_0.2.1    xfun_0.22
## [73] gower_0.2.2         prodlim_2019.11.13  broom_0.7.10
## [76] class_7.3-18        survival_3.2-7      timeDate_3043.102
## [79] iterators_1.0.13    lava_1.6.9          ellipsis_0.3.2
## [82] ipred_0.9-12
```

Py: version: 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] Architecture: 64bit numpy_version: 1.19.2