Peeking into the Black Box

An Actuarial Case Study for Interpretable Machine Learning

Christian Lorentzen
christian.lorentzen@mobiliar.ch

Michael Mayer
michael.mayer@mobiliar.ch

Prepared for:
Fachgruppe "Data Science"
Swiss Association of Actuaries SAV

Version of May 7, 2020

This tutorial gives an overview of tools for explaining and interpreting black box machine learning models like boosted trees or deep neural networks. All our methods are illustrated on a publicly available real car insurance data set on claims frequencies.

Keywords. XAI, machine learning, explainability, interpretability, black box models, model-agnostic technique, flashlight, motor insurance, claims frequency.

1 Introduction and overview

This study has been carried out for the working group "Data Science" of the Swiss Association of Actuaries SAV, see

https://www.actuarialdatascience.org

The purpose of this tutorial is to provide an overview of *interpretable machine learning* and *explainable artificial intelligence* (XAI) from a practical, actuarial perspective applied to supervised learning.

Let us start with commonly used definitions of these terms.

"Interpretable Machine Learning refers to methods and models that make the behavior and predictions of machine learning systems understandable to humans." — Christoph Molnar [28]

“Explainable AI (XAI) is the class of systems that provide visibility into how an AI system makes decisions and predictions and executes its actions.”
— Arun Rai [32]

In the context of machine learning (ML), both terms “interpretable machine learning” and “XAI” describe tools to explain predictions and behavior of ML models and we will thus use them interchangeably. Furthermore, like in [28], we do not distinguish between model *interpretability* and *explainability*.

Modern ML models like tree-based ensemble methods and artificial neural networks are celebrated for their excellent predictive performance. Without much tedious, manual interference like feature engineering, they learn representations of the training data in an automated way that usually—if done right—generalize well to new, unseen data. As they use many parameters and have a complex internal structure, they face one major accusation: being a *black box*.

Instead of discussing whether or not the black box accusation is true, let us imagine a situation where we only have access to the predictions of a particular model, but not to its internals. This way, even a linear model from ordinary least squares becomes a black box. Here, we want to show that, even in this situation, it is possible to analyze and explain important aspects of the model such as the importance of a variable, feature effects and the contribution of each feature to single predictions. Such model explanations are essential in order

- to gain confidence and trust in a model,
- to uncover model limitations,
- to improve a model by guiding through the modeling process, and
- for the model to serve as a source of information and to validate hypotheses—instead of only serving as a prediction machine.

It is the aim of interpretable ML and XAI to provide such *model agnostic* tools[1], i.e. tools that work for any kind of supervised modeling technique, be it a linear regression, a neural network, a tree-based method or even an average of multiple such models. The only requirement is the availability of a numeric prediction function, i.e. a function that takes a data set and returns predictions. An excellent reference on this topic is the online book [28] by Molnar.

This seems to be the right place to caution against the in-production-use of any model that is not understood (by the user or modeler), is untested and produces unstable or unexplainable results. A few examples of model failure “due to what humans would consider as cheating rather than valid problem-solving” are given in [19].

[1] XAI also provides model *specific* tools, in particular for neural networks, see [30, 19].

Generalized Linear Models and XAI One important model family, especially within the actuarial profession, is the well established, good old generalized linear model (GLM), see [27]. Besides their mathematical and computational tractability as well as their stability, cf. [35, Chapter 13] and [26], two predominant reasons for the success story of GLMs, in the authors' view, are:

(A) They are easy to interpret.

(B) They are flexible.

GLMs are *easy to interpret* thanks to their parametric form that allows us to read model parameters as effects on the response variables. Furthermore, suitable test statistics help to distinguish important from unimportant covariables. GLMs are *flexible* due to the possibility to include interaction terms, variable transformations and the use of non-linear terms such as squares or splines. Curiously, (A) and (B) have opposite effects: The more terms we include, the harder it is to interpret model parameters and to judge variable importance. In the extreme case, a complex GLM can be as "black box" and powerful as a modern ML algorithm like a neural network or a boosted tree. Accordingly, tools from XAI might become a valuable asset for interpretation not only of modern ML models but also of complex GLMs, blurring the borders between classic and modern modeling techniques.

Outline The tutorial is structured as follows: In Sections 2 and 3, we will introduce the data set as well as the actuarial frequency models. Section 4 then introduces the most important model agnostic tools to elicit global model properties such as model performance, variable importance, effects, and interaction strength. In Section 5, we will briefly explain tools to investigate *local* properties of the model, i.e. the behavior of the model for single predictions. Finally, in Section 6, we will show how modern ML models spiced with XAI can help to improve and guide the construction of GLMs with good performance. The complete R code can be found at `https://github.com/JSchelldorfer/ActuarialDataScience`.

We will consider only tools for tabular data and not for unstructured input like images, text, audio or video. These areas have their own XAI tools.

Configuration All R code was run on the following system.

- OS: Microsoft Windows 10 Pro
- Processor: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 2112 Mhz, 4 Core(s)
- R version: 3.6.3
- Python version (Keras/TensorFlow backend): 3.6.10

Specific package versions are indicated in the code.

Note: Since version 3.6.0, R uses a different random number generator. Results are not reproducible under older versions.

2 Data and descriptive statistics

Throughout this tutorial, we consider the data set `freMTPL2freq` included in the R package **CASdatasets** [5], and model claims frequency $\texttt{Freq} = \frac{\texttt{ClaimNb}}{\texttt{Exposure}}$ similar to [31]. It contains information for 678 013 motor third-party liability policies collected over one year, e.g. the age of the driver and the vehicle power. Overall, 36 102 claims occurred over a total exposure period of 358 499 years, yielding an overall frequency of 10 %.

Listing 1 shows the first nine rows of the data.

Listing 1: output of command `head(freMTPL2freq, 9)`

```
> head(freMTPL2freq, 9)
  IDpol ClaimNb Exposure Area VehPower VehAge DrivAge BonusMalus VehBrand  VehGas Density Region
1     1       1     0.10    D        5      0      55         50      B12 Regular    1217    R82
2     3       1     0.77    D        5      0      55         50      B12 Regular    1217    R82
3     5       1     0.75    B        6      2      52         50      B12  Diesel      54    R22
4    10       1     0.09    B        7      0      46         50      B12  Diesel      76    R72
5    11       1     0.84    B        7      0      46         50      B12  Diesel      76    R72
6    13       1     0.52    E        6      2      38         50      B12 Regular    3003    R31
7    15       1     0.45    E        6      2      38         50      B12 Regular    3003    R31
8    17       1     0.27    C        7      0      33         68      B12  Diesel     137    R91
9    18       1     0.71    C        7      0      33         68      B12  Diesel     137    R91
```

According to `?freMTPL2freq`, the meaning of the columns is as follows:

- `IDpol`: Policy identifier used to join another data set in the package
- `ClaimNb`: Number of claims during exposure period
- `Exposure`: Exposure in years
- `Area`: The area code
- `VehPower`: The power of the car (categorical)
- `VehAge`: The vehicle age (years)
- `DrivAge`: The driver age
- `BonusMalus`: Bonus malus level (low is good)
- `VehBrand`: Car brand (unknown levels)
- `VehGas`: Diesel or regular
- `Density`: Number of inhabitants per km2 where the driver lives
- `Region`: Standard classification of French policy region

For all our models, we use the preprocessing shown in Listing 2—mostly clipping of large feature values. Furthermore, we add a variable `group_id` identifying rows that are identical except for `IDpol`, `ClaimNb` and `Exposure`. In the rows shown in Listing 1, `IDpol` 1 and 3 form such a group. The largest group in the data set contains 22 identical rows (up to `IDpol`)[2]. We suspect that those identical rows are not independent. We will

[2] This largest group has even identical values for `Exposure` and `ClaimNb`.

use this `group_id` later to allow for more honest model evaluation without leakage across training and test data sets.

Listing 2: Code for data preprocessing

```
>library(CASdatasets)  # 1.0.6
>library(dplyr)        # 0.8.5

>data(freMTPL2freq)
>head(freMTPL2freq)

# Grouping id
>distinct <- freMTPL2freq %>%
>  distinct_at(vars(-c(IDpol, Exposure, ClaimNb))) %>%
>  mutate(group_id = row_number())

# Preprocessing
>dat <- freMTPL2freq %>%
>  left_join(distinct) %>%
>  mutate(Exposure = pmin(1, Exposure),
>         Freq = pmin(15, ClaimNb / Exposure),
>         VehPower = pmin(12, VehPower),
>         VehAge = pmin(20, VehAge),
>         VehGas = factor(VehGas),
>         DrivAge = pmin(85, DrivAge),
>         logDensity = log(Density),
>         VehBrand = factor(VehBrand, levels =
>                    paste0("B", c(12, 1:6, 10, 11, 13, 14))),
>         PolicyRegion = relevel(Region, "R24"),
>         AreaCode = Area)
```

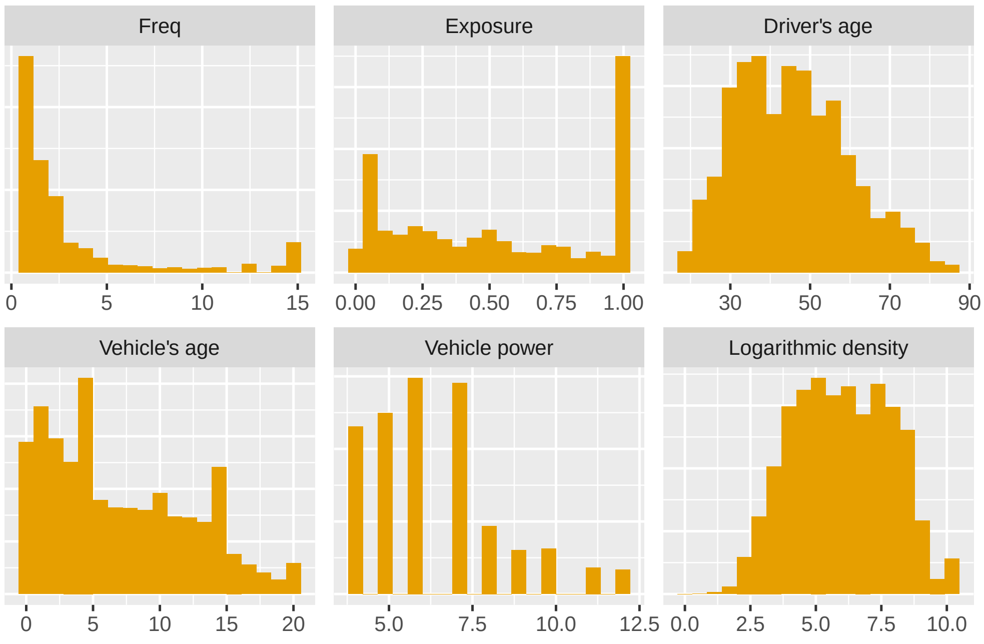


Figure 1: Histograms of selected variables.

Figure [1] shows histograms of the response variable `Freq` conditional on `Freq >= 1`, the exposure period `Exposure` as well as some numerical covariables.

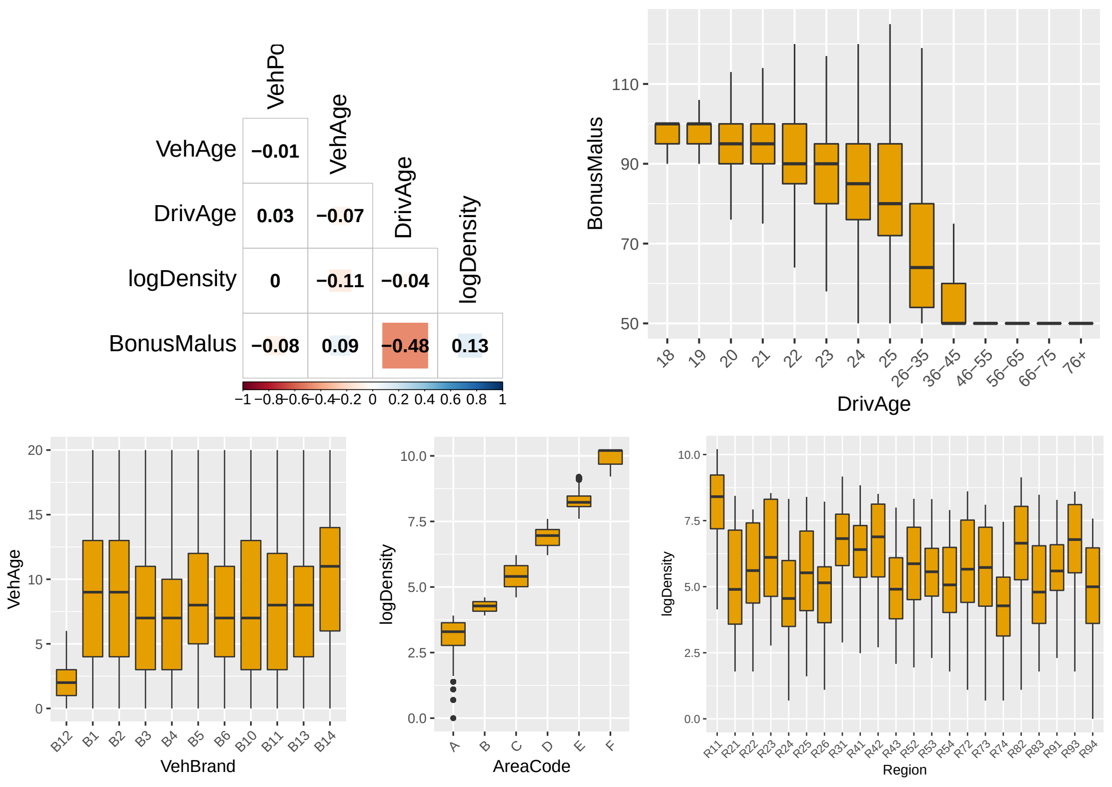


Figure 2: Top left: Correlogram between selected numerical variables. Top right: Boxplots of bonus malus level by driver's age. Bottom left: Boxplots of vehicle's age by brand. Bottom middle: Boxplots of log-density against area code. Bottom right: Boxplots of log-density against policy region. Note that outliers have been hidden for readability, except for the plot of log-density against area code.

