AI TOOLS FOR ACTUARIES
Chapter 7: Gradient Boosting Machines

Mathias Lindholm, Michael Mayer, Mario V. Wüthrich

2025-09-17

Additive boosting

Overview

- This section introduces *additive boosting*.
- Additive boosting is a simplified boosting algorithm that helps us to shape ideas and to understand the functioning of boosting more generally.
- The classical gradient boosting machine (GBM) is going to be introduced in the next section.

Idea of boosting

- Boosting is an *iterative updating scheme*, aiming at approximating the unknown true regression function $\boldsymbol{X} \mapsto \mu^*(\boldsymbol{X})$.
- In this iterative updating scheme of approximating μ^* by $\widehat{\mu}$, one *adds* new *base learners* in each step of the algorithm to $\widehat{\mu}$ to improve on the previous approximation.
- Base learners are simple regression functions that can efficiently be fitted.
- This makes boosting very closely connected to additive modeling; this is discussed in this section.
- In alignment with the GLM and the FNN chapters, we could adapt our additive boosting schemes to link functions g. For more clarity, we use the identity link $g(x) = x$ in this section (this will be changed in later sections).

Additive boosting algorithm

- Select a parametrized class of base learners $\mathcal{B} = \{\boldsymbol{X} \mapsto b(\boldsymbol{X}; \vartheta)\}_\vartheta$.
- Based on a learning sample $\mathcal{L} = (Y_i, \boldsymbol{X}_i, v_i)_{i=1}^n$, the additive boosting update in iteration $j \geq 1$ of the algorithm solves

$$\widehat{\vartheta}^{(j)} = \arg\min_\vartheta \sum_{i=1}^n \frac{v_i}{\varphi} L\left(Y_i, \widehat{\mu}^{(j-1)}(\boldsymbol{X}_i) + b(\boldsymbol{X}_i; \vartheta)\right),$$

  subject to existence and uniqueness; this considers the identity link $g(x) = x$.
- The updated regression function estimate in iteration $j \geq 1$ is

$$\widehat{\mu}^{(j)}(\boldsymbol{X}) = \widehat{\mu}^{(j-1)}(\boldsymbol{X}) + b(\boldsymbol{X}; \widehat{\vartheta}^{(j)}) = \sum_{s=0}^j b(\boldsymbol{X}; \widehat{\vartheta}^{(s)}),$$

  for initialization $\widehat{\mu}^{(0)}(\boldsymbol{X}) = b(\boldsymbol{X}; \widehat{\vartheta}^{(0)}) \equiv \widehat{\mu}_0 = \sum_{i=1}^n v_i Y_i / \sum_{i=1}^n v_i$.

- This describes the *additive boosting scheme* given in Algorithm 10.2 of Hastie, Tibshirani and Friedman (2009).
- The close connection to additive modeling is obvious. The update in iteration $j \geq 1$ tries to (additively) capture the remaining signal after having adjusted the regression function to

$$\widehat{\mu}^{(j-1)}(\boldsymbol{X}) \;=\; \widehat{\mu}_0 + \sum_{s=1}^{j-1} b(\boldsymbol{X}; \widehat{\vartheta}^{(s)}) \;=\; \sum_{s=0}^{j-1} b(\boldsymbol{X}; \widehat{\vartheta}^{(s)}).$$

- Thus, the next base learner $b(\boldsymbol{X}; \widehat{\vartheta}^{(j)})$ tries to find the weaknesses of the (previous) regression model $\boldsymbol{X} \mapsto \widehat{\mu}^{(j-1)}(\boldsymbol{X})$.
- This latter motivation/interpretation is precisely the same methodology as in the CANN approach discussed in the neural network chapter.

- In each boosting step j we have

$$\sum_{i=1}^{n} \frac{v_i}{\varphi} L\left(Y_i, \widehat{\mu}^{(j)}(\boldsymbol{X}_i)\right) \leq \sum_{i=1}^{n} \frac{v_i}{\varphi} L\left(Y_i, \widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right),$$

  supposed that the base learners $\mathcal{B}$ contain the constant zero.

- This is an in-sample loss improvement (on the learning sample $\mathcal{L}$).
- A stopping rule can be determined with cross-validation.
- *Shrinkage* often improves the predictive model.
- Select a *shrinkage factor* (*learning rate*) $\eta^{(j)} \in (0, 1]$, and replace the above update by

$$\widehat{\mu}^{(j)}(\boldsymbol{X}) = \widehat{\mu}^{(j-1)}(\boldsymbol{X}) + \eta^{(j)}\, b(\boldsymbol{X}; \widehat{\vartheta}^{(j)}).$$

- The intuition behind shrinkage is to avoid taking too large steps in each iteration of the algorithm, i.e., rather take two similar smaller ones.

Example: deterministic sine curve

- We study a deterministic sine curve example $x \mapsto y = \mu^*(x) = \sin(x)$.
- This example does not involve randomness.
- The (one-dimensional) covariate space is set to $x \in \mathcal{X} = [0, 2\pi]$.
- As base learners we use *tree stumps*, these are regression trees with only one single partition $(\mathcal{X}_{00}, \mathcal{X}_{01})$ of $\mathcal{X}$.
- These base learners take the following tree stump form

