

# DET RAN OTE

DETRA NOTE 2023-3

## BOOSTED POISSON REGRESSION TREES: A GUIDE TO THE BT PACKAGE IN R

---

By Gireg Willame, Julien Trufin and Michel Denuit

## DISCLAIMER

The content of the Detra Notes for a pedagogical use only. Each business case is so specific that a careful analysis of the situation is needed before implementing a possible solution. Therefore, Detra Analytics does not accept any liability for any commercial use of the present document. Of course, the entire team remain available if the techniques presented in this Detra Note required your attention.

## ABSTRACT

Thanks to its outstanding performances, boosting has rapidly gained wide acceptance among actuaries. Hainaut et al. (2022) established that boosting can be conducted directly on the response under Tweedie loss function and log-link, by adapting the weights at each step. This is particularly useful to analyze low counts (typically, numbers of reported claims at policy level in personal lines). Huyghe et al. (2022) adopted this approach to propose a new boosting machine with cost-complexity pruned trees. In this approach, trees included in the score progressively reduce to the root-node one, in an adaptive way. This paper reviews these results and presents the new BT package in R contributed by Willame (2022), which is designed to implement this approach for insurance studies. A numerical illustration demonstrates the relevance of the new tool for insurance pricing.

**Keywords:** Risk classification, Boosting, Adaptive Boosting, Regression Trees.

# 1 Introduction and motivation

Boosting emerged from the field of machine learning and became rapidly popular among insurance analysts. In each iteration, boosting fits a base, or weak learner that improves the fit of the overall model so that the ensemble arrives at an accurate prediction. Thus, the score is not specified by the actuary and estimated at once, as in generalized linear models (GLMs) or generalized additive models (GAMs) but it is built sequentially. In every boosting iteration, only the best-performing base learner and hence the best-performing feature is included in the final model. Also, only a small amount of the fit of the best-performing base-learner is added to the current additive score. This is achieved by multiplying the new effect entering the score with a shrinkage coefficient (a typical value is 0.1). Boosting is particularly effective when base learners are trees of limited depth.

Boosting is often applied on gradients of the loss function, that is, on the gradients of the deviance function in insurance applications. Instead of maximizing the log-likelihood associated with the response, gradient boosting applies a least-squares principle on its gradients. In this form, boosting has become very popular among data analysts and several packages available in open-source software implement highly effective boosting algorithms. There have been numerous applications of gradient boosting to insurance pricing in the last decade. We refer the interested reader to Lee and Lin (2018) for an extensive review of the different boosting algorithms that have been proposed so far, as well as to the references listed in Hainaut et al. (2022).

However, gradient boosting beyond the Normal case typically faces the same problem that lead to the adoption of GLMs for insurance applications. In Poisson regression for instance, gradient boosting consists in fitting raw residuals (corresponding to numbers of claims minus the expected claim frequencies) by least squares. Least-squares principle is known to be outperformed by Poisson regression for low counts, as those encountered in pricing personal lines. Indeed, for low expected claim frequencies, raw residuals are concentrated around integer values. For severities modeled by the Gamma distribution, gradients obey a translated Gamma distribution, which can be markedly skewed and rules out the least-squares principle. For general Tweedie response corresponding to compound Poisson sums with Gamma-distributed terms, the abundance of zeroes and the mixture of Gamma distributions for positive outcomes also result in highly asymmetric gradients and poor performances of least-squares estimation. This suggests that applying the least-squares principle on gradients is not effective and exposes actuaries to the same deficiency that led to the massive adoption of GLMs in the 1990s, in lieu of Gaussian linear models.

Hainaut et al. (2022) established that likelihood-based boosting can easily be achieved when trees are used as weak learners under Tweedie loss with log-link, provided the responses are re-weighted and re-scaled at each boosting step. Hence, responses can be used directly and there is no need to replace them with gradients as long as Tweedie loss is adopted, with log-link. This is the approach adopted in the present paper.