Figure [2] visualizes selected associations between covariables. Pearson correlations between most numerical variables are low[3]. Because of the complex relationship between driver's age and the `BonusMalus` level, we do not consider the latter in our models for the sake of simplicity. The area codes seem to be a discrete version of population density, so we may drop it without any loss of information. A more detailed description of the data set can be found in [31].

Listing [3] defines variable names of covariables (`x`), the response (`y`), as well as the exposure (`w`).

Listing 3: Relevant variables for modelling

```
# Covariables, Response, Weight
>x <- c("VehPower", "VehAge",  "VehBrand", "VehGas", "DrivAge",
```

[3] We would have expected a stronger positive correlation between age of driver and age of car.

```
>        "logDensity", "PolicyRegion")
>y <- "Freq"
>w <- "Exposure"

# Save data
>save(x, y, w, dat, file = "rdata/prep.RData")
```

3 Regression models

For the illustration of model agnostic techniques, we will use a classic "white box" approach as well as two black box approaches to model claims frequency. All models optimize the `Exposure` weighted Poisson deviance, see Eq. (1).

1. Poisson GLM without interactions. Driver's age as well as vehicle's age are both modeled as natural cubic splines in order to well represent their effects.

2. Deep neural network fitted by the R interface to Keras and TensorFlow [7] using a model architecture adapted from [34]. The model represents the two categorical input variables `VehBrand` and `Region` as one-dimensional embeddings. The model uses three hidden layers with contracting number of nodes (20-15-10) each followed by a hyperbolic tangent activation, leading to a total number of trainable parameters of 679. The output layer is mapped to the frequency response by exponentiation, i.e. the inverse link of the Poisson model is used. Training was done in batch sizes of 10 000 and for 300 epochs using Nesterov Adam optimizer [9] with default settings and a learning rate of 0.002. Slight changes in the layer layout, using different activation functions, adding dropout, and using a different optimizer did not lead to noticeable better five-fold cross-validation performance on the training set. In order to remove overall mean bias on the training data, as suggested in [10], a Poisson GLM is trained on top of the ten dimensional output of the last hidden layer.

3. Gradient boosted trees using XGBoost [6]. Parameters were chosen by five-fold cross-validation on the training set, yielding parameters as set in Listing 4.

Data split The full data set is split into 80 % training and 20 % test rows. Sampling was done by groups, i.e. keeping all rows with the same `group_id` either in the training or the test data in order to minimize bias and to select reasonable parameters for the black box models. The code to fit the models is shown in Listing 4. It may serve as a template for weighted regression with other loss functions, e.g. Gamma loss for severity models with `ClaimNb` as weights.

Listing 4: Code to fit the three models

```
>library(dplyr)      # 0.8.5
>library(splines)    # 3.6.3
>library(splitTools) # 0.2.0
>library(xgboost)    # 1.0.0.2
>library(keras)      # 2.2.5.0 / tensorflow: 2.0.0 / Python 3.6
```

```
# Load data
>load("rdata/prep.RData", verbose = TRUE)

# Stratified split
>ind <- partition(dat[["group_id"]], p = c(train = 0.8, test = 0.2),
>                  seed = 22, type = "grouped")
>train <- dat[ind$train, ]
>test <- dat[ind$test, ]

# GLM: Using "quasipoisson" avoids warning about non-integer response.
#  Has no impact on coefficients/predictions
>fit_glm <- glm(Freq ~ VehPower + ns(VehAge, 5) + VehBrand +
>                  VehGas + ns(DrivAge, 5) + logDensity + PolicyRegion,
>                data = train,
>                family = quasipoisson(),
>                weights = train[[w]])

#  GBM
>prep_xgb <- function(dat, x) {
>  data.matrix(dat[, x, drop = FALSE])
>}

>dtrain <- xgb.DMatrix(prep_xgb(train, x),
>                       label = train[[y]],
>                       weight = train[[w]])

# Parameters chosen by 5-fold grouped CV
>params_freq <- list(learning_rate = 0.2,
>                     max_depth = 5,
>                     alpha = 3,
>                     lambda = 0.5,
>                     max_delta_step = 2,
>                     min_split_loss = 0,
>                     colsample_bytree = 1,
>                     subsample = 0.9)

# Fit
>set.seed(1)
>fit_xgb <- xgb.train(params_freq,
>                      data = dtrain,
>                      nrounds = 580,
>                      objective = "count:poisson",
>                      watchlist = list(train = dtrain),
>                      print_every_n = 10)

# Neural net

# Input list maker
>prep_nnet <- function(dat, x, cat_cols = c("PolicyRegion", "VehBrand")) {
>  dense_cols <- setdiff(x, cat_cols)
>  c(list(dense1 = data.matrix(dat[, dense_cols])),
>    lapply(dat[, cat_cols], function(z) as.integer(z) - 1))
>}

# Initialize neural net
>new_neural_net <- function() {
>  k_clear_session()
>  set.seed(1)
>  tensorflow::tf$random$set_seed(0)
>
>  # Model architecture
>  dense_input <- layer_input(5, name = "dense1", dtype = "float32")
>  PolicyRegion_input <- layer_input(1, name = "PolicyRegion", dtype = "int8")
>  VehBrand_input <- layer_input(1, name = "VehBrand", dtype = "int8")
>
>  PolicyRegion_emb <- PolicyRegion_input %>%
>    layer_embedding(22, 1) %>%
>    layer_flatten()
>
>  VehBrand_emb <- VehBrand_input %>%
>    layer_embedding(11, 1) %>%
>    layer_flatten()
>
>  outputs <- list(dense_input, PolicyRegion_emb, VehBrand_emb) %>%
>    layer_concatenate() %>%
>    layer_dense(20, activation = "tanh") %>%
>    layer_dense(15, activation = "tanh") %>%
>    layer_dense(10, activation = "tanh") %>%
>    layer_dense(1, activation = "exponential")
>
>  inputs <- list(dense1 = dense_input,
>                 PolicyRegion = PolicyRegion_input,
>                 VehBrand = VehBrand_input)
>
>  model <- keras_model(inputs, outputs)
>
>  model %>%
>    compile(loss = loss_poisson,
>            optimizer = optimizer_nadam(),
>            weighted_metrics = "poisson")
>
>  return(model)
>}

>neural_net <- new_neural_net()

>neural_net %>%
>  summary()

# Fit
>history <- neural_net %>%
>  fit(x = prep_nnet(train, x),
>      y = train[, y],
>      sample_weight = train[, w],
>      batch_size = 1e4,
>      epochs = 300,
>      verbose = 2)

# Calibrate by using last hidden layer activations as GLM input encoder
>encoder <- keras_model(inputs = neural_net$input,
>                       outputs = get_layer(neural_net, "dense_2")$output)

# Creates input for calibration GLM (extends prep_nn)
>prep_nn_calib <- function(dat, x, cat_cols = c("PolicyRegion", "VehBrand"),
>                          enc = encoder) {
>  prep_nn(dat, x, cat_cols) %>%
>    predict(enc, ., batch_size = 1e4) %>%
>    data.frame()
>}

# Calibration GLM
>fit_nn <- glm(Freq ~ .,
>              data = cbind(train["Freq"], prep_nn_calib(train, x)),
>               family = quasipoisson(),
>               weights = train[[w]])
```

Remark 3.1. The purpose of the helper functions `prep_xgb` and `prep_nn_calib` is to map the prepared `data.frame` to the data interface of XGBoost and the calibrated neural network.

4 Global model agnostic methods

In this section, we will present model agnostic tools that describe global properties of the model. First, we will introduce the **flashlight** package used throughout the tutorial.

4.1 Relevant R packages

Different R packages are available to explain black box models by model agnostic tools. The most relevant—in the authors' view—at the time of writing are **DALEX** [3], **iml** [29], and **flashlight** [25]. In this tutorial, we will use the latter (maintained by the authors) because of its broad palette of explainability tools and the fact that each of its implemented methods is able to deal with case weights, an essential aspect of many actuarial models[4]. So far, **iml** does not support case weights, while **DALEX** offers case weights for some of its methods. Further advantages of **flashlight** include stratified analyses and the simultaneous comparison of multiple models. Common to all three mentioned packages is the basic workflow:

1. Fit the model.
2. Bundle relevant information (prediction function, model, data) to an explainer object.
3. Use this object for explaining the model.

Having said this, the choice of the package is less important than the decision to *use* model interpretability tools.

Remark 4.1. Note that throughout the process, the models are never refitted.

Remark 4.2. Often, explainability is improved by moving model-specific feature transformations (like dummy or integer codings, log-transformations, and spline basis expansions) to the prediction function. We explicitly did so for our XGBoost model and the neural network and implicitly also for our GLM (dummy codings and spline expansion in the model formula). Take, for instance, the categorical predictor "PolicyRegion". In the XGBoost model and the neural network, it is represented by integers, while the GLM implicitly uses dummy variables. It would be difficult to compare its effects across models if the explainers used different data interfaces.

[4] In the case of Poisson regression, case weights can be avoided by the use of offsets.

Defining the explainers The **flashlight** package expects the following information in its explainer/flashlight object:

- `model`: The fitted model object, e.g. the one returned by `glm`.
- `data`: A data set used for evaluation.
- `predict_function`: A function taking `model` and `data` and returning numeric predictions. The default function is `predict`, though this often has to be set manually.
- `label`: The name of the model (will be shown in plots).
- `y`: The name of the response variable in `data`. Required for some (but not all) explainability tools.
- `metrics`: A named list of scoring functions. The default is mean squared error `mse`, which needs to be adapted depending on the model objective.[5]
- `linkinv`: Optional function used to transform the values returned by the `predict_function`.
- `w`: Optional name of the variable in `data` representing the case weights.
- `by`: Optional character vector of names of grouping variables in `data` used to stratify the results.

After having specified the model specific flashlights, we can combine them to a *multi-flashlight* to allow for easy comparison across models. In order to minimize redundancies, common arguments like `y`, `by`, `data` and/or `w` can be passed to all flashlights during this step. If necessary, the resulting completed flashlights contained in the multiflashlight can be extracted again by `$` as in a list.

Listing 5 shows how to construct the explainers/flashlights for the three fitted models. Since in some situations, interpretability is easier on the link-scale—in our case, it is the canonical scale of the Poisson model, i.e. the log-scale—we have added a second (multi-)flashlight with the logarithm as `linkinv` in order to analyze the explainers on the canonical scale.

Our test data set serves as evaluation data set.