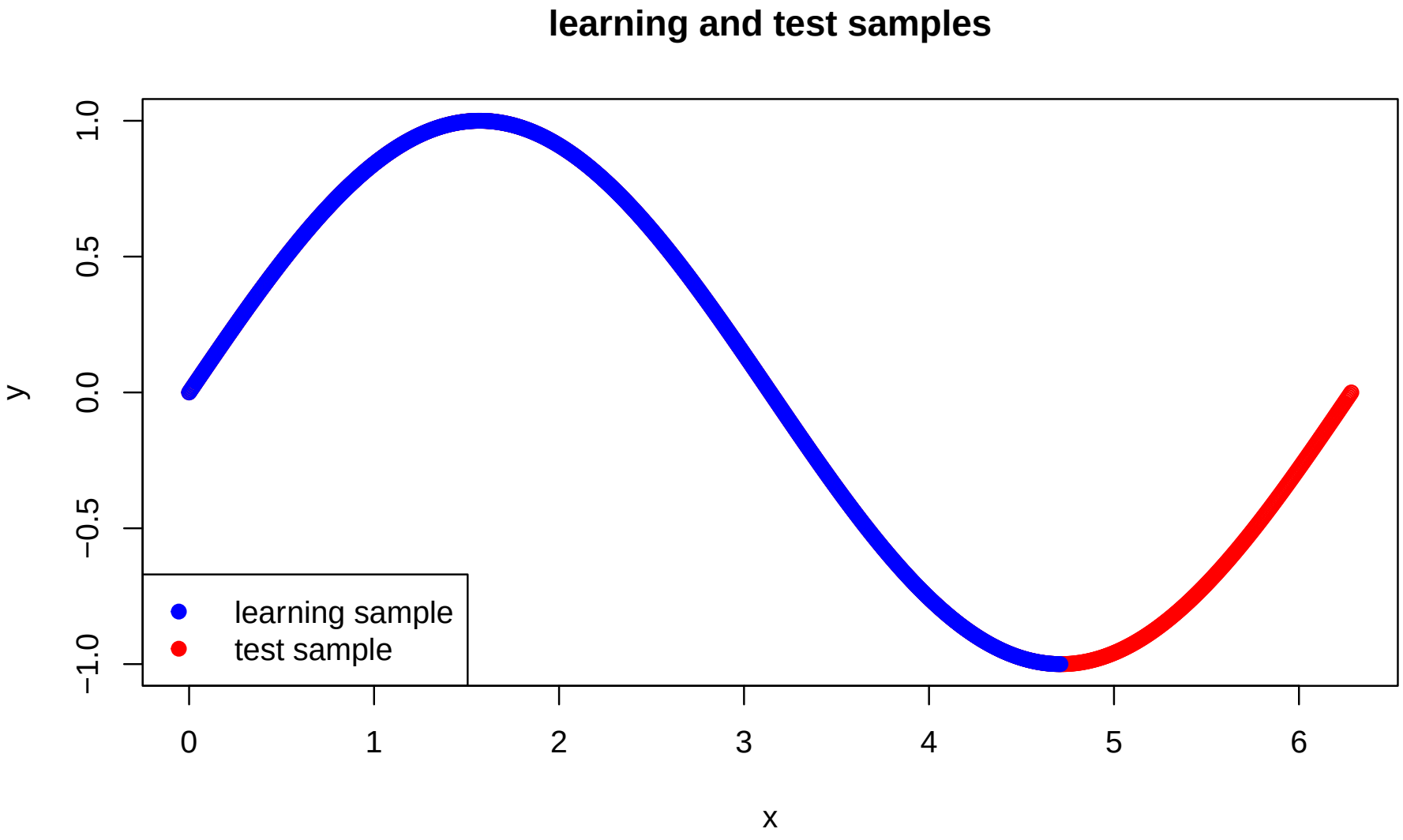
$$b(x; \vartheta) \;=\; \widehat{\mu}_{00}\, \mathbf{1}_{\{x \le c\}} + \widehat{\mu}_{01}\, \mathbf{1}_{\{x > c\}},$$

  with split constants $c \in \mathcal{X}$ and mean estimates $\widehat{\mu}_{00}, \widehat{\mu}_{01} \in \mathbb{R}$ on the two leaves.
- Thus, these base learners have parameter $\vartheta = (c, \widehat{\mu}_{00}, \widehat{\mu}_{01})$.
- We set learning and test sample $\mathcal{L} = [0, 1.5\pi]$ and $\mathcal{T} = (1.5\pi, 2\pi]$.

- We use the `xgboost` algorithm to compute this example.
- We initialize the example by selecting the true (sine) regression function, and learning and test samples, respectively.

```
library(xgboost)
library(tidyverse)

# Choose covariate space and true regression function
x <- seq(0, 2 * pi, length = 1000)
y <- sin(x)

# Define the learning sample
learn   <- 1:750
x_learn <- x[learn]
y_learn <- y[learn]
```

- The following graph shows the true regression function μ^*, and the learning and test sample split.

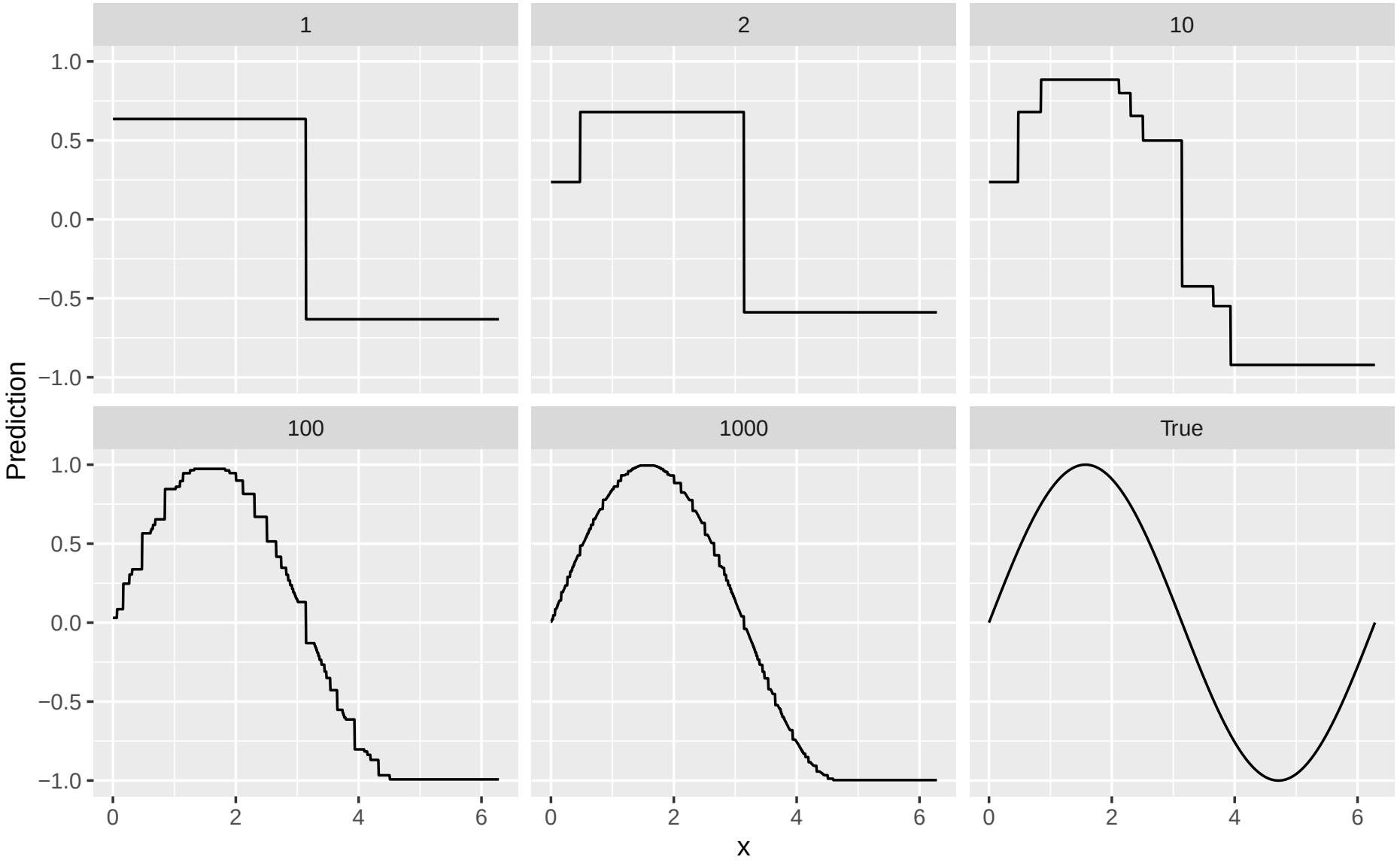


- True regression function (split into learning and test samples).