Standard boosting algorithm does not adapt along the sequence of scores produced by the forward stagewise additive procedure. Instead of allowing for trees with constant interaction depth at each iteration, it might be more powerful to let the complexity of the newly added tree adapt to the structure remaining to be learned from the data. This is exactly the idea leading to the ABT machine proposed by Huyghe et al. (2022), where trees added to the

score progressively adapt their complexity to the amount of information left to discover from the data. Here, ABT stands for Adaptive Boosting Trees. This new approach also comes with a great added benefit: the stopping criterion is then built inside the ABT algorithm, since the score stops growing when the newly added tree reduces to the root-node one. A small bag fraction can be used to avoid that the ABT machine gets trapped in a sub-optimal solution when this occurs too early (but this seems to be needed only when interaction depth is kept small, which is not required with the ABT machine).

The present paper presents the new BT package in R, contributed by Willame (2022). This package implements boosting trees for Tweedie-distributed responses using log-link function as derived by Hainaut et al. (2022), as well as the adaptive version proposed by Huyghe et al. (2022). It allows actuaries to build predictive models and explore the influence of different features on the response. The BT package in R is now available from CRAN where it can be downloaded through `install.packages('BT')`. See [cran.r-project.org/package=BT](https://cran.r-project.org/package=BT).

The remainder of this paper is organized as follows. Sections 2-4 review the methodology. The new package is described in Section 5. A numerical illustrations is proposed in Section 6 to demonstrate the capabilities of this new tool. It is shown there that BT package is highly competitive for analyzing claim counts.

## 2 Insurance pricing

Consider a response  $Y$  and a set of features  $X_1, \dots, X_p$  gathered in the vector  $\mathbf{X}$ . The dependence structure inside the random vector  $(Y, X_1, \dots, X_p)$  is exploited to extract the information contained in  $\mathbf{X}$  about  $Y$ . In insurance pricing, the aim is to evaluate the pure premium as accurately as possible. This means that the target is the conditional expectation  $\mu(\mathbf{X}) = E[Y|\mathbf{X}]$  of the response  $Y$  (claim number or claim amount) given the available information  $\mathbf{X}$ . Henceforth,  $\mu(\mathbf{X})$  is referred to as the true (pure) premium. Notice that in some applications,  $\mu(\mathbf{X})$  only refers to one component of the pure premium. For instance, working in the frequency-severity decomposition of insurance losses,  $\mu(\mathbf{X})$  can be either the expected number of insured events or the expected claim size, or severity.

The function  $\mathbf{x} \mapsto \mu(\mathbf{x}) = E[Y|\mathbf{X} = \mathbf{x}]$  is unknown to the actuary, and may exhibit a complex behavior in  $\mathbf{x}$ . This is why this function is approximated by a (working, or actual) premium  $\mathbf{x} \mapsto \pi(\mathbf{x})$  with a relatively simple structure compared to the unknown regression function  $\mathbf{x} \mapsto \mu(\mathbf{x})$ .

Let

$$\mathcal{D} = \{(y_1, \mathbf{x}_1, \nu_1), (y_2, \mathbf{x}_2, \nu_2), \dots, (y_n, \mathbf{x}_n, \nu_n)\}, \quad (2.1)$$

be the training set, where  $y_i$  corresponds to the observed response for the  $i$ th record, the vector  $\mathbf{x}_i$  gathers the corresponding features and  $\nu_i$  is a known weight (the exposure to risk for claim frequencies, for instance). The estimates, or fitted values  $\hat{\pi}(\mathbf{x})$  for  $\mu(\mathbf{x})$  are obtained by minimizing the empirical loss on  $\mathcal{D}$ .

	Type	Name
$\xi < 0$	Continuous	-
$\xi = 0$	Continuous	Normal
$0 < \xi < 1$	Non existing	-
$\xi = 1$	Discrete	Poisson
$1 < \xi < 2$	Mixed, non-negative	Compound Poisson-Gamma
$\xi = 2$	Continuous, positive	Gamma
$2 < \xi < 3$	Continuous, positive	-
$\xi = 3$	Continuous, positive	Inverse Gaussian
$\xi > 3$	Continuous, positive	-

Table 3.1: Tweedie distributions.