Listing 5: Code to create the (multi-)explainers

```
>set.seed(1)

>library(MetricsWeighted)  # 0.5.0
>library(flashlight)       # 0.7.2
>library(ggplot2)          # 3.3.0
```

[5] All scoring functions need to be available in the R workspace and require arguments `actual`, `predicted`, `w` (case weights) as well as a placeholder `...` for further arguments. Metrics available in the R package **MetricsWeighted** [24] follow this structure.

```
# Setting up explainers
>fl_glm <- flashlight(
>  model = fit_glm, label = "GLM",
>  predict_function = function(fit, X) predict(fit, X, type = "response")
>)

>fl_nn <- flashlight(
>  model = fit_nn, label = "NNet",
>  predict_function = function(fit, X)
>    predict(fit, prep_nn_calib(X, x), type = "response")
>)

>fl_xgb <- flashlight(
>  model = fit_xgb, label = "XGBoost",
>  predict_function = function(fit, X) predict(fit, prep_xgb(X, x))
>)

# Combine them and add common elements like reference data
>metrics <- list(‘Average deviance‘ = deviance_poisson,
>                ‘Relative deviance reduction‘ = r_squared_poisson)
>fls <- multiflashlight(list(fl_glm, fl_nn, fl_xgb), data = test,
>                       y = y, w = w, metrics = metrics)

# Version on canonical scale
>fls_log <- multiflashlight(fls, linkinv = log)
```

Using the explainers As soon as the explainers are built, we can apply the explainability functions, whose results can be visualized by their **ggplot2**-based `plot` method:

- `light_performance`: Model performance.
- `light_importance`: Variable importance.
- `light_ice`: Individual conditional expectation (ICE) profiles.
- `light_profile`: Partial dependence, accumulated local effects or other profiles.
- `light_effects`: Combines different profiles.
- `light_interaction`: Overall and pairwise interaction strength.
- `light_global_surrogate`: Global surrogate tree.
- `light_breakdown`: Local variable contribution breakdown for a single observation.
- `light_scatter`: Calculates values (e.g. predictions) to be plotted against a variable of interest.

In the rest of the section, we will explain all the global model agnostic methods in detail using the **flashlight** package.

4.2 Model performance

How good are model predictions if applied to unseen data? This aspect is of key interest for any supervised ML model and helps to identify good models or, if applied to subgroups, identify segments with low performance.

For comparing our three frequency models on the test set, we will use as absolute performance metric the exposure weighted average

$$D(y, \hat{y}) = \sum_{i=1}^{n} w_i S(y_i, \hat{y}_i) / \sum_{i=1}^{n} w_i \tag{1}$$

of the unit Poisson deviance

$$S(y, \hat{y}) = 2\left(y \log \frac{y}{\hat{y}} - (y - \hat{y})\right),$$

where $y_i \geq 0$ is the observed response `Freq`, $\hat{y}_i > 0$ is the predicted expectation of the response, and $w_i \geq 0$ is the weight `Exposure` for observations $i = 1 \ldots n$ of the test data. Note that the Poisson deviance is a strictly consistent scoring function for the expectation $\mathbb{E}[Y]$, which is our model target[6], see [14]. In addition, we will compare the relative reduction

$$\text{Pseudo } R^2 := 1 - \frac{D(y, \hat{y})}{D(y, \bar{y})}$$

of D with respect to a null model consisting only of the exposure weighted average $\bar{y}$ (for simplicity calculated on the data on which Pseudo R^2 is evaluated). This figure measures the proportion of deviance explained by the model.

Listing 6 shows the R code used to calculate the two performance measures and its results[7]. The performance scores are visualized in Figure 3. The boosted trees explain about 4.5 % of the deviance, while the neural network does slightly worse. With a relative deviance reduction of 2.6 %, the additive GLM does substantially worse than the two black box models. It consists of main effects only, hence it is clear that its performance is worse compared to the two black box models that include interactions by construction. As a comparison: the GLM estimates 45 coefficients, while the deep neural network uses 679 parameters and is accordingly more flexible.

Note that relative deviance reductions are expected to be rather low since car liability claims occur highly at random.

In Section 6 we will check the performance of a more competitive GLM with interactions.

Listing 6: output of command `light_performance(fls)`

```
>(perf <- light_performance(fls))

  metric                         value  label
1 Average deviance              0.589   GLM
2 Relative deviance reduction 0.0261   GLM
```

[6] Strictly speaking, it is the expectation conditionally on X.

[7] Plots of flashlight methods can be modified by **ggplot2**-syntax. In what follows, we suppress such modifications for the sake of brevity.

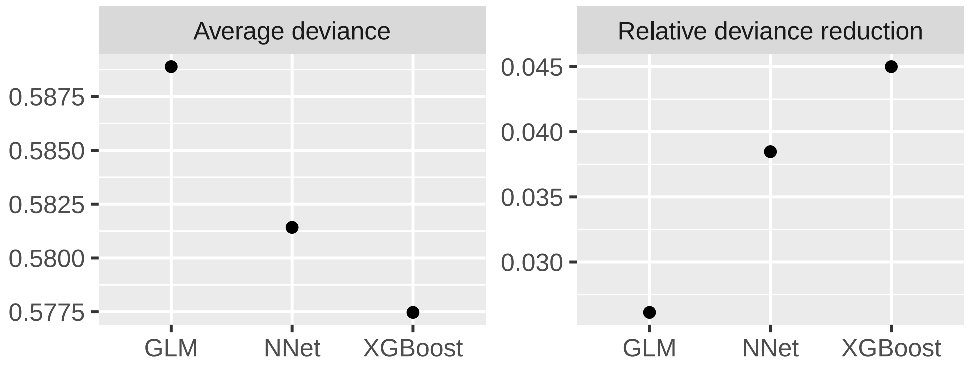


Figure 3: Average unit Poisson deviance (left) and its relative improvement (right). Both scoring functions are evaluated on the 20 % hold-out data.

```
3 Average deviance             0.581  NNet
4 Relative deviance reduction 0.0385 NNet
5 Average deviance             0.577  XGBoost
6 Relative deviance reduction 0.0450 XGBoost

# Plot
>plot(perf, geom = "point") +
>  labs(x = element_blank(), y = element_blank())
```

Model insights

- Overall, the deviance gains seem to be on a realistic level.
- The two black box models perform comparably.
- The black box models perform clearly better than the GLM, possibly due to unmodeled interaction effects in the GLM.

4.3 Variable importance

Which variables are particularly relevant for the model? This aspect is helpful in different ways. Firstly, it provides additional information and therefore insight into the models and the regression modeling task, i.e. the problem itself. Secondly, it might help to simplify the models by removing potentially irrelevant (but maybe difficult to assess) covariables. Thirdly, it might help to identify problems in the data structure: If one covariable is extremely relevant and all others are not, then there might be some sort of information leakage from the response into that covariable.

Different modeling techniques offer different ways to assess variable importance. For (generalized) linear models, we can study likelihood ratio test statistics (available, for example, through the R function `drop1`) or scaled absolute values of the coefficients. For tree-based methods, the number of splits or split gains can be considered.

A *model agnostic* way to assess variable importance is called *permutation importance*: For a covariable X, its values in the data set are randomly shuffled and the drop in performance with respect to a scoring function is calculated, see Algorithm 1 for the corresponding pseudocode. The larger the drop, the higher we interpret the importance of that variable.

Algorithm 1: Permutation importance

```
scoreOriginal ← performance on data
for x in variables do
    dataShuffled ← data with permuted column x
    scoreShuffled ← performance on dataShuffled
    importance[x] ← scoreShuffled − scoreOriginal
end
output : importance
```

If a variable can be shuffled without any impact on model precision, it is assumed to be irrelevant. This method was originally proposed by Leo Breiman in his random forest paper [4] and investigated in [11]. For large data sets, one single permutation is often sufficient to get stable results. For smaller data sets, we recommend running multiple permutations and average the results[8].

Which variables are considered important in our models? Listing 7 shows the code to calculate and plot permutation importance values for our covariable vector. The results

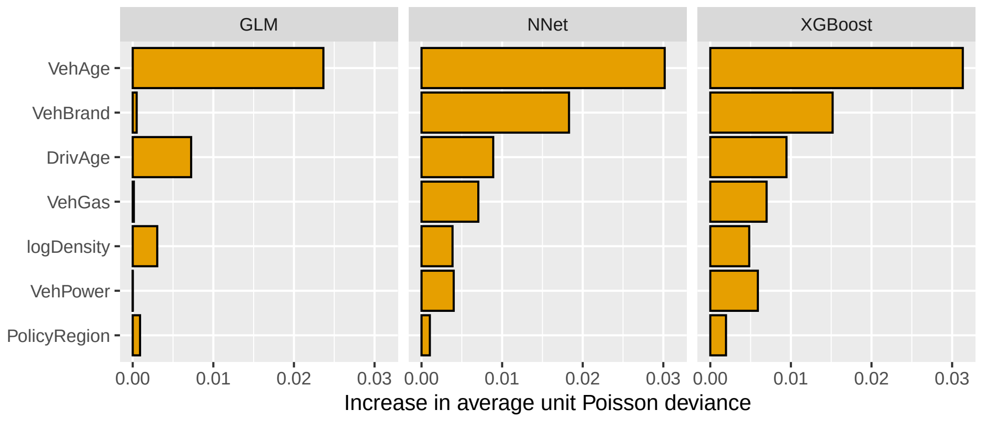


Figure 4: Permutation importance with respect to the increase in average unit Poisson deviance due to shuffling. The variables are ordered by their average importance across all three models.

are depicted in Figure 4. In all three models considered, the most important variable is

[8] Available through option `m_repetitions` of function `light_importance` in the **flashlight** package.

age of car. The order of importance is almost identical for the two black box models and quite different for the GLM, partly because we did not include interaction terms in its model formula. This is most obvious for the vehicle brand and will be addressed again in Section 6.

Listing 7: calculate importance plot

```
>imp <- light_importance(fls, v = x)
>plot(imp, fill = "#E69F00", color = "black")
```

Remark 4.3 (Permutation importance and interactions)**.** For models with interactions, shuffling a variable does not only destroy its main effect but also all associated interactions.

Model insights

- The most important variable for all models is vehicle's age.
- The order of importance is similar across our black box models and seems plausible.
- The GLM lacks interaction effects, e.g. with the two categorical variables `VehBrand` and `VehGas`.

4.4 Effects

In a generalized linear regression without non-linear terms and interactions, the fitted model consists of an affine linear function in the link space. Its coefficients immediately tell us how the response is expected to react on changes in the covariables. How to describe the effect of a variable X for a (generalized) linear model with complex non-linear terms and high-order interactions or black box models like boosted trees or neural networks? In the following, we will introduce different established methods to do so.

Individual conditional expectations One approach is to study *Individual Conditional Expectation* (ICE) profiles of a couple of observations, see [15]: Such a profile shows how predictions of an observation i react when the input variable X slides over its range, see Algorithm 2.

Algorithm 2: ICE for variable x and one observation

```
obs ← data row
for $v$ in grid of values do
    obs[$x$] ← $v$
    ice[$v$] ← prediction for obs
end
output: ice
```

Notes:

- By construction, ICE profiles of a variable X are parallel as long as the corresponding X is modeled additively.
- The stronger the interaction effects associated with X, the larger the differences in shape across ICE profiles of different observations. This, however, does not reveal with *which* other variable(s) the interaction occurs, see Section 4.5 on how to check this.
- ICE profiles are sometimes vertically shifted to meet in one point[9]. In such *centered* ICE profiles, interactions are even easier to spot.

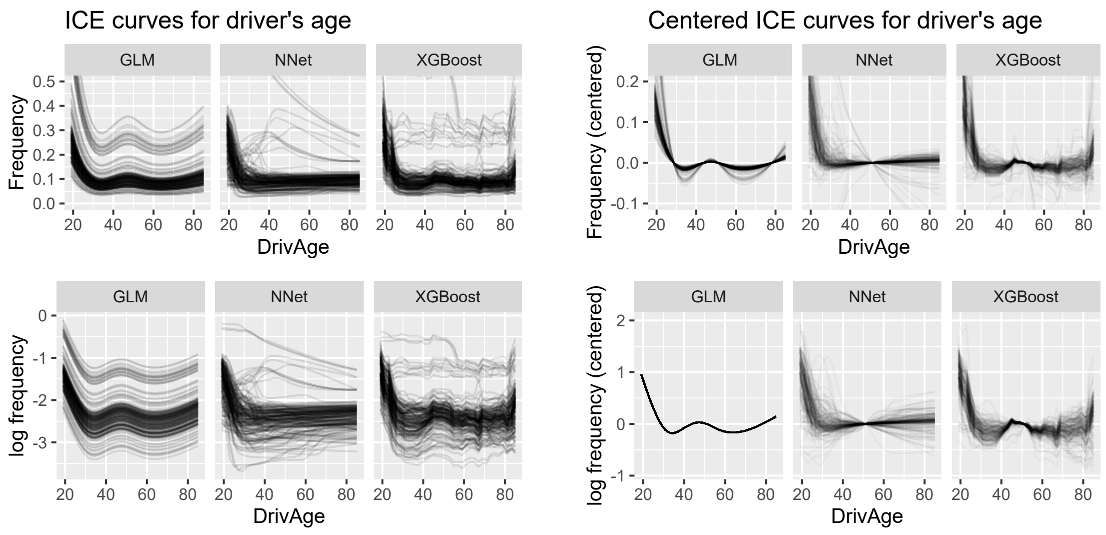


Figure 5: Uncentered (lhs) and centered (rhs) ICE profiles for 200 randomly picked observations and the covariable *driver's age*. The upper figures are on the frequency scale, the lower on the log frequency scale.