- Fitting tree stumps with XGBoost:

```
# Fitting boosted tree stumps (max_depth=1)
fit <- xgb.train(
   params  = list(learning_rate = 1, max_depth = 1),
   data    = xgb.DMatrix(cbind(x = x_learn), label = y_learn),
   nrounds = 1000
)

# Get predictions for different numbers of boosting iterations (rounds)
df_plot <- data.frame(x = x, True = y)
J <- c(1, 2, 10, 100, 1000)
#
for (j in J) {
  df_plot[as.character(j)] <- predict(fit, cbind(x=x), iteration=c(1,j+1))
}
```

- The following graph shows the fitting results after different rounds of boosting $j \in \{1, 2, 10, 100, 100\}$.



- Good in-sample fit, but (clearly) extrapolation does not work!

Gradient boosting machines

Overview

- This section introduces *gradient boosting machines* (GBMs).
- GBMs include the state-of-the-art boosting algorithms, with its specialized versions XGBoost, LightGBM or CatBoost.

- Above, boosting was introduced as an iterative updating scheme, iteratively improving the fit of (and the complexity in) the approximation of $\mu^*(\boldsymbol{X})$ using base learners $\mathcal{B} = \{b(\cdot; \vartheta)\}_\vartheta$.
- A different route forward is to consider learning an unknown function using *functional gradient descent*. We discuss this next.

Learning an unknown real-valued parameter

We take this introduction to functional gradient descent in different steps.

- We start with a single instance i, and we fit a one-dimensional real-valued parameter $\vartheta \in \mathbb{R}$ to the response Y_i, giving the mean estimate $\mu = g^{-1}(\vartheta)$ for Y_i, where g is a pre-selected link function.
- For the moment, the (generic) parameter ϑ plays the role of the link transformed mean estimate $\vartheta = g(\mu)$ of Y_i.
- To ease the notation of this exposition, we define

$$L^g(Y_i, \vartheta) = L(Y_i, g^{-1}(\vartheta)).$$

- Assume the derivative $\nabla_\vartheta L^g(Y_i, \vartheta)$ w.r.t. ϑ exists.

Gradient descent algorithm

- We aim at finding an optimal parameter $\widehat{\vartheta}$ with gradient descent.
- The standard gradient descent step at algorithmic time j gives update

$$\widehat{\vartheta}^{(j)} = \widehat{\vartheta}^{(j-1)} - \eta^{(j)} \nabla_{\vartheta} L^{g} \left(Y_i, \widehat{\vartheta}^{(j-1)} \right),$$

  for a given learning rate $\eta^{(j)} > 0$.
- For sufficiently small learning rates $\eta^{(j)} > 0$, there is an (in-sample) loss decrease

$$L^{g} \left(Y_i, \widehat{\vartheta}^{(j)} \right) \leq L^{g} \left(Y_i, \widehat{\vartheta}^{(j-1)} \right).$$

- This describes the classical gradient descent algorithm as introduced in the neural network chapter. In the sequel, we are going to present a functional version thereof.

Optimal learning rates

- One option to select the learning rates $\eta^{(j)} > 0$ is a so-called *full relaxation optimization*, which corresponds to the *line search*

$$\widehat{\eta}^{(j)} = \underset{\eta>0}{\arg\min}\, L^g\left(Y_i, \widehat{\vartheta}^{(j-1)} - \eta\, \nabla_\vartheta L^g\left(Y_i, \widehat{\vartheta}^{(j-1)}\right)\right).$$

- For more on this, including conditions for convergence and convergence rates; see Nesterov (2018).

Functional gradient boosting

- Learning an unknown function μ instead of an unknown parameter ϑ can be approached quite similarly.
- Again, consider a single instance i with observation $(Y_i, \boldsymbol{X}_i, v_i)$.
- Using the above abbreviation, we have

$$L(Y_i, \mu(\boldsymbol{X}_i)) = L^g\left(Y_i, g(\mu(\boldsymbol{X}))\right).$$

- Differentiating the right-hand side w.r.t. $\vartheta := g(\mu(\boldsymbol{X}_i))$ allows one to rewrite the standard gradient descent update as

$$g\left(\widehat{\mu}^{(j)}(\boldsymbol{X}_i)\right) = g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right) - \eta^{(j)} \nabla_\vartheta L^g\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right)\right).$$

- This coincides with the estimation of ϑ from above.

- Bringing the link g to the other side, suggests the gradient descent update

$$\widehat{\mu}^{(j)}(\boldsymbol{X}_i) = g^{-1}\left[g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right) - \eta^{(j)}\,\nabla_\vartheta L^g\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right)\right)\right].$$

- The above gradient descent update considers one single instance i.
- Applying this iteration to each instance $1 \leq i \leq n$ will converge to the *saturated model* (under suitable learning rates), naturally, resulting in in-sample over-fitting.
- The second crucial step of the GBM is to approximate the gradients

$$\nabla_\vartheta L^g(Y_i, g(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i))), \qquad \text{for } 1 \leq i \leq n,$$

  by base learners $\mathcal{B} = \{b(\cdot;\vartheta)\}_\vartheta$ to prevent from over-fitting.
- This implies *regularization* to the fitting problem, because the base learner $b(\cdot;\vartheta)$ needs to simultaneously approximate *all* gradients.

Working responses and their approximation

- Define the *working responses* for $1 \leq i \leq n$ in iteration $j \geq 1$ by

$$r_i^{(j)} = -\nabla_\vartheta L^g \left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i) \right) \right).$$

- For GBMs, one fits a base learner from the parametrized class $\mathcal{B} = \{b(\cdot;\vartheta)\}_\vartheta$ to the *working sample* $\mathcal{L}^{(j)} = (r_i^{(j)}, \boldsymbol{X}_i, v_i)_{i=1}^n$.
- This is obtained by the weighted least square

$$\widehat{\vartheta}^{(j)} = \arg\min_\vartheta \sum_{i=1}^n v_i \left(r_i^{(j)} - b(\boldsymbol{X}_i;\vartheta) \right)^2.$$

- The weighting with $v_i > 0$ seems natural, as it is also obtained in additive boosting. However, some care is needed because it is not always clear whether the available GBM libraries use this weighting.