### 3 Tweedie loss functions

In practice, actuaries often use distributions belonging to the Tweedie family together with the log-link function for modeling responses. The log-link function is generally chosen because of the multiplicative structure it produces for the resulting estimates. In boosting, this link function is retained because it ensures  $\hat{\pi} \geq 0$ . Since Tweedie distributions are closed under scaling, log-link allows the actuary to re-express boosting as an iterative procedure acting on re-scaled and re-weighted responses.

The Tweedie class regroups the members of the Exponential Dispersion family having power variance functions  $V(\mu) = \mu^\xi$  for some  $\xi$ . Specifically, the Tweedie class contains continuous distributions such as the Normal, Gamma and Inverse Gaussian distributions. It also includes the Poisson and compound Poisson-Gamma distributions. Typically, the Poisson distribution is used for modeling claim counts and the Gamma or Inverse Gaussian distributions for claim severities. Compound Poisson-Gamma distributions can be used for modeling annual claim amounts, having positive probability at zero and a continuous distribution on the positive real numbers. We refer the reader to Delong et al. (2021) for a through presentation of Tweedie models with applications to insurance.

Table 3.1 gives a list of all Tweedie distributions. Negative values of  $\xi$  gives continuous distributions on the whole real axis. For  $0 < \xi < 1$ , no member of the Exponential Dispersion family exists. Only the cases  $\xi \geq 1$  are thus interesting for applications to insurance. From e.g. Denuit et al. (2019a, Table 4.7 pp 153), Tweedie deviance loss function is given by

$$L(y, \hat{\pi}(\mathbf{x})) = \begin{cases} (y - \hat{\pi}(\mathbf{x}))^2 & \text{if } \xi = 0, \\ 2 \left( y \ln \frac{y}{\hat{\pi}(\mathbf{x})} - (y - \hat{\pi}(\mathbf{x})) \right) & \text{if } \xi = 1, \\ 2 \left( -\ln \frac{y}{\hat{\pi}(\mathbf{x})} + \frac{y}{\hat{\pi}(\mathbf{x})} - 1 \right) & \text{if } \xi = 2, \\ 2 \left( \frac{\max\{y, 0\}^{2-\xi}}{(1-\xi)(2-\xi)} - \frac{y \hat{\pi}(\mathbf{x})^{1-\xi}}{1-\xi} + \frac{\hat{\pi}(\mathbf{x})^{2-\xi}}{2-\xi} \right) & \text{otherwise.} \end{cases} \quad (3.1)$$

For  $\xi = 0$ , we recover the quadratic loss function whereas  $\xi = 1$  and 2 correspond to the Poisson and Gamma deviance functions, respectively.

## 4 Boosting trees

### 4.1 Forward stagewise additive modeling

Ensemble techniques assume structural models of the form

$$\pi(\mathbf{x}) = g^{-1}(\text{score}(\mathbf{x})) = g^{-1}\left(\sum_{m=1}^M T(\mathbf{x}; \mathbf{a}_m)\right), \quad (4.1)$$

where  $g$  is the link function and  $T(\mathbf{x}; \mathbf{a}_m)$ ,  $m = 1, 2, \dots, M$ , are usually simple functions of the features  $\mathbf{x}$ , characterized by parameters  $\mathbf{a}_m$ . In (4.1), the score is the function of features  $\mathbf{x}$  mapped to  $\pi(\mathbf{x})$  by the inverse of the link function  $g$ , that is,

$$\text{score}(\mathbf{x}) = \sum_{m=1}^M T(\mathbf{x}; \mathbf{a}_m).$$

Consider the training set (2.1). Estimating  $\text{score}(\mathbf{x})$  appearing in (4.1) by minimizing the corresponding training sample estimate of the generalized error

$$\min_{\{\mathbf{a}_m\}_1^M} \sum_{i=1}^n \nu_i L\left(y_i, g^{-1}\left(\sum_{m=1}^M T(\mathbf{x}_i; \mathbf{a}_m)\right)\right) \quad (4.2)$$

is in general infeasible. It requires computationally intensive numerical optimization techniques. One way to overcome this problem is to approximate the solution to (4.2) by using a greedy forward stagewise approach, also called boosting. This consists in sequentially fitting a single function and adding it to the expansion of prior fitted terms. Each fitted term is not readjusted as new terms are added into the expansion, contrarily to a stepwise approach where previous terms are each time readjusted when a new one is added.