Figure 5 shows ICE profiles (uncentered and centered) for 200 randomly picked observations for the covariable *driver's age*. Since the GLM does not incorporate

[9] e.g. an endpoint or a central value on the x-axis

interactions, its ICE profiles are perfectly parallel on the log frequency scale and almost parallel on the frequency scale. Both black box models show some very wild profiles. Still, the bulk of them follows a quite homogeneous pattern similar to the GLM. From this, we may conclude that interaction effects associated with driver's age are present but not too strong. We will confirm this later in Section 4.5 where we dive deeper into interaction effects. Listing 8 contains the code used to generate all figures in this Subsection 4.4.

Listing 8: Code used to produce figures in Subsection 4.4

```
# ICE (uncentered and centered)
>plot(light_ice(fls, v = "DrivAge", n_max = 200, seed = 3), alpha = 0.1)
>plot(light_ice(fls, v = "DrivAge", n_max = 200, seed = 3,
>               center = "middle"), alpha = 0.03)

# Partial dependence curves
>plot(light_profile(fls, v = "VehAge", pd_evaluate_at = 0:20))
>plot(light_profile(fls, v = "DrivAge", n_bins = 25))
>plot(light_profile(fls, v = "logDensity"))
>plot(light_profile(fls, v = "VehGas"))

# ALE versus partial dependence
>ale_DrivAge <- light_effects(fls, v = "DrivAge", counts_weighted = TRUE,
>                             v_labels = FALSE, n_bins = 20, cut_type = "quantile")
>plot(ale_DrivAge, use = c("pd", "ale"), show_points = FALSE)

# Classic diagnostic plots
>plot(light_profile(fls, v = "VehAge", type = "predicted"))
>plot(light_profile(fls, v = "VehAge", type = "residual")) +
  geom_hline(yintercept = 0)
>plot(light_profile(fls, v = "VehAge", type = "response"))

# Multiple aspects combined
>eff_DrivAge <- light_effects(fls, v = "DrivAge", counts_weighted = TRUE)
>p <- plot(eff_DrivAge, show_points = FALSE)
>plot_counts(p, eff_DrivAge, alpha = 0.3)
```

Remark 4.4 (Monotonicity constraints and boosting)**.** Major boosting implementations such as XGBoost [6], LightGBM [18], or CatBoost [8] offer the option to impose monotonicity constraints by rejecting splits that violate the condition. They implicitly guarantee monotone increasing and decreasing ICE profiles, respectively. Adding such constraints might considerably increase explainability and usefulness of a model, e.g. by ensuring that a higher deductible would never lead to a higher predicted frequency. In actuarial applications, one often requires to have such monotonicity constraints. Figure 6 shows the result of fitting XGBoost with parameter `monotone_constraints = c(0,-1,0,0,0,0,0)`, telling XGBoost that the second variable (`DrivAge`) should have a monotonically decreasing effect on the response (code not shown).

Partial dependence profiles If many ICE profiles are averaged, we get a *partial dependence profile*, which can be viewed as the main effect of variable X pooled over all interactions, i.e. the average effect of variable X. Partial dependence plots where introduced in Friedman's seminal 2001 article on gradient boosting [12]. The algorithm is described in Algorithm 3.

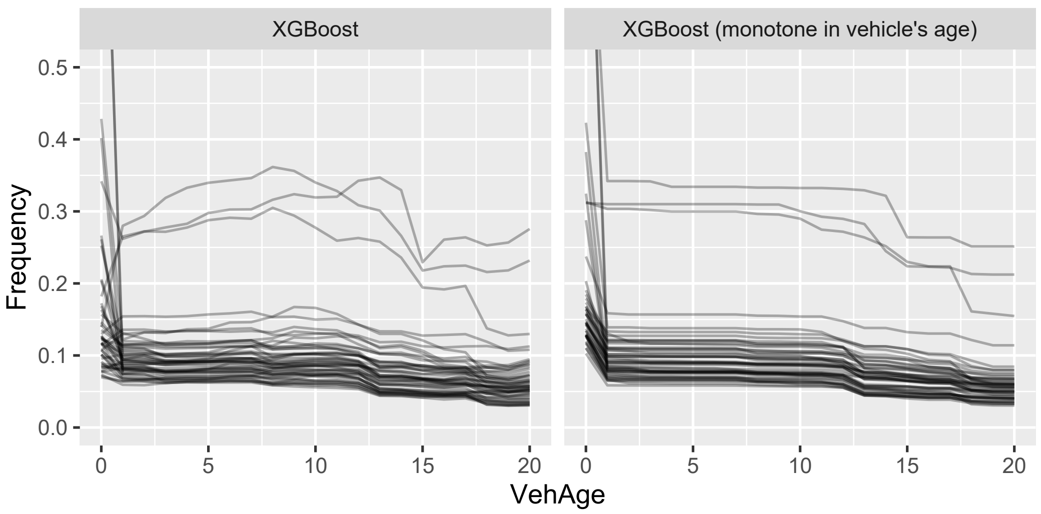


Figure 6: ICE profiles for vehicle's age. LHS without, RHS with monotonicity constraints for this variable.

Algorithm 3: Partial dependence profile for variable x

```
$n \leftarrow$ number of observations
$m \leftarrow$ grid size for variable $x$
iceProfiles $\leftarrow$ matrix with $n$ rows and $m$ columns
**for** $i$ *in* 1 *to* $n$ **do**
    iceProfiles$[i, :] \leftarrow$ ice curve for $i$th obs and variable $x$
**end**
pd $\leftarrow$ column means of iceProfiles
**output :** pd
```

For additive linear models, i.e. models without interactions, the partial dependence plot simply visualizes the estimated model coefficient.

Note that partial dependence profiles as well as ICE profiles are not limited to one single variable X. Also, multiple variables can be systematically varied on a grid in order to study the multivariate impact on the typical response (not shown in this paper).

Studying ICE and partial dependence profiles means investigating the effect of X while holding all other predictors fixed. The more unnatural this Ceteris Paribus assumption is, e.g. due to causal relationships across predictors, the less interesting are the results of ICE and partial dependence, *exactly as when interpreting coefficients of GLMs.* In our example, regional factors could be problematic: Depending on how policy regions are constructed, e.g. integers ordered by population density, it might not make sense to change the population density while keeping policy region fixed.

Figure 7 shows (univariate) partial dependence plots for selected variables. We use

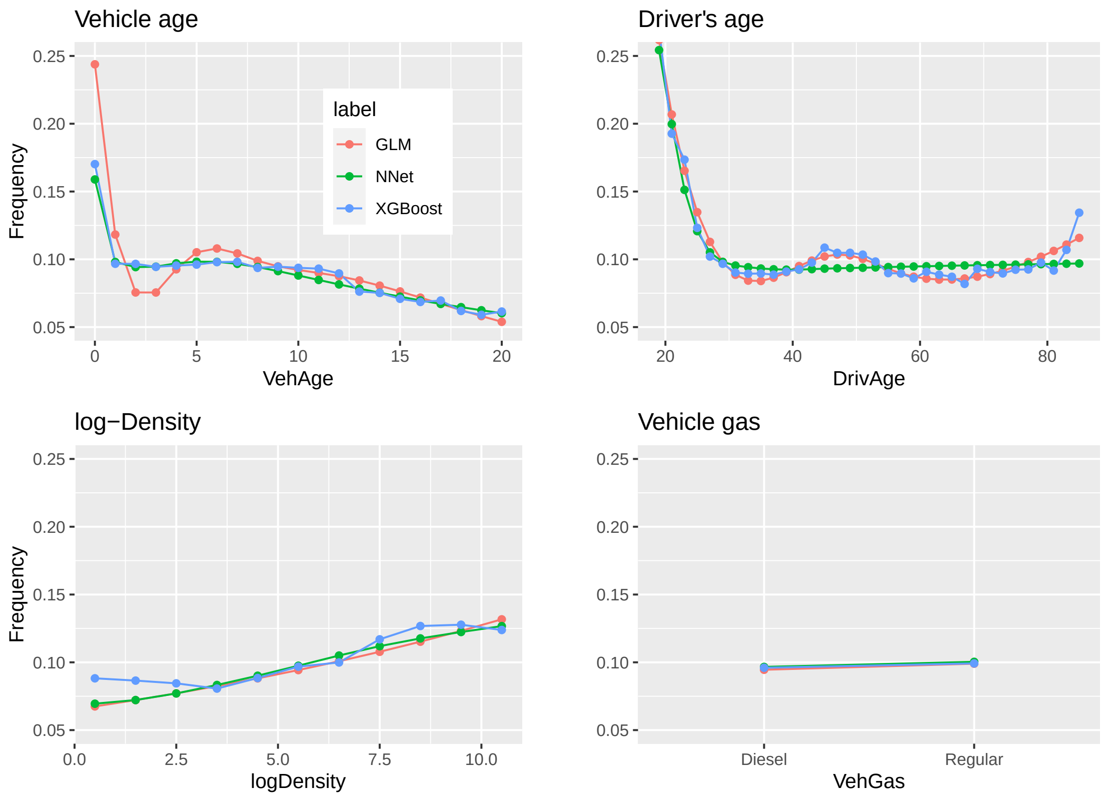


Figure 7: Partial dependence plots of selected features with fixed vertical axis to highlight the size of the effect.

fixed vertical axes across the plots to show that the (main) effect of `VehGas` is negligible. The effects are comparable across all three models. However, while the effects of the GLM and the neural network are smooth, the effects from XGBoost are sometimes wild, especially for driver's age. This behavior is expected for tree-based models as tree predictions are not continuous[10]. It depends on the specific application if such behavior is acceptable or not.

For vehicle's age, both black box models show a monotone decreasing effect, while the GLM shows pronounced non-monotonicity, possibly to compensate for the lack of interaction effects. Curiously, for each of the three important predictors, one model stands out: For vehicle's age it is the GLM, for driver's age its the neural network and finally for log-density, it is XGBoost.

Accumulated local effects profiles *Accumulated local effects* or *ALE* profiles [2] try to overcome the above mentioned weakness of ICE and partial dependence plots by relaxing the Ceteris Paribus assumption. They consider *local* effects only and thus lead to more natural and less biased results in case of strongly correlated covariables. The ALE profile at position x_i of a covariable x is calculated as follows:

1. Calculate slope Δ_i of partial dependence profile for covariable x based on all observations with $x \in [x_{i-1}, x_i]$. Δ_i can be considered a numerical derivative.
2. The uncalibrated ALE value at x_i is the cumulative sum $\sum_{j \leq i} \Delta_j$.
3. In the (optional) calibration step, the effects are shifted vertically to the average response or prediction.

In contrast to a partial dependence plot, ALE profiles estimate the effect at x only based on observations *close to* x.

If associations to other features are not too strong, ALE plots look similar to partial dependence plots, see Figure 8. The stronger the associations to other features, the more they may differ from partial dependence plots. In these cases, they are considered more trustworthy. Still, partial dependence plots are more commonly used because of their simplicity.

Remark 4.5 (ALE and categorical predictors)**.** While versions of ALE for nominal (unordered) categorical variables exist—a possibility is outlined in [2]—we recommend them primarily for numerical covariables.

Remark 4.6 (Training or test data?)**.** Should the models be explained on the training data set or on the test (resp. validation) data? For explainability methods that do *not* utilize the response (like partial dependence or ALE profiles), it does not matter. Methods that use the response (performance considerations, permutation importance, residual plots) should usually be evaluated on data set independent of the training set.

[10] Ways to smoothen the predictions from XGBoost include: more trees, smaller learning rate, stronger regularization and, if appropriate, monotonicity constraints.

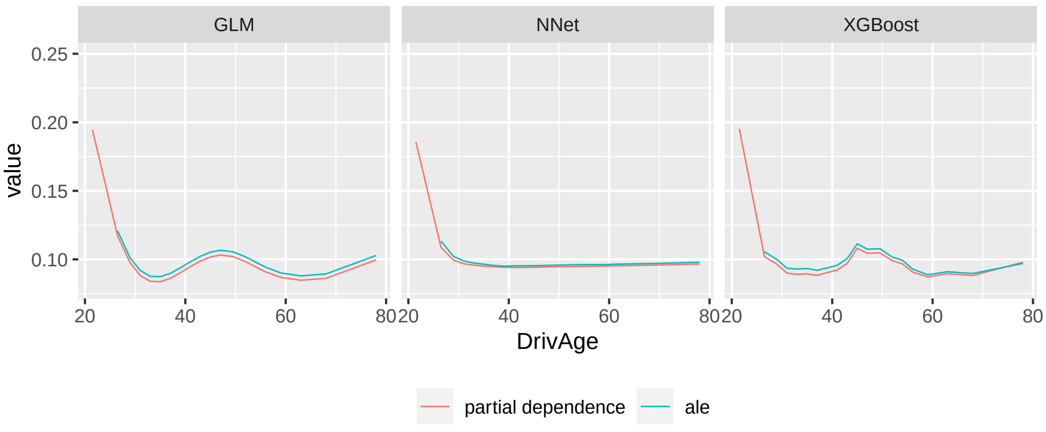


Figure 8: Comparison of ALE and partial dependence plots for driver's age.