- For the Poisson EDF with log-link, one can either model claim frequencies $Y = N/v$ using weights v, or one can model claim counts N using an offset $\log(v)$; see our previous notebook on the GLM.
- With existing GBM libraries, it needs to be checked which options are available for Poisson claims frequencies versus claim counts modeling.

Gradient boosting step

- Using the optimal base learner gives the *GBM step* in iteration $j \geq 1$

$$\widehat{\mu}^{(j)}(\boldsymbol{X}_i) = g^{-1}\left(g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right) + \eta^{(j)}\, b(\boldsymbol{X}_i; \widehat{\vartheta}^{(j)})\right).$$

- (Generalized) additive boosting (discussed above) and the GBM lead to the same updating structure. The main difference between the two is how the next optimal update $b(\boldsymbol{X}_i; \widehat{\vartheta}^{(j)})$ from $\mathcal{B}$ is selected.
- We equip the GBM step with a line search for the learning rate $\eta^{(j)}$

$$\widehat{\eta}^{(j)} = \underset{\eta>0}{\arg\min} \sum_{i=1}^{n} \frac{v_i}{\varphi} L\left(Y_i, g^{-1}\left(g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right) + \eta\, b(\boldsymbol{X}_i; \widehat{\vartheta}^{(j)})\right)\right).$$

- Iterating this for $j \geq 1$ gives the GBM algorithm of Friedman (2001).

Remarks

- The distribution of the response Y and/or the selection of the strictly consistent loss function L for mean estimation only enters the computation of the working residuals $r_i^{(j)}$.
- The boosting step for finding the optimal base learners is then fully based on weighted least squares.
- This makes gradient boosting widely applicable because the boosting step itself does not need to allow for different loss functions (once the working residuals are computed).
- The different GBM libraries may however be restricted in terms of available link functions g.

Tree-based gradient boosting machines

- The most popular classes $\mathcal{B} = \{b(\cdot; \vartheta)\}_\vartheta$ of base learners are low depth and low interaction regression trees of a fixed cardinality.
- They are simple to implement, easy and fast to compute, and they provide an excellent predictive performance to GBMs.
- Recall, a regression tree gives a finite partition of the covariate space $\mathcal{X} = \bigcup_{t \in \mathsf{T}} \mathcal{X}_t$, with a finite index set T of cardinality $|\mathsf{T}|$.
- Using a fixed small cardinality $|\mathsf{T}|$, we select the class $\mathcal{B}$ of base learners by

$$\boldsymbol{X} \;\mapsto\; b(\boldsymbol{X}; \vartheta) = \sum_{t \in \mathsf{T}} \mu_t \, \mathbf{1}_{\{\boldsymbol{X} \in \mathcal{X}_t\}};$$

parameter ϑ collects the full description of the characterization of the binary regression tree of cardinality $|\mathsf{T}|$.

Line search for the learning rate

- Revisiting the above line search for optimal learning rates $\eta^{(j)} > 0$.
- The line search is replaced by directly updating the piecewise constant estimates $(\mu_t)_{t \in \mathsf{T}}$ on every leaf $\widehat{\mathcal{X}}_t^{(j)}$, i.e.,

$$\widehat{\mu}_t^{(j)} \;=\; \operatorname*{arg\,min}_{m \in \mathbb{R}} \sum_{i:\, \boldsymbol{X}_i \in \widehat{\mathcal{X}}_t^{(j)}} v_i \, L^g \left(Y_i, g \left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i) \right) + m \right);$$

  this relies on *first* having fitted a regression tree to the working responses to obtain the covariate space partition $(\widehat{\mathcal{X}}_t^{(j)})_{t \in \mathsf{T}}$.

- The *tree-based gradient boosting update* in iteration $j \geq 1$ of the GBM algorithm then is

$$\widehat{\mu}^{(j)}(\boldsymbol{X}) = g^{-1} \left(g \left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}) \right) + \sum_{t \in \mathsf{T}} \widehat{\mu}_t^{(j)} \, \mathbf{1}_{\{\boldsymbol{X} \in \widehat{\mathcal{X}}_t^{(j)}\}} \right).$$

Conclusions on gradient boosting machines

- Tree-based GBMs are very popular, and they can be found in off-the-shelf software.
- Tree-based GBMs have an excellent predictive performance on tabular data.
- To promote interactions, one should use trees with at least two splits, i.e., not tree stumps.
- For the log-link $g = \log(\cdot)$ and with tree stumps, one receives a multiplicative model, similar to a GLM.
- It is common to add yet another learning rate (shrinkage factor) in the updating step so that the updates are only moderate in each step.
- Hyper-parameters are selected with cross-validation and (randomized) grid search.

Explainability - variable importance

- A popular explainability measure based on trees – called *variable importance score* (VI-score) – was introduced in Breiman *et al.* (1984).
- Let $S(k;\mathsf{T})$ denote which of the q covariate components $(X_l)_{l=1}^q$ is used in split k in tree T, thus, $S(k;\mathsf{T}) \in \{1,\ldots,q\}$.
- The VI-score for covariate component $1 \leq l \leq q$ for a tree T is defined by

$$\mathrm{VI}_l(\mathsf{T}) = \sum_{k=1}^{|\mathsf{T}|-1} \widehat{i}_k(\mathsf{T})^2 \, \mathbf{1}_{\{S(k;\mathsf{T})=l\}},$$

  where $\widehat{i}_k(\mathsf{T})^2$ corresponds to the loss decrease in the k-th split of the regression tree T construction.
- This aggregates all the decreases in loss that can be attributed to covariate component X_l leading to tree T.

- Assume that the GBM regression function $\widehat{\mu}^{(j_{\max})}(\boldsymbol{X})$ has been obtained by boosting $j_{\max} \in \mathbb{N}$ regression trees $(\mathsf{T}_j)_{j=1}^{j_{\max}}$.
- This suggests the total VI-score for covariate component $1 \leq l \leq q$

$$\mathsf{VI}_l = \sum_{j=1}^{j_{\max}} \mathsf{VI}_l(\mathsf{T}_j).$$

- In practice, these VI-scores are often normalized to sum to one.
- When using these VI-scores in GBMs, the importance is measured on the tree scale, hence, focusing on the loss decrease w.r.t. the *gradient approximations*, not w.r.t. the (empirical) loss given by the deviance loss function.
- For more on VI-scores; see Chapter 10.13 of Hastie, Tibshirani and Friedman (2009); note that this reference additionally scales the total VI-score with $1/j_{\max}$.