To prevent overfitting, cross validation is used to stop the boosting algorithm when its prediction capabilities start to deteriorate. Early stopping plays a central role to ensure a sparse model with optimal performances on new data. The optimal stopping iteration is the one which leads to the smallest average empirical loss on an out-of-sample test data or as measured by cross validation. The latter technique consists in randomly splitting the training data set into several parts (or folds). Each part is then held out of the analysis and the model is fitted on the remaining data to predict the observed values of the response in the part set aside. Cross validation is a convenient way to balance goodness of fit and model complexity: a model too close to the training set will often be worse for predictions, as it reproduces noise in the data (or over-fits the training data).

This leads to the following algorithm:

#### 1. Initialization :

Initialize  $\widehat{\text{score}}_0(\mathbf{x})$  to be a constant by

$$\widehat{\text{score}}_0(\mathbf{x}) = \underset{s}{\operatorname{argmin}} \sum_{i=1}^n \nu_i L(y_i, g^{-1}(s)).$$

#### 2. Main procedure :

**For**  $m = 1$  to  $M$  **do**

a) Compute  $\hat{\mathbf{a}}_m$  by solving the subproblem

$$\hat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\operatorname{argmin}} \sum_{i=1}^n \nu_i L\left(y_i, g^{-1}\left(\widehat{\operatorname{score}}_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \mathbf{a}_m)\right)\right). \quad (4.3)$$

b) Update  $\widehat{\operatorname{score}}_m(\mathbf{x}) = \widehat{\operatorname{score}}_{m-1}(\mathbf{x}) + \beta T(\mathbf{x}; \hat{\mathbf{a}}_m)$   
for some specified shrinkage coefficient  $\beta$ .

**End for**

3. **Output:**

$$\hat{\pi}_{\mathcal{D}}^{\text{boost}}(\mathbf{x}) = g^{-1}\left(\widehat{\operatorname{score}}_{m^*}(\mathbf{x})\right)$$

where  $m^*$  is the optimal iteration selected by cross validation.

Boosting is thus an iterative method based on the idea that combining many simple functions should result in a powerful one. In a boosting context, the simple functions  $T(\mathbf{x}; \mathbf{a}_m)$  are called weak learners or base learners.

## 4.2 Binary regression trees as base learners

In this paper, we use binary regression trees as base learners. This is the case in the majority of applications of boosting to insurance. Trees only use binary features  $\mathbb{I}[X_j < t]$  for ordered features  $X_j$ , with arbitrary threshold  $t$  or  $\mathbb{I}[X_j \in \mathcal{S}]$  for unordered categorical feature  $X_j$  with  $\mathcal{S}$  an arbitrary subset of the levels. Precisely,

- for a quantitative feature  $X_j$ , candidate splits  $t_1, t_2, \dots, t_{m_j}$ , with  $t_1 < t_2 < \dots < t_{m_j}$  are considered. Thresholds  $t_k$  are half-way between consecutive values of  $x_j$  in the database. This leads to binary features  $B_{jk} = \mathbb{I}[X_j < t_k]$ ,  $k = 1, \dots, m_j$ .
- for an ordered categorical feature  $X_j$  with levels  $\ell_1, \ell_2, \dots, \ell_{m_j}$  ranked in ascending order, lower levels are grouped to create

$$B_{jk} = \mathbb{I}[X_j \in \{\ell_1, \dots, \ell_k\}] \text{ for } k = 1, \dots, m_j - 1.$$

- for an unordered categorical feature  $X_j$  with levels  $\ell_1, \ell_2, \dots, \ell_{m_j}$ , all strict subsets of levels are considered to create

$$B_{jk} = \mathbb{I}[X_j \in \mathcal{S}_k] \text{ for every } \mathcal{S}_k \subset \{\ell_1, \dots, \ell_{m_j}\}, \mathcal{S}_k \neq \emptyset.$$