Further profile plots Further ways to assess effects include the following visualizations borrowed from traditional diagnostic plots of linear regression.

- Response-versus-covariables: Averages (or boxplots) of responses are plotted against a covariable X.
- Predicted-versus-covariables: Averages of predicted values are plotted against a covariable. This visualizes the effect of X including the effects of correlated predictors and is sometimes called *marginal plot* or *M-plot*, see [2]. Differences to the response-versus-covariables plot lead to the next point.
- Residual-versus-covariables: Averages (or boxplots) of residuals are plotted against a covariable. Ideally, the average residual is sufficiently close to zero across the whole range of X. Systematic differences from zero might reveal model underfit, at least if evaluated on a not too small data set. Similarly, if the average residuals are close to zero on the training data set but not on an independent hold-out data set, this is a typical sign of overfit. This plot is a binned version of the Tukey-Ascombe plot [1].

These plots are all based on plotting weighted means (of the responses, predictions or the residuals) against the covariable of interest. The package **flashlight** bins continuous X for this purpose[11].

The three above mentioned visualizations for the covariable `VehAge` are shown in Figure 9. Figure 10 combines such response and prediction profiles along with partial dependence plots for the driver's age. There, the average prediction curves are very similar to the partial dependence plots, indicating that the effect of driver's age is not influenced by other predictors. Note that all models seem to be biased for drivers aged

[11] An alternative to binning is a scatter plot smoother as used by the diagnostic residual plots of the R function `lm`, for example.

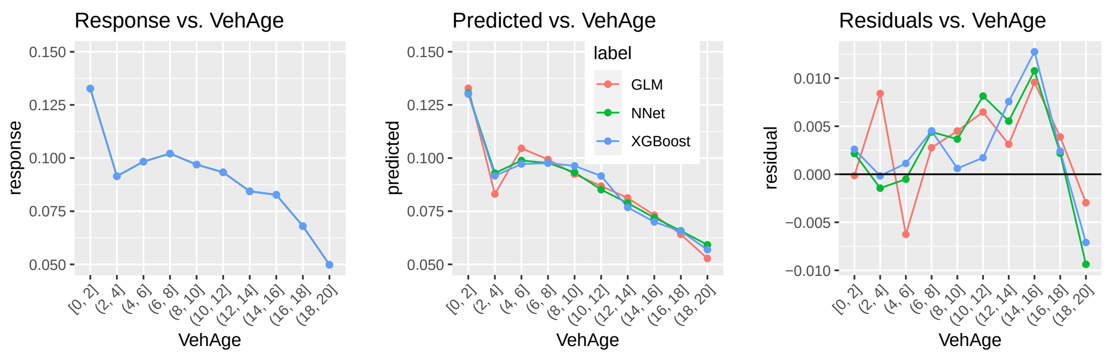


Figure 9: Further types of profile plots adopted from traditional diagnostic plots. The left image shows exposure weighted frequencies per vehicle age group. The middle image weighted averages of predictions. Their difference (i.e. weighted averages of residuals) is visualized in the right picture.

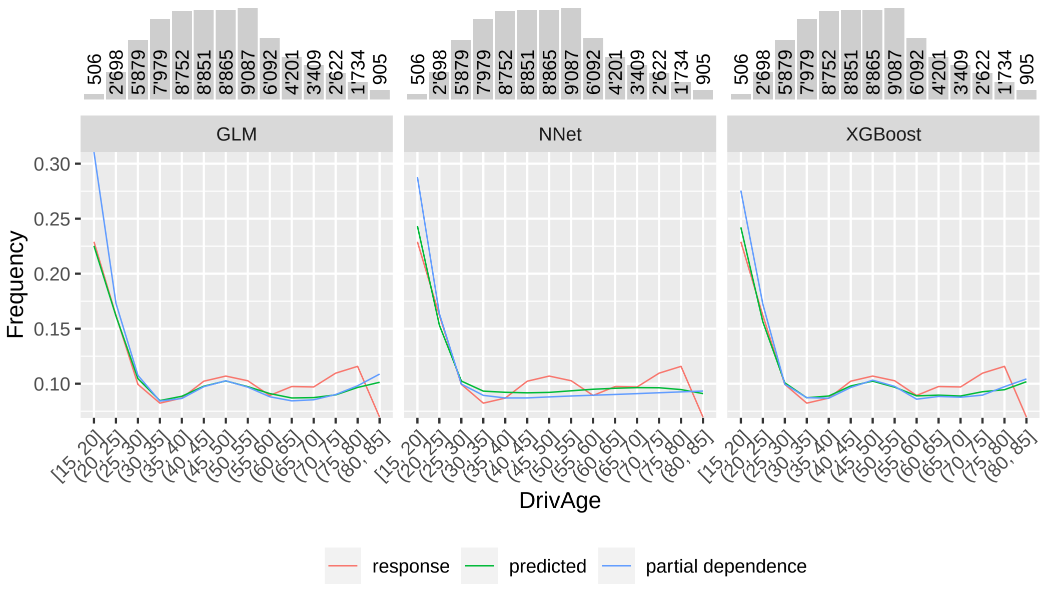


Figure 10: Combination of partial dependence plots, M-plots and residual-versus-covariables plots for the variable `DrivAge`. The histograms visualize exposure.

above 75 years. This is mostly explained by the low exposure in that age range and the fact that these curves are calculated on the independent test data.

Model insights

- The (main) effects of the strongest predictors look reasonable, e.g. higher population density tends to higher claims frequency.
- The black box models reveal strong non-linear effects.
- In the GLM, `VehAge` and `DrivAge` are already well-represented by splines. There might be room for improvement for `logDensity`.
- We do not see problematic overfitting.

4.5 Interaction strength

After studying main effects e.g. by looking at partial dependence plots for the most important variables, the next step could be to evaluate the strength of overall or pairwise interaction effects, at least for the non-additive models. In Section 4.4 we have seen that non-parallel ICE curves give a hint on the presence of interactions. How to quantify the strength of interaction effects associated with a covariable X? Between which variable pairs do the strongest interactions occur?

A model agnostic way to assess such measures is based on partial dependence profiles as introduced by Friedman and Popescu in [13]. Their Friedman's H-statistic to measure pairwise interaction strength between covariables X_j and X_k is defined as follows:

$$H_{jk}^2 = \sum_{i=1}^{n} \left[\mathrm{PD}_{jk}(x_j^{(i)}, x_k^{(i)}) - \mathrm{PD}_j(x_j^{(i)}) - \mathrm{PD}_k(x_k^{(i)})\right]^2 / \sum_{i=1}^{n} \mathrm{PD}_{jk}^2(x_j^{(i)}, x_k^{(i)}) \,,$$

where the sums run over a subset of n randomly[12] selected observations i. By PD_j, PD_k, and PD_{jk} we denote the zero-mean centered one- and two-dimensional partial dependence profiles for X_j and X_k. The superscript (i) refers to the value of the partial dependence profile of observation i. In words, H^2 measures the proportion of variability in the joint effect of X_j and X_k unexplained by their main effects. A value close to zero indicates almost no pairwise interaction, while a value close to one means that most effects come from the pairwise interaction. Versions of Friedman's H measure the total interaction effect strength associated with one covariable X_j (i.e. pairwise and higher order interaction effects) or interaction strengths between more than two variables. Here, we will focus on pairwise effects. While H^2 measures interaction strength *relative* to the total joint effect of X_j and X_k, additional information is gained by considering the

[12] This is different from partial dependence profiles, where a fixed grid is used.

square-root of the numerator of H^2 as

$$\tilde{H}_{jk} = \sqrt{\sum_{i=1}^{n} \left[\mathrm{PD}_{jk}(x_j^{(i)}, x_k^{(i)}) - \mathrm{PD}_j(x_j^{(i)}) - \mathrm{PD}_k(x_k^{(i)})\right]^2},$$

which serves as an absolute measure of interaction strength that can be used to figure out the strongest absolute interaction effects.

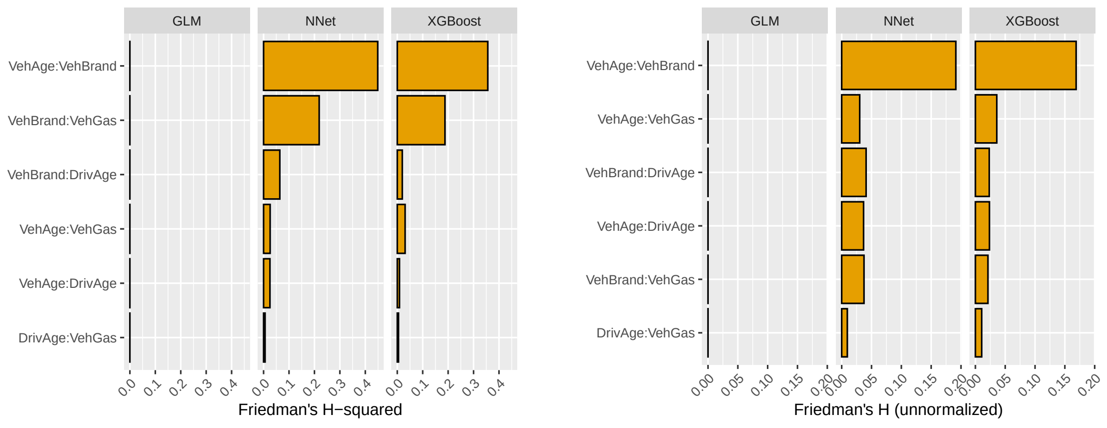


Figure 11: Pairwise interaction strength between the four strongest predictors. The left plot shows Friedman's H^2. The plot on the right-hand side shows unnormalized Friedman's H. These plots show the corresponding figures on the canonical scale.

The left plot of Figure 11 shows that most pairwise interactions modeled by the two black box models are rather weak compared to the corresponding main effects. Only the vehicle's age/brand interaction and the brand/gas interaction explain more than 10 % of the variability in their two-dimensional partial dependence profiles. With an H^2 of about 40 %, the interaction between vehicle's age and brand is very strong, namely almost as strong as the two main effects together. The strongest absolute interaction effect for the boosted trees as well as for the neural network, as shown on the right-hand side of Figure 11, occurs also between vehicle's age and the brand. All statistics have been calculated from partial dependence profiles on the canonical scale in order to stress the fact that interaction effects of the additive GLM model are all zero, which is confirmed by Figure 11. As we will see in Section 6, adding the strongest interaction effects to the GLM will lead to substantial improvement of the GLM. The code to calculate these interaction plots is highlighted in Listing 9.