MTPL example

- We revisit the French MTPL claims frequency example from our previous notebooks.
- We apply a GBM to this data.
- We use the version LightGBM. Besides XGBoost and CatBoost, this is one of the main GBM implementations. These versions are presented in more detail in the next section (after the example).
- To improve interpretability, we do not use all covariates, and we merge the smaller levels of the categorical covariates. Naturally, the results will not be fully competitive, but they will be more accessible (for this teaching material).

Loading and pre-processing the data

```
library(arrow)
library(tidyverse)
library(lightgbm)    # we use the lightgbm library

freMTPL2freq <- read_parquet("../Data/freMTPL2freq.parquet")

# Pre-processing the covariates to improve interpretability
prep <- freMTPL2freq %>%
  mutate(
    frequency = ClaimNb / Exposure,     # claims frequency Y regression
    Region    = fct_lump(Region, 6),    # merge the smallest levels
    VehBrand  = fct_lump(VehBrand, 5),  # merge the smallest levels
    VehAge    = pmin(25, VehAge),       # censor vehicle age
    lDensity  = log(Density)            # transform for better interpretation
  )
```

- Prepare for the LightGBM fitting.

```
# Learning and test sample partition (as in Wuthrich-Merz, Springer 2023)
learn <- subset(prep, LearnTest == "L")
test  <- subset(prep, LearnTest == "T")

# Considered covariates (for explainability we consider only a subset)
features <- c("DrivAge", "lDensity", "VehAge", "VehPower", "Region", "VehBrand",
↪  "VehGas")

# Create a LightGBM (lgb) learning data set
dat <- lgb.Dataset(
    data.matrix(learn[features]),
    label = learn$frequency,       # this models frequencies with weights
    weight = learn$Exposure,       # case weights v>0
    categorical_feature = c("Region", "VehBrand"),
    params = list(feature_pre_filter = FALSE)
)
```

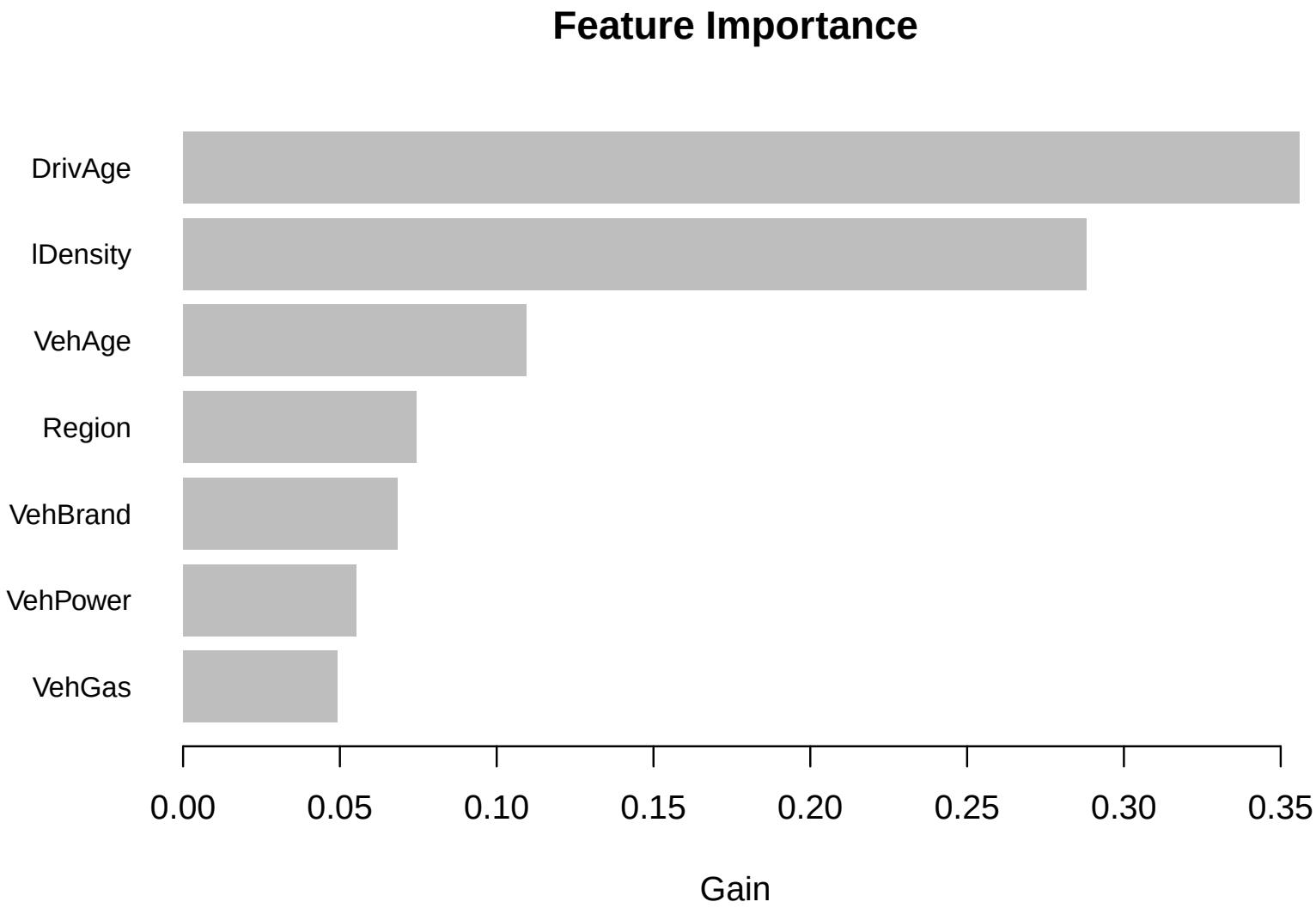
```
# Parameters were selected by randomized 5-fold CV search on the training data,
↪  using early stopping to determine the number of rounds (j_max); this is not
↪  shown here for brevity.

params_freq <- list(
  objective = "poisson",
  learning_rate = 0.02,
  num_leaves = 31,
  reg_lambda = 0,
  reg_alpha = 2,
  colsample_bynode = 0.8,
  subsample = 0.8,
  min_child_samples = 20,
  min_split_gain = 0.1,
  poisson_max_delta_step = 0.1
)
nrounds <- 241
```

- This sets all the hyper-parameters for the LightGBM algorithm, some of these will be explained in more detail below.