Considering all non-empty subsets  $\mathcal{S}_k \subset \{\ell_1, \dots, \ell_{m_j}\}$  is effective for moderate  $m$ . For larger  $m$ , some approximations are needed to keep computational time within reasonable limits. Often, this amounts to considering the categorical feature as an ordered one through the average response value in each level. Precisely, for an unordered categorical feature  $X_j$  with a larger number  $m$  of levels, it is turned into an ordered categorical feature  $X_j$  by ranking levels  $\ell_1, \ell_2, \dots, \ell_{m_j}$  according to the mean value of the response in each level.



Trees are built sequentially and not optimized globally. Precisely, trees are grown by recursive partitioning. This means that trees recursively partition the feature space in hyper-rectangles. The estimated mean response is then taken as the average response for all data points falling in the hyper-rectangle under consideration. The analyst starts with a large number of candidate binary features  $B_{jk}$  for splitting the data. The score obtained from the preceding iteration is then combined with new candidates  $B_{jk}$  and the binary split causing the largest drop in deviance is integrated in the score.

This produces base learners  $T(\mathbf{x}; \mathbf{a}_m)$  of the form

$$T(\mathbf{x}; \mathbf{a}_m) = \sum_{t \in \mathcal{T}_m} c_{tm} \mathbf{I}[\mathbf{x} \in \chi_t^{(m)}], \quad (4.4)$$

where  $\{\chi_t^{(m)}\}_{t \in \mathcal{T}_m}$  is the partition of the feature space  $\chi$  induced by the regression tree  $T(\mathbf{x}; \mathbf{a}_m)$  and  $\{c_{tm}\}_{t \in \mathcal{T}_m}$  contains the corresponding predictions for the score in each terminal node. For regression trees,  $\mathbf{a}_m$  gathers the splitting variables and their split values as well as the corresponding observed averages in the terminal nodes, that is,

$$\mathbf{a}_m = \{c_{tm}, \chi_t^{(m)}\}_{t \in \mathcal{T}_m}.$$

We refer the reader to Denuit et al. (2020) for a presentation of tree-based methods applied to insurance.

### 4.3 Boosting with Tweedie loss function under log-link

The next result shows that when we work with the log-link function and a response that belongs to the Tweedie class (and so with a loss function of the form (3.1)), solving (4.3) amounts to build a single weak learner on the working training set

$$\mathcal{D}^{(m)} = \{(r_{1,m}, \mathbf{x}_1, \nu_{1,m}), (r_{2,m}, \mathbf{x}_2, \nu_{2,m}), \dots, (r_{n,m}, \mathbf{x}_n, \nu_{n,m})\}$$

replacing the initial training set  $\mathcal{D}$  in (2.1), where adapted weights and ratios are given by

$$\nu_{i,m} = \nu_i \exp(\widehat{\text{score}}_{m-1}(\mathbf{x}_i))^{2-\xi} \text{ and } r_{i,m} = \frac{y_i}{\exp(\widehat{\text{score}}_{m-1}(\mathbf{x}_i))}.$$

We provide an alternative, elementary proof of this result, based on the closure of Tweedie distributions under scaling.

**Proposition 4.1** (Proposition 3.1 in Hainaut et al. (2022)). *Consider the deviance loss function (3.1). Then, (4.3) with the log-link function, that is*

$$\hat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\operatorname{argmin}} \sum_{i=1}^n \nu_i L\left(y_i, \exp\left(\widehat{\text{score}}_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \mathbf{a}_m)\right)\right),$$

can be rewritten as

$$\hat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\operatorname{argmin}} \sum_{i=1}^n \nu_{i,m} L(r_{i,m}, \exp(T(\mathbf{x}_i; \mathbf{a}_m))). \quad (4.5)$$