Listing 9: Code to create interaction strength plots

```
># Interaction (relative)
>interact_rel <- light_interaction(
>  fls_log,
>  v = most_important(imp, 4),
>  take_sqrt = FALSE,
>  pairwise = TRUE,
>  use_linkinv = TRUE,
>  seed = 61
>)
>plot(interact_rel, color = "black", fill = "#E69F00")

># Interaction (absolute)
>interact_abs <- light_interaction(
>  fls_log,
>  v = most_important(imp, 4),
>  normalize = FALSE,
>  pairwise = TRUE,
>  use_linkinv = TRUE,
>  seed = 61
>)
>plot(interact_abs, color = "black", fill = "#E69F00")
```

There are basically two ways to visualize interaction effects. The first one is to plot two-dimensional partial dependence or ALE curves, the second one is to show partial dependence or ALE curves of one variable stratified by the other. Figure 12 shows the

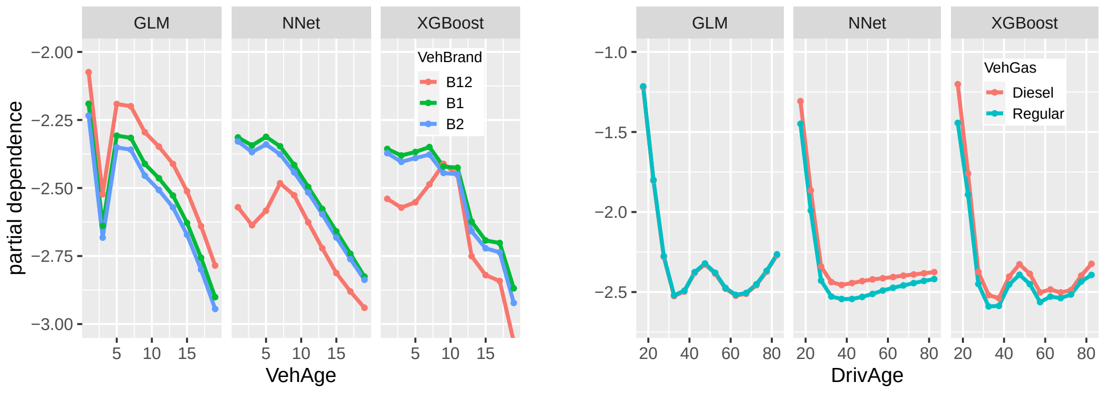


Figure 12: Stratified partial dependence plots to visualize interaction effects, on the left between vehicle's age and brand, on the right for driver's age and vehicle gas.

second approach for interactions between vehicle brand and vehicle's age (the strongest interaction) and between driver's age and vehicle gas (weakest interaction). In order to improve visual impression for the interaction between brand and vehicle's age, we focus on the three most frequent brands (B1, B2 and B12) which cover about 72 % of all test rows. In line with the impression from Friedman's H, the interaction between vehicle brand and vehicle age is clearly visible for both black box models (left-hand side), i.e. lines are not parallel. The lines for the weak interaction between driver's age and vehicle gas are almost parallel (right-hand size). Thanks to working on the log-scale, the partial dependence profiles are exactly parallel for the GLM without interactions. The code to generate the stratified partial dependence plots is shown in Listing 10.

Listing 10: Code to create stratified partial dependence plots

```
# Filter on largest three brands
>sub_data <- test %>%
>  filter(VehBrand %in% c("B1", "B2", "B12"))
```

```
>pdp_vehAge_Brand <- light_profile(fls_log, v = "VehAge", by = "VehBrand",
>  pd_seed = 50, data = sub_data)
>plot(pdp_vehAge_Brand))

>pdp_DrivAge_Gas <- light_profile(fls_log, v = "DrivAge", by = "VehGas",
>  pd_seed = 50)
>plot(pdp_DrivAge_Gas)
```

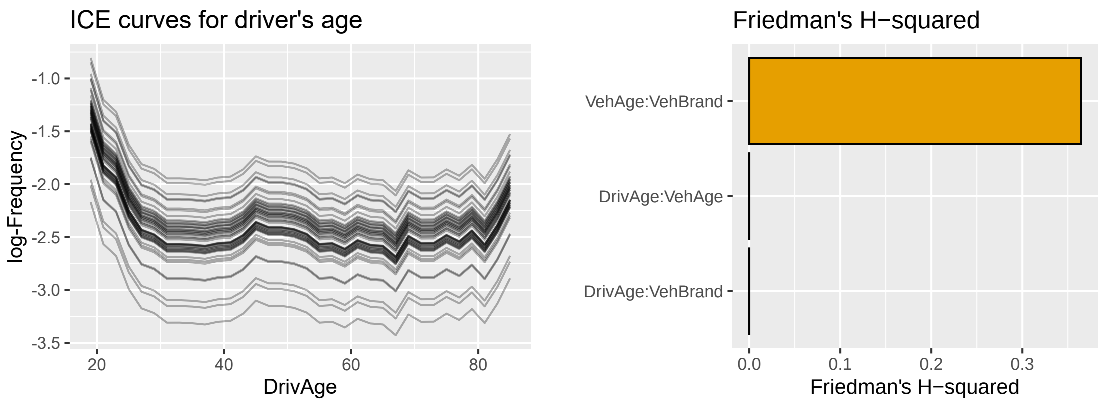


Figure 13: Modified XGBoost fit, disallowing driver's age to interact with any other variable. Consequently, the ICE curves are all parallel (on log-scale) and Friedman's measure of interaction strength is 0 for all pairs involving driver's age.

Remark 4.7 (Interaction constraints)**.** In contrast to GLMs, most black box models implicitly utilize interactions across variables in order to maximize performance. In certain circumstances, some interactions should be disallowed, e.g. for technical, ethical or regulatory reasons. One example is calibration: If an insurance pricing model is to be applied to the current or future portfolio, a time trend factor might replace the estimated (past) time effects. Here, modeling time without any interactions could be desirable. In flexible neural network frameworks such as Keras and TensorFlow, some interaction constraints can be set through careful choice of the network architecture. For example, variables without interactions would be added as input to the last hidden layer or through skip-connections. Interaction constraints are also possible for some tree boosting implementations like XGBoost. Figure 13 shows the effect of disallowing driver's age to interact with any other variable, setting the parameter `interaction_constraints = list(4, c(0, 1, 2, 3, 5, 6))` (driver's age is the fourth variable, starting with index 0). An interaction constraint does not prevent that other covariables capture part of the suppressed interaction effect, see also [20].

Model insights

- The black box models reveal interactions across all considered variable pairs. The two strongest ones occur between `VehAge` and `VehBrand` as well as between `VehBrand` and `VehGas`.
- One observes that the GLM was indeed specified without interactions.

4.6 Global surrogate models

A very different approach to look into a black box model is explained in [28]: The predictions themselves are modeled again by a simpler model that is easy to explain, usually a single decision tree. Corresponding pseudocode is shown in Algorithm 4.

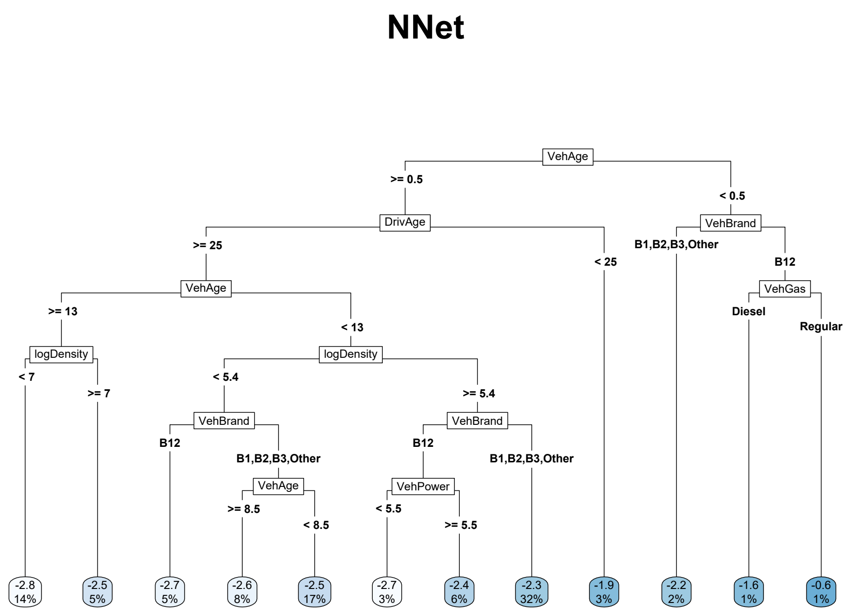
```
Algorithm 4: Global surrogate model
pred ← calculate predictions on data
fit ← fit surrogate model on pred
output : fit
```

This global surrogate model is then explained in place of the complex original model. In order to judge the approximation quality of the surrogate, [28] suggests providing its R-squared in explaining the predictions of the original model.

Figure 14 visualizes the idea by fitting (log) predictions of the neural network and the boosted trees by a single regression tree. Despite its simplicity and readability, the two surrogate trees explain more than 70 % of the variability of the original predictions. Note that both surrogate trees use the most important variable `VehAge` in the first split, cf. Figure 4. Overall, the surrogate models confirm our impression of reasonable black box models. The corresponding code to produce the surrogate model and its plot is given in Listing 11.

Listing 11: Code to create a global surrogate tree

```
> tree_and_net <- multiflashlight(list(fls_log$NNet, fls_log$XGBoost))
> surr <- light_global_surrogate(tree_and_net, v = x)
> plot(surr, mfrow = 2:1)
> surr
   label    r_squared tree
  <fct>         <dbl> <list>
1 NNet          0.742 <rpart>
2 XGBoost       0.716 <rpart>
```



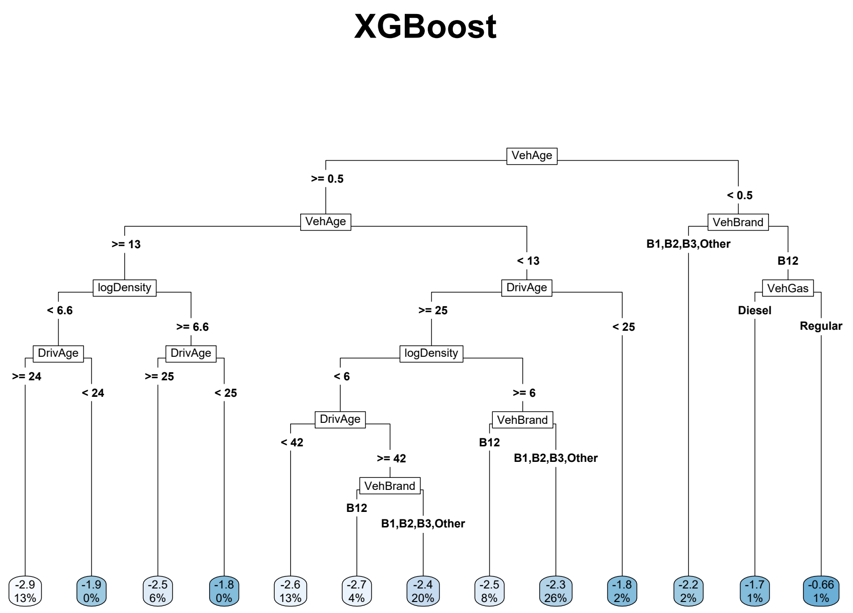


Figure 14: Decision tree explaining the (log) predictions of the neural network and the boosted trees on the test data. Leaf nodes show mean prediction on the log-scale (sorted from low to high) and the relative proportion of observations.

Model insights

- Confidence in our black box models is increased because most splits of the surrogate trees seem intuitive, e.g. younger drivers are associated with higher claims frequency.
- The surrogate trees provide us with a quick way to approximately describe how predictions in the black box models are composed.

5 Local model agnostic methods

Instead of studying global model properties such as permutation importance or partial dependence profiles, there are different techniques to explain the prediction of *one single observation*. As the observed values are not needed, this is even applicable for predictions on new single data points. We will briefly describe these methods here.

5.1 LIME and LIVE

Locally Interpretable model agnostic Explanations (LIME) [33] and Local Interpretable Visual Explanations (LIVE) [37] both start by generating an artificial data set consisting of observations, covariables and predicted responses, similar to the chosen observation. The predicted values on this artificial data set are then modeled and interpreted by a "white box" model such as a linear regression or a single decision tree.

5.2 SHAP

SHAP (SHapley Additive exPlanations) is an approach based on Shapely values [36] from game theory and was introduced by Lundberg and Lee [21]. It provides the only *fair* and *additive* decomposition of a prediction $\hat{y}_i$ of the i-th observation into contributions s_{ij} from covariables $X_j, 1 \leq j \leq m$, such that $\hat{y}_i = \sum_{j=1}^{m} s_{ij}$. A computationally efficient implementation for tree-based models is available in the Python module **shap** [22]. An R wrapper is available through the R package **shapper** [23]. These packages also allow to calculate SHAP values for other types of models like neural networks, however less efficiently.

Remark 5.1 (Additive decomposition)**.** It is the **additivity** that makes the decomposition invaluable for simple and vivid interpretations. This way, you can answer to a customer's question what the contribution of every single one of his features is to the final model outcome, see Figure 15.

5.3 Breakdown and approximate SHAP

Another approach to decompose a prediction into additive components is called *breakdown*, see [37, 16]. For a given visit order $(X_{(1)}, ..., X_{(m)})$ of specified variables, the decomposition is done as described in Algorithm 5.

Algorithm 5: Breakdown of observation "obs" (for given visit order)

```
currentPred ← mean(prediction on data)
for $x$ in visit order of variables do
    data[:, $x$] ← obs[$x$]
    newPred ← mean(prediction on data)
    effects[$x$] ← newPred − currentPred
    currentPred ← newPred
end
output : effects
```

A complication with this approach is that the visit order is relevant, at least for non-additive models and when predictors are correlated. Ideally, the algorithm could be repeated for all $m!$ possible visit orders. Their average decomposition would agree with SHAP. For practical application, this is computationally unfeasible, and one has to rely on approximations. One strategy is to run breakdown on a small number, say 20, permutations and average the resulting contributions. Another possibility is to order the variables in descending order of importance, see [16]. In this tutorial we call the first strategy "approximate SHAP" and the second one "breakdown".