Fitting LightGBM

```
# Fit lightgbm model
lgb_freq <- lgb.train(params = params_freq, data = dat, nrounds = nrounds)

# Poisson deviance loss (generally scaled with 10^2)
Poisson.Deviance <- function(pred, obs, weight){
  10^2 * 2*(sum(pred)-sum(obs)+sum(log((obs/pred)^(obs))))/sum(weight) }

#
pred.L <- predict(lgb_freq, data.matrix(learn[features]))*learn$Exposure
pred.T <- predict(lgb_freq, data.matrix(test[features]))*test$Exposure
round(c(Poisson.Deviance(pred.L, learn$ClaimNb, learn$Exposure),
↪  Poisson.Deviance(pred.T, test$ClaimNb, test$Exposure)), 3)

# Relative (split gain) VI-score (variable importance measure)
lgb.plot.importance(lgb.importance(lgb_freq), measure = "Gain")
```

```
[1] Poisson deviance losses: 46.297 and 46.896
```



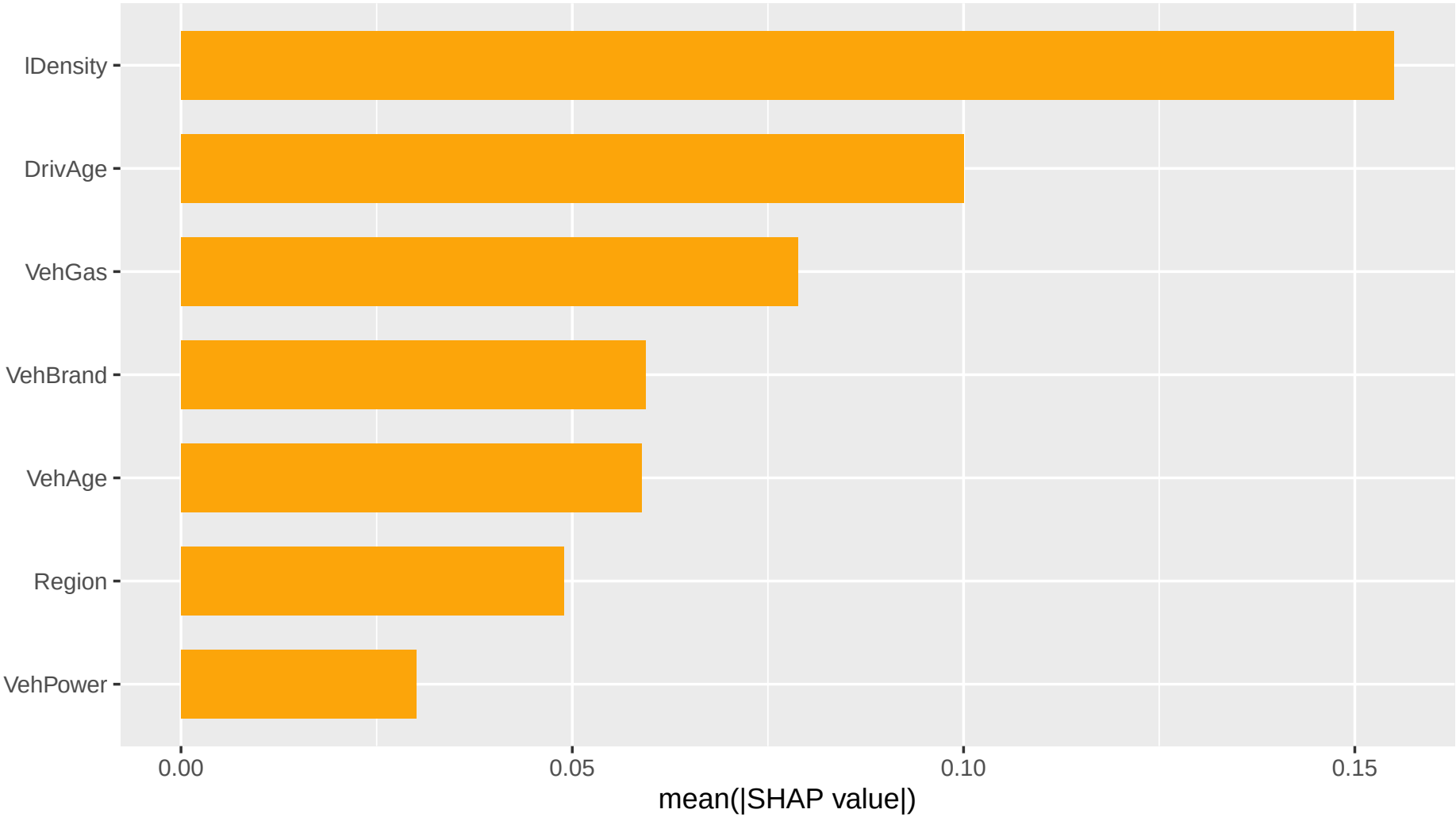
- VI-score (normalized to 1): 35% of the total decrease in loss is allocated to `DrivAge`, and roughly 30% to `(log-)Density`.

SHAP variable importance

- We close this example with a SHAP variable importance analysis. SHAP is introduced in a later chapter (on explainability), and we refer to that chapter for technical details.
- `TreeSHAP` is a very efficient way of computing SHAP in tree-based regression functions.