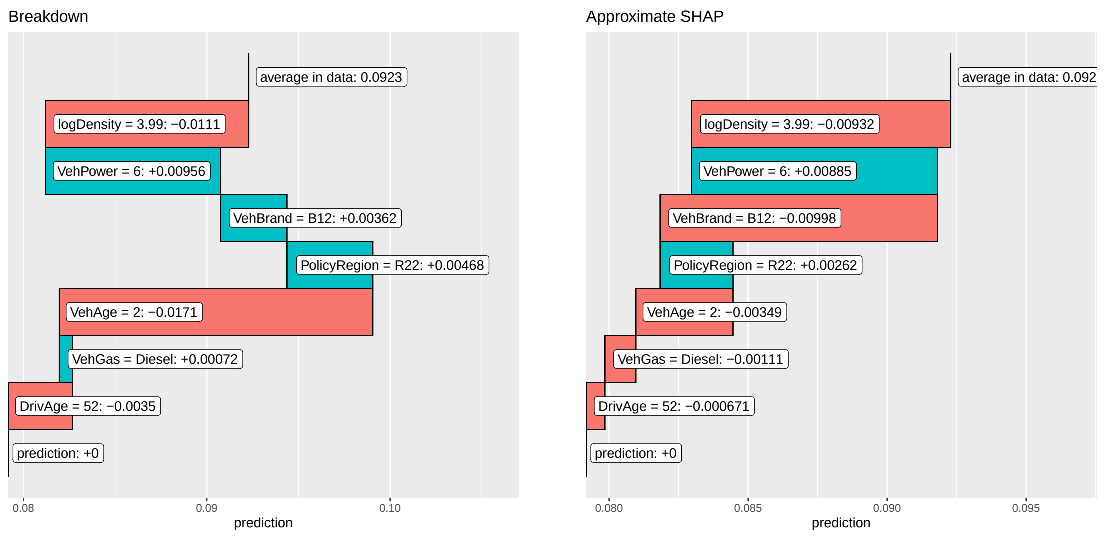


Figure 15: The left-hand side shows the breakdown decomposition (order of variables chosen by importance), the right-hand side the SHAP decomposition (average of running breakdown 20 times with random variable order) of the first observation for the XGBoost model. The plots are read from top to bottom.

Figure 15 shows the result of applying the two strategies on the first observation for the XGBoost model. The plots are read from top to bottom. For the breakdown decomposition in the left figure, this means the following:

- The average frequency of 1000 reference rows is 9.23 %.
- Due to the "good" population density at observation 1, the frequency decreases by 1.1 %.
- Then the "bad" vehicle power increases the value by about 1 %,
- . . . ,
- until eventually, the final prediction of a little less than 8 % frequency is reached.

For an additive linear model, the bar lengths equal the product of the (centered) regressor value and the model coefficient. The plots were generated by the code in Listing 12.

Listing 12: Code to create breakdown plots

```
>new_obs <- test[1, ]
>new_obs[, x]
  VehPower VehAge VehBrand VehGas DrivAge logDensity PolicyRegion
3        6      2      B12 Diesel      52   3.988984          R22

>unlist(predict(fls, data = new_obs))
     GLM.3       NNet    XGBoost
0.07692262 0.07640730 0.07999122

# Breakdown
>bd <- light_breakdown(fls$XGBoost, new_obs = new_obs,
>                       v = x, n_max = 1000, seed = 20)
>plot(bd)

# Extract same order of variables for visualization only
>v <- setdiff(bd$data$variable, c("baseline", "prediction"))

# Approximate SHAP
>shap <- light_breakdown(fls$XGBoost, new_obs,
>                         visit_strategy = "permutation",
>                         v = v, n_max = 1000, seed = 20)
>plot(shap)
```

5.4 From local to global properties

If many observations are decomposed by SHAP or breakdown, different global properties of the model can be derived.

1. Variable importance: The average absolute value $\varnothing_i|s_{ij}|$ of the contributions of variable X_j provides a measure of variable importance with excellent properties, cf. [21].
2. Effects: If contributions s_{ij} of covariable X_j are plotted against the covariable values, we get an impression of the effect of X_j on the typical response.
3. Interactions: In a scatterplot, an additional covariable X_k can be highlighted e.g. on the color scale to reveal interactions between X_j and X_k.

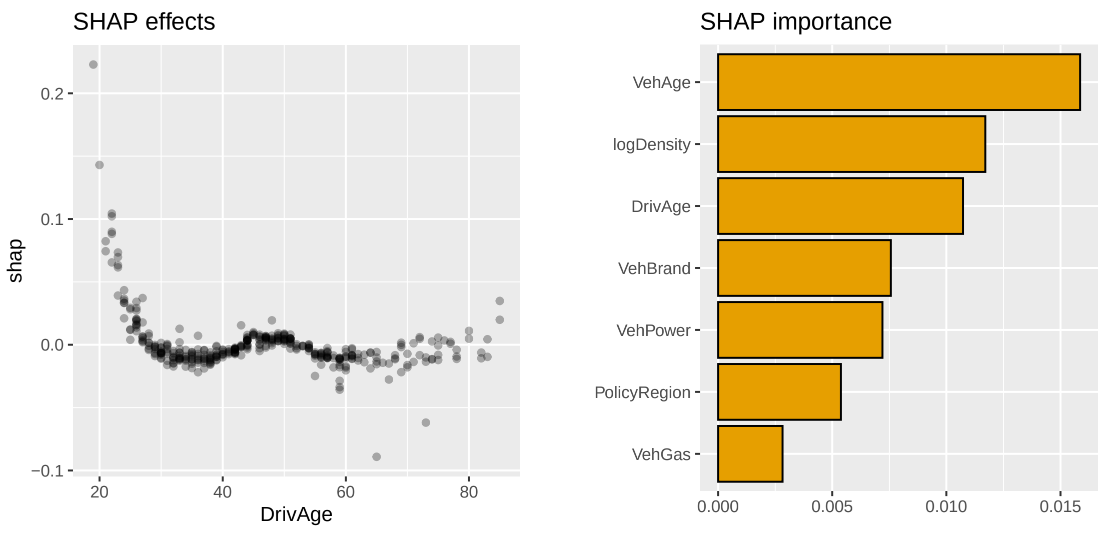


Figure 16: The left-hand side shows contributions (the signed bar heights in Figure 15 for multiple observations, plotted against driver's age). This visualizes the effect of driver's age on the frequency (the vertical scale is relative to a typical frequency). The average absolute value gives a fair measure of importance for driver's age. If we calculate such averages for all covariables, we get the SHAP variable importance plot on the right-hand side.

Figure 16 shows the idea from decomposing 500 observations by approximate SHAP for the XGBoost model. The code to generate the plots is depicted in Listing 13. The effects from the scatter plot on the left-hand side of Figure 16 give a very similar impression as the partial dependence plot in Figure 7, even if the approaches are totally different. As with permutation importance, SHAP based importance has also identified the vehicle age as the most important predictor. Still, the order of importances is quite different across the two techniques (Spearman's rank correlation is 0.46).

Listing 13: From local to global properties

```
>fl_with_shap <- add_shap(fls$XGBoost, v = x, n_shap = 500,
>                          n_perm = 12, n_max = 1000, seed = 100)
# saveRDS(fl_with_shap, file = "rdata/fl_with_shap.rds")

>imp_shap <- light_importance(fl_with_shap, v = x, type = "shap")
>plot(imp_shap, fill = "#E69F00", color = "black")
>plot(light_scatter(fl_with_shap, v = "DrivAge", type = "shap"), alpha = 0.3)
```

Model insights

- Decomposing a specific prediction into additive contributions from individual covariables gives a different view onto our XGBoost model.
- Like permutation importance, SHAP importance identifies the same variable `vehAge` as the strongest predictor for the XGBoost model.
- The SHAP dependence plot for `DrivAge` reveals a similar main effect as the partial dependence plot in Figure 7, increasing confidence not only in our XGBoost model but also in the model agnostic tools themselves.

6 Improving the GLM by interpretable machine learning

Throughout the last sections, we have collected relevant model specific insights gained thanks to applying model agnostic tools from XAI. In this section, we will use them to improve our GLM.

Building a strong GLM is a difficult task. What variables should be selected? How should they be transformed? How many degrees of freedom should be invested to well-represent each variable without introducing problematic overfit? What interactions should be modeled? A classical statistical approach for some of the necessary steps to consider is given in Harrell's book [17], for example. Fitting modern ML models is often easier and can be automated to a higher degree. Thanks to advances in interpretable ML, their drawback of being black boxes diminishes. Thus, replacing GLMs by modern ML models is becoming a viable option. This holds also in actuarial settings where model interpretability is often as important as getting most accurate predictions.

Even if internal processes or other reasons do not allow us to move away from classic GLMs, modern ML tools combined with XAI might help to guide the process of GLM modeling. The strategy could look as follows:

1. Run a simple GLM and a tuned black box ML model as benchmark.
2. Compare performance: How much room is left for improvement of the GLM? If the GLM does almost as good as the modern ML model, stop. Otherwise, use XAI to improve the GLM by following the next steps.
3. Study variable importance: Are the same variables important for the GLM and the ML benchmark model? Can certain variables be dropped?
4. Go through (main) effects plots for the strongest predictors in the ML benchmark. What numerical predictor needs more flexibility, e.g. by adding squared terms or splines? Can certain categorical classes be reasonably collapsed? Etc.
5. Study interaction strength: Do Friedman's H statistics suggest the addition of certain pairwise interactions?

6. Use these findings to improve the GLM. If performance is acceptably close to the ML benchmark, stop. Otherwise, iterate.

Ideally, these steps are iteratively performed on a validation set that is independent of training and test set. Only at the end, you assess the quality on the test set. Another excellent strategy to improve a GLM based on a neural network is described in Schelldorfer and Wüthrich [34].

In our example, the modern ML models are almost twice as performant as the GLM regarding relative deviance gain (Figure 3). This is despite the fact that the GLM is already using natural cubic splines to well represent the main effects of two major predictors. Still, there seems to be a lot of space for improvement. According to Figure 4, no covariable should be dropped. Furthermore, looking at partial dependence profiles (Figure 7) suggests investing more parameters in the main effect of logarithmic population density. Thus we represent this variable by natural cubic spline terms, just like the age of the driver and the age of the car. Figure 11 suggests adding interactions between vehicle's age and brand as well as between gas type and brand.

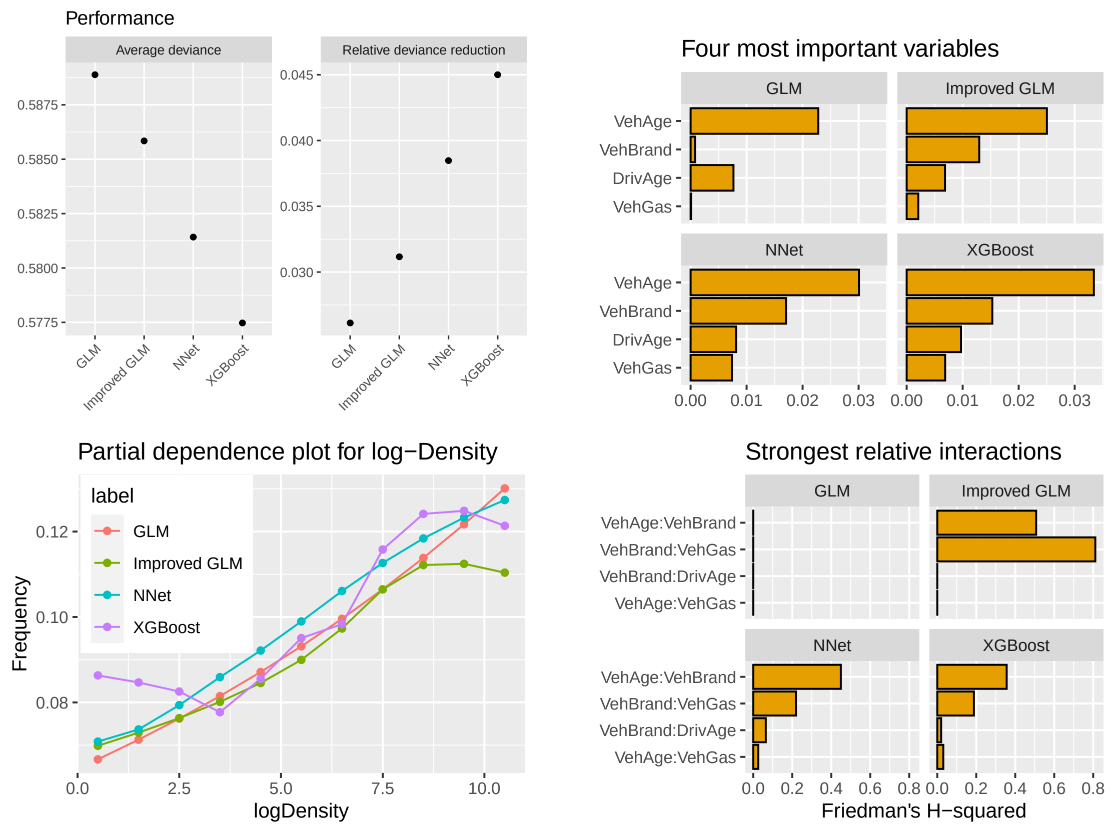


Figure 17: Some results with the improved GLM.

Listing 14 shows the code to update the GLM and to recalculate some of the results, see Figure 17. Modifying the GLM based on insights from the modern ML models has clearly improved the GLM. In particular, have a look at the largely increased permutation importance of `VehBrand` of the improved GLM, which stems from the added interaction terms. However, the performance in relative deviance gain is still behind the black box models indicating that there is still room for improvement.

Listing 14: Improving the GLM

```
>fit_glm2 <- glm(Freq ~ VehPower + VehBrand * VehGas + PolicyRegion +
>                  ns(DrivAge, 5) + VehBrand * ns(VehAge, 5) +
>                  ns(logDensity, 5),
>                data = train,
>                family = quasipoisson(),
>                weights = train[[w]])

# Setting up expainers
>fl_glm2 <- flashlight(
>  model = fit_glm2, label = "Improved␣GLM",
>  predict_function = function(fit, X) predict(fit, X, type = "response")
>)
>fls2 <- multiflashlight(list(fl_glm, fl_glm2, fl_nn, fl_xgb),
>                        metrics = metrics,
>                        data = test, y = y, w = w)
>fls2_log <- multiflashlight(fls2, linkinv = log)

# Some results
>plot(light_performance(fls2), geom = "point")
>plot(light_importance(fls2, v = x), top_m = 4, fill = "#E69F00", color = "black")
>plot(light_profile(fls2, v = "logDensity"))
>interact_rel_improved <- light_interaction(
>  fls2_log, v = most_important(imp, 4), take_sqrt = FALSE,
>  pairwise = TRUE, use_linkinv = TRUE, seed = 61)
>plot(interact_rel_improved, top_m = 4, fill = "#E69F00", color = "black")
```

7 Conclusions

Without tools from interpretable ML and XAI, choosing between a classic GLM and a modern ML model basically amounts to choosing between interpretability and predictive accuracy. Thanks to advances in XAI, using modern ML is more and more getting a viable option even in fields like actuarial pricing with a long-standing history of classic statistical modeling techniques. *Model agnostic* tools from XAI allow us to use the same interpretation techniques for (possibly complex) GLMs as for any ML black box model. This widely opens the gate to switch to modern ML altogether or, as an intermediate step as outlined in Section 6, help to improve GLMs by insights gained from XAI combined with modern ML techniques. R packages like **flashlight**, **iml**, and **DALEX** provide the corresponding tools in a user friendly way and without much programming effort.

We would like to close this tutorial with a sketch of the concept of modeling drawn in Figure 18 and the statement that the blackness of a model box seems no longer to be caused by the obscurity of the model itself but rather depends on the modeler switching on the light of the box.

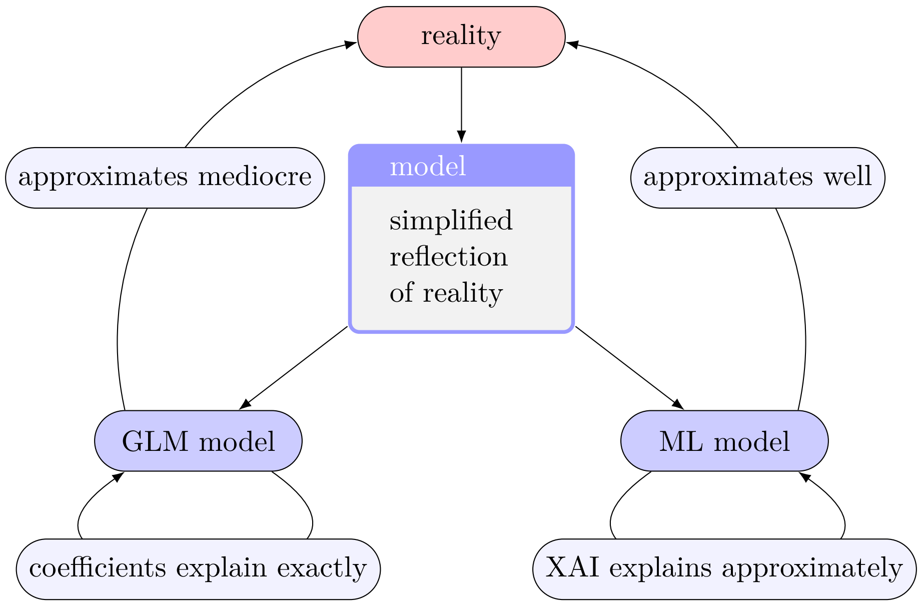

reality
approximates mediocre
model
simplified reflection of reality
approximates well
GLM model
ML model
coefficients explain exactly
XAI explains approximately

Figure 18: Sketch of model and explainability.

Acknowledgement

We are indepted to Jürg Schelldorfer and Mario Wüthrich for their exceptionally comprehensive reviews and their innumerable inputs which led to substantial improvement of this tutorial. Furthermore, we would like to thank the Data Science working group of the SAV and our company La Mobilière.

References

[1] F. J. Anscombe and John W. Tukey. "The Examination and Analysis of Residuals". In: *Technometrics* 5.2 (1963), pp. 141–160. DOI: 10.1080/00401706.1963.10490071. URL: http://www.jstor.org/stable/1266059.

[2] Daniel W. Apley and Jingyu Zhu. *Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models*. 2016. arXiv: 1612.08468 [stat.ME].

[3] Przemysław Biecek. "DALEX: Explainers for Complex Predictive Models in R". In: *Journal of Machine Learning Research* 19.84 (2018), pp. 1–5. URL: http://jmlr.org/papers/v19/18-416.html.

[4] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[5] Arthur Charpentier. *Computational Actuarial Science with R*. 1st ed. Chapman and Hall/CRC, 2014. ISBN: 978-1-466-59259-9. DOI: 10.1201/b17230.

[6] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[7] Francois Chollet and J. J. Allaire. *Deep Learning with R*. 1st. USA: Manning Publications Co., 2018. ISBN: 161729554X.

[8] Anna Veronika Dorogush et al. “Fighting biases with dynamic boosting”. In: *CoRR* abs/1706.09516 (2017). arXiv: 1706.09516.

[9] Timothy Dozat. *Incorporating Nesterov Momentum into Adam*. Tech. rep. CS229: Machine Learning, Stanford University, 2015. URL: http://cs229.stanford.edu/proj2015/054_report.pdf.

[10] Andrea Ferrario, Alexander Noll, and Mario V. Wüthrich. “Insights from Inside Neural Network”. In: *SSRN Electronic Journal* (2018). DOI: 10.2139/ssrn.3226852. URL: https://ssrn.com/abstract=3226852.

[11] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. *All Models are Wrong, but Many are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously*. 2018. arXiv: 1801.01489 [stat.ME].

[12] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *Ann. Statist.* 29.5 (Oct. 2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://projecteuclid.org/euclid.aos/1013203451.

[13] Jerome H. Friedman and Bogdan E. Popescu. “Predictive Learning via Rule Ensembles”. In: *The Annals of Applied Statistics* 2.3 (2008), pp. 916–954. DOI: 10.1214/07-AOAS148. URL: http://www.jstor.org/stable/30245114.

[14] Tilmann Gneiting. “Making and Evaluating Point Forecasts”. In: *Journal of the American Statistical Association* 106.494 (2011), 746–762. DOI: 10.1198/jasa.2011.r10138. arXiv: 0912.0902 [math].

[15] Alex Goldstein et al. “Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation”. In: *Journal of Computational and Graphical Statistics* 24.1 (2015), pp. 44–65. DOI: 10.1080/10618600.2014.907095.

[16] Alicja Gosiewska and Przemysław Biecek. *Do Not Trust Additive Explanations*. 2019. arXiv: 1903.11420 [cs.LG].

[17] Frank E. Harrell. *Regression Modeling Strategies*. 2nd ed. Springer Series in Statistics. Berlin, Heidelberg: Springer International Publishing, 2015. ISBN: 978-3-319-19424-0. DOI: 10.1007/978-3-319-19425-7.

[18] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 3146–3154. URL: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[19] Sebastian Lapuschkin et al. “Unmasking Clever Hans predictors and assessing what machines really learn”. In: *Nature Communications* 10.1 (Mar. 2019). DOI: 10.1038/s41467-019-08987-4.

[20] Mathias Lindholm et al. “Discrimination-Free Insurance Pricing”. In: *SSRN Electronic Journal* (2020). DOI: 10.2139/ssrn.3520676. URL: https://ssrn.com/abstract=3520676.

[21] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. arXiv: 1705.07874. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[22] Scott M. Lundberg et al. “From local explanations to global understanding with explainable AI for trees”. In: *Nature Machine Intelligence* 2.1 (2020), pp. 56–67. DOI: 10.1038/s42256-019-0138-9.

[23] Szymon Maksymiuk, Alicja Gosiewska, and Przemysław Biecek. *shapper: Wrapper of Python Library 'shap'*. R package version 0.1.2. 2019. URL: https://CRAN.R-project.org/package=shapper.

[24] Michael Mayer. *MetricsWeighted: Weighted Metrics, Scoring Functions and Performance Measures for Machine Learning*. R package version 0.5.0. 2020. URL: https://CRAN.R-project.org/package=MetricsWeighted.

[25] Michael Mayer. *flashlight: Shed Light on Black Box Machine Learning Models*. R package version 0.6.0. 2020. URL: https://github.com/mayer79/flashlight.

[26] Michael Mayer et al. “Estimation and updating methods for hedonic valuation”. In: *Journal of European Real Estate Research* 12.1 (2019), pp. 134–150. DOI: 10.1108/jerer-08-2018-0035.

[27] Peter McCullagh and John A. Nelder. *Generalized Linear Models*. Chapman and Hall/CRC, 1989. ISBN: 9780412317606.

[28] Christoph Molnar. *Interpretable Machine Learning*. 1st ed. Raleigh, North Carolina: Lulu.com, 2019. ISBN: 978-0-244-76852-2. URL: https://christophm.github.io/interpretable-ml-book.

[29] Christoph Molnar, Bernd Bischl, and Giuseppe Casalicchio. “iml: An R package for Interpretable Machine Learning”. In: *JOSS* 3.26 (2018), p. 786. DOI: 10.21105/joss.00786. URL: http://joss.theoj.org/papers/10.21105/joss.00786.

[30] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for Interpreting and Understanding Deep Neural Networks”. In: *CoRR* abs/1706.07979 (2017). arXiv: 1706.07979.

[31] Alexander Noll, Robert Salzmann, and Mario V. Wüthrich. “Case Study: French Motor Third-Party Liability Claims”. In: *SSRN Electronic Journal* (2018). DOI: 10.2139/ssrn.3164764. URL: https://ssrn.com/abstract=3164764.

[32] Arun Rai. “Explainable AI: from black box to glass box”. In: *Journal of the Academy of Marketing Science* 48 (Dec. 2019). DOI: 10.1007/s11747-019-00710-5.

[33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, 1135–1144. DOI: 10.1145/2939672.2939778.

[34] Jürg Schelldorfer and Mario V. Wüthrich. “Nesting Classical Actuarial Models into Neural Networks”. In: *SSRN Electronic Journal* (2019). DOI: 10.2139/ssrn.3320525. URL: https://ssrn.com/abstract=3320525.

[35] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning*. Cambridge University Press, 2014. ISBN: 1107057132. DOI: 10.1017/cbo9781107298019. URL: https://www.cse.huji.ac.il/~shais/UnderstandingMachineLearning.

[36] Lloyd S. Shapley. “17. A Value for n-Person Games”. In: *Contributions to the Theory of Games (AM-28), Volume II*. Ed. by Harold William Kuhn and Albert William Tucker. Princeton University Press, 1953, pp. 307–318. DOI: 10.1515/9781400881970-018.

[37] Mateusz Staniak and Przemysław Biecek. “Explanations of Model Predictions with live and breakDown Packages”. In: *The R Journal* 10.2 (2018), pp. 395–409. DOI: 10.32614/RJ-2018-072.