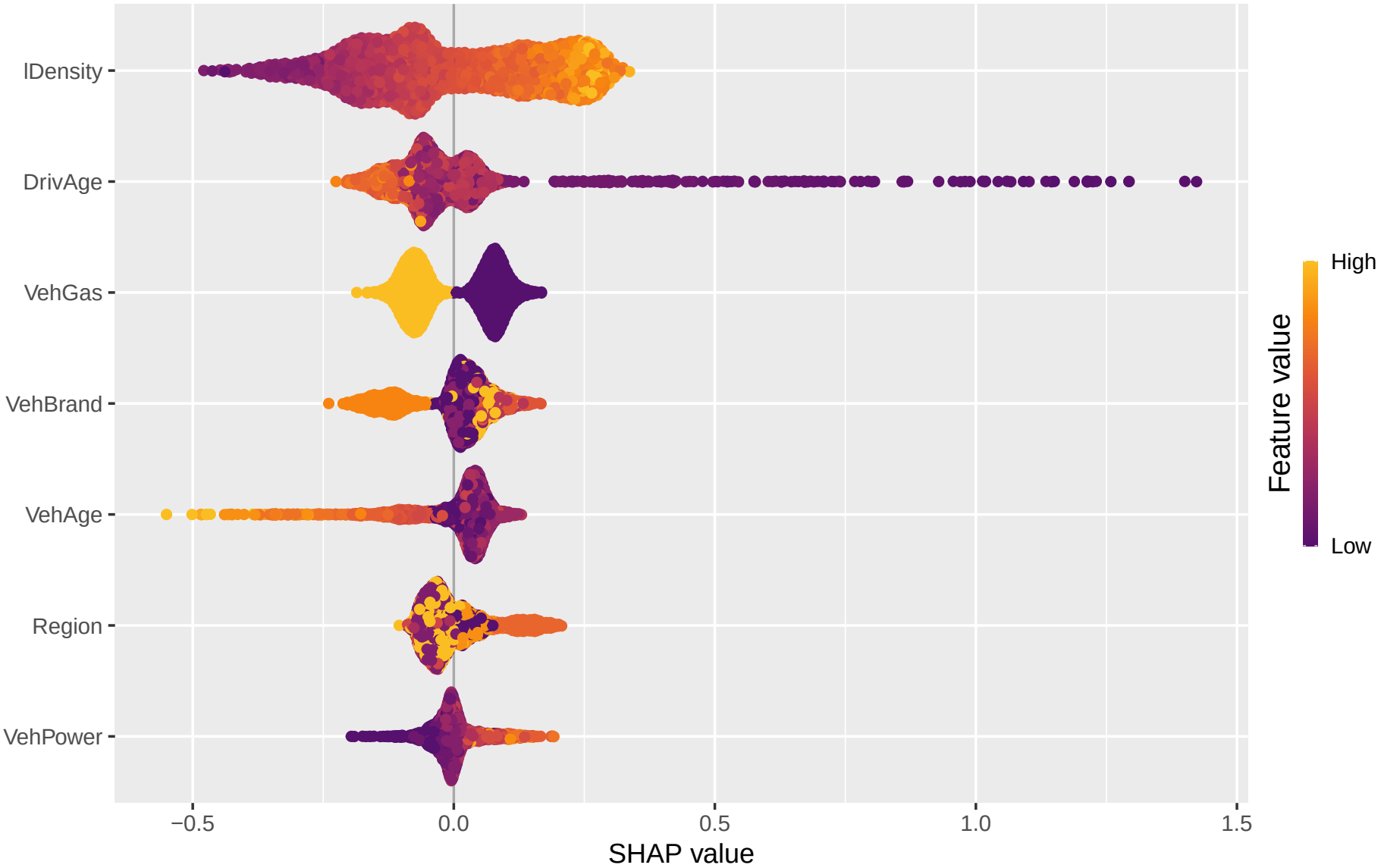
```
# SHAP analysis on log scale
library(shapviz)

# For computational reasons, randomly select 2000 instances (to explain)
to_explain <- sample_n(learn[features], size = 2000)
shap_freq  <- shapviz(lgb_freq, X_pred = data.matrix(to_explain), X = to_explain)

# We present the following 3 plots
sv_importance(shap_freq)                    # SHAP importance
sv_importance(shap_freq, kind = "bee")      # SHAP bee swarm plot
sv_dependence(shap_freq, v = features[1:6]) # marginals & interactions
```

- SHAP variable importance:

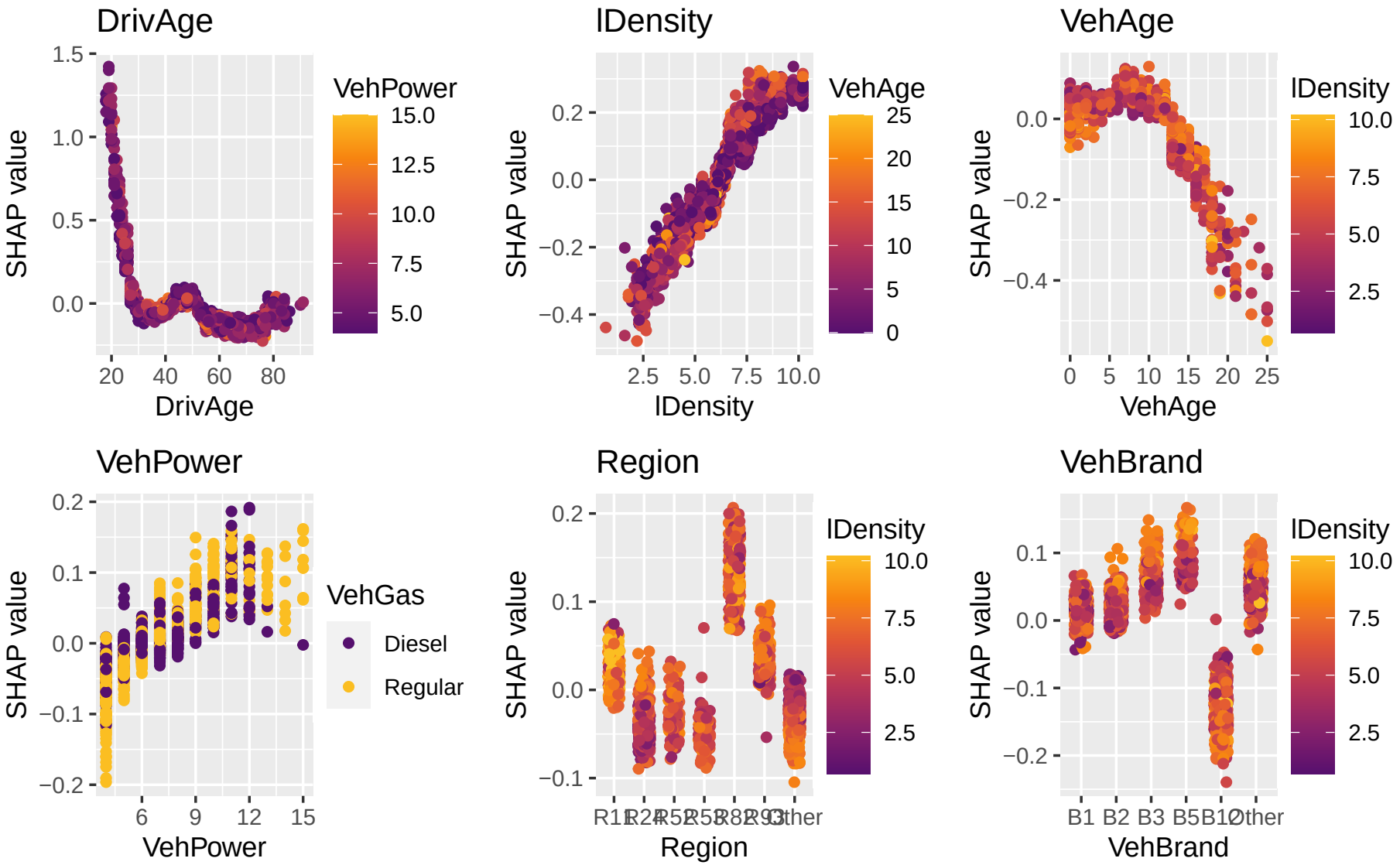


- This slightly differs from the VI-score from the GBM.

- Bee swarm plot of individual SHAP values:



- Marginalized SHAP values (and interactions through the coloring):



State-of-the-art gradient boosting machines

Overview

- The previous sections presented the basic ideas underpinning additive boosting and gradient boosting.
- This was done with an extra focus on the situation when using low-cardinality trees as base learners.
- In practice, usually different high-speed versions of GBMs are used, such as **XGBoost**, **LightGBM** or **CatBoost**.
- We present the first two to give some flavor of what kind of short-cuts these fast GBM implementations involve.

XGBoost and LightGBM

- XGBoost of Chen and Guestrin (2016) was open-sourced in 2014.
- LightGBM was open-sourced by Ke *et al.* (2017) after XGBoost.
- LightGBM introduced some additional features, which have later been adapted by XGBoost. Therefore, we describe these additional features in the LightGBM paragraph.
- All these packages use short-cuts to improve speed and accuracy, and the results typically differ from the classical GBM, and they also differ between the different algorithms.
- In insurance pricing, often LightGBM is used.
- In online competitions, also XGBoost and CatBoost are frequently found. There is no clear preference for one of these algorithms.
- Probably, LightGBM is the fastest boosting algorithm on CPUs, and XGBoost and CatBoost are very fast on GPUs.

XGBoost

- XGBoost of Chen and Guestrin (2016) considers a functional second order approximation to the loss function.
- For a single instance i, there is the second order Taylor expansion (we drop the 0th order term)

$$
\begin{aligned}
& L^g\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right) + b(\boldsymbol{X}_i; \vartheta)\right) \\
& \quad \propto \quad \nabla_\vartheta L^g\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right)\right) b(\boldsymbol{X}_i; \vartheta) \\
& \qquad\qquad + \frac{1}{2} \nabla^2_\vartheta L^g\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i)\right)\right) b(\boldsymbol{X}_i; \vartheta)^2 \\
& =: \quad L^g_{\text{XGB}}\left(Y_i, g\left(\widehat{\mu}^{(j-1)}(\boldsymbol{X}_i); b(\boldsymbol{X}_i; \vartheta)\right)\right),
\end{aligned}
$$

where ∇_ϑ denotes the first derivative and ∇^2_ϑ the second derivative w.r.t. the second loss component.

- In contrast to the above gradient descent steps, this second order Taylor expansion is related to a Newton step, and the working residuals are suitably scaled by their Hessians before being approximated by the base learners.
- This is rather similar to Fisher's scoring method for GLM fitting.
- By using low-cardinality trees as base learners in $L^g_{\text{XGB}}(\cdot)$, the leaf values for a given partition of the covariate space are explicitly given. This avoids time-consuming line searches for optimal learning rates.
- Moreover, XGBoost introduced a built-in handling of missing values, which is now standard in most other implementations. The decision where to split does not take into account missing values, but once a split is made, the algorithm memorizes whether sending missing values to the left or the right child leads to better loss improvements.

LightGBM

LightGBM of Ke *et al.* (2017) introduced the following key innovations:

- Histogram-based algorithm: LightGBM discretizes continuous covariates into bins. Instead of evaluating all possible split points on continuous covariates, it only loops over these bins, significantly reducing computational costs.
- Internal memory footprint is reduced by replacing floating-point numbers with integers (the bin indices).
- Categorical feature support: LightGBM offers different options to treat (integer encoded) categorical features without the need of one-hot encoding by using an efficient split-finding algorithm, similar to target encoding.

- Leaf-wise tree growth: While XGBoost grows trees level-wise (all leaves at the same depth are split before moving to the next level), LightGBM grows trees leaf-wise. While this does not guarantee better performance, it may help speeding up the training procedure.
- Further innovations include gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB).

Nowadays, many of the innovations of LightGBM are also implemented in XGBoost (and other GBM libraries), e.g., histogram binning, leaf-wise tree growth, and (initial) categorical feature support.

GBMs vs. neural networks

- In view of the many successes in online competitions, it is fair to say that typically GBMs outperform networks on tabular data.
- GBM learning algorithms are specifically designed to focus on the weaknesses of the present model which gives a rather targeted and fast model improvement, whereas networks more strongly suffer from over-fitting issues resulting in under-fitting (through a too early stopping).
- Tree-based GBMs do not lead to continuous regression functions, and extrapolation beyond the observed covariates does not work; see sine example above. Networks are more flexible in this regard.
- When it comes to unstructured data, time-series data, longitudinal data or images, we see advantages in networks, as these types of regression functions can deal with a huge variety of different data types.

Hyper-parameter tuning in GBMs

- As objective function, we select a strictly consistent loss function L for mean estimation reflecting the properties of the responses Y.
- The main hyper-parameters correspond
  - the number of leaves of the considered trees, $|\mathrm{T}|$,
  - the number of boosting iterations $j_{\max}$, and
  - the learning rates.
- These hyper-parameters interact, and they cannot be chosen in isolation. These hyper-parameters are selected by a grid search with (cross-)validation or by a randomized grid search.
- A rule of thumb is that one would like to receive roughly 100 to 1000 trees T_j. Fewer trees often results in too simplistic regression functions, and having more trees is computationally demanding.

- To quickly calibrate a suitable learning rate: by halving the learning rate, one should roughly double the number of boosting steps. Because one needs to add twice the same tree to receive the same step size in the case of halving the learning rate. This helps to select the learning rate to result in roughly 100 to 1000 trees T_j with an optimal minimal cross-validation loss.
- There are further hyper-parameters involved, e.g., for regularization, for a minimal leaf size, the bagging fraction, or a minimal loss improvement to continue splitting, etc. Many of the boosting libraries have further options like, e.g., interaction constraints and monotonicity constraints; these may be helpful for explainability.
- There are specialized tools like Optuna that are designed for hyper-parameter optimization; see Akiba *et al.* (2019).

Copyright

- © The Authors
- This notebook and these slides are part of the project "AI Tools for Actuaries". The lecture notes can be downloaded from:

https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5162304

References I

Akiba, T. *et al.* (2019) 'Optuna: A next-generation hyperparameter optimization framework', in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Available at: https://arxiv.org/abs/1907.10902.

Breiman, L. *et al.* (1984) *Classification and regression trees*. Chapman; Hall/CRC. Available at: https://doi.org/10.1201/9781315139470.

Chen, T. and Guestrin, C. (2016) 'XGBoost: A scalable tree boosting system', *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794. Available at: https://doi.org/10.1145/2939672.2939785.

References II

Friedman, J. (2001) 'Greedy function approximation: A gradient boosting machine', *Annals of Statistics*, 29/5, pp. 1189–1232. Available at: https://www.jstor.org/stable/2699986.

Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The elements of statistical learning: Data mining, inference, and prediction*. 2nd edn. Springer. Available at: https://link.springer.com/book/10.1007/978-0-387-84858-7.

Ke, G. *et al.* (2017) 'LightGBM: A highly efficient gradient boosting decision tree', *Advances in Neural Information Processing Systems*, 30, pp. 3146–3154. Available at: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

References III

Nesterov, Y. (2018) *Lectures on convex optimization*. Springer. Available at: https://link.springer.com/book/10.1007/978-3-319-91578-4.