*Proof.* Recall that if  $Y$  obeys the Tweedie distribution with mean  $\mu$ , power parameter  $\xi$  and weight  $\nu$  then for any positive constant  $c$ ,  $cY$  is Tweedie with mean  $c\mu$ , the same power parameter and modified weight  $\nu(c)^{\xi-2}$ . See e.g. Property 2.5.1 in Denuit et al. (2019a) for a proof. Since the essence of boosting consists in treating  $\widehat{\text{score}}_{m-1}(\mathbf{x}_i)$  as a constant to estimate  $\mathbf{a}_m$ , this means that we can equivalently work with response  $r_{i,m}$  obeying the Tweedie distribution with adapted weight  $\nu_{i,m}$  to perform that estimation. This is a direct application of the result recalled earlier with  $c = 1/\exp(\widehat{\text{score}}_{m-1}(\mathbf{x}_i))$ .  $\square$

Boosting can thus be performed on the responses, proceeding in an iterative way, by dividing the response  $r_{i,m}$  with  $\exp(T(\mathbf{x}_i; \mathbf{a}_m))$  and multiplying  $\nu_{i,m}$  with  $\exp((2-\xi)T(\mathbf{x}_i; \mathbf{a}_m))$  at each step. Proposition 4.1 shows that gradient boosting introduces an extra step which is unnecessary and leads to an approximation that can be easily avoided with boosting. This extends the point made by Wüthrich and Buser (2019) for the Poisson distribution to the whole Tweedie class.

This leads to the following algorithm:

**1. Initialization :**

Initialize  $\widehat{\text{score}}_0(\mathbf{x})$  to be a constant by

$$\widehat{\text{score}}_0(\mathbf{x}) = \underset{s}{\operatorname{argmin}} \sum_{i=1}^n \nu_i L(y_i, g^{-1}(s)).$$

**2. Main procedure :**

**For**  $m = 1$  to  $M$  **do**

a) Calculate weights

$$\nu_{i,m} = \nu_i \exp\left(\widehat{\text{score}}_{m-1}(\mathbf{x}_i)\right)^{2-\xi}$$

and ratios

$$r_{i,m} = \frac{y_i}{\exp(\widehat{\text{score}}_{m-1}(\mathbf{x}_i))}.$$

b) Calculate

$$\begin{aligned} \widehat{\mathbf{a}}_m &= \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^n \nu_i L\left(y_i, \exp\left(\widehat{\text{score}}_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \mathbf{a})\right)\right) \\ &= \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^n \nu_{i,m} L\left(r_{i,m}, \exp\left(T(\mathbf{x}_i; \mathbf{a})\right)\right). \end{aligned}$$

c) Update  $\widehat{\text{score}}_m(\mathbf{x}_i) = \widehat{\text{score}}_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \widehat{\mathbf{a}}_m)$ .

**End for**

**3. Output:**

$$\widehat{\pi}_{\mathcal{D}}^{\text{boost}}(\mathbf{x}) = g^{-1}\left(\widehat{\text{score}}_{m^*}(\mathbf{x})\right)$$

where  $m^*$  is the optimal iteration selected by cross validation.

Updating the weights each time together with the responses appears to be very intuitive for actuaries. Responses  $r_{i,m}$  at iteration  $m$  corresponds to actual over expected ratios, or AE ratios often used in insurance studies. Only the weights differ according to the Tweedie distribution used to model responses. In the Poisson case, weights  $\nu_{i,m}$  are estimated expected responses at the preceding iteration  $m - 1$ . In the Gamma case, weights are left unchanged and  $\nu_{i,m} = \nu_i$  at every iteration  $m$ .

## 4.4 Adaptive boosting

The idea behind ABT for Adaptive Boosting Trees is to fit cost-complexity pruned trees in an adaptive way at each boosting step. In this approach, larger trees are fitted at earlier stages and they progressively simplify until reducing to the single-node tree where the ABT machine stops. The stopping criterion is thus built within the ABT algorithm and no computationally-intensive cross validation step is needed. The cost-complexity measure is a combination of the in-sample deviance and a penalty for the complexity of the tree under interest. The latter is taken to be proportional to the number of terminal nodes (considering that more terminal nodes result in a more flexible, and hence complex model). We refer the reader to Huyghe et al. (2022) for an extensive presentation of this method.

# 5 Getting started with the BT package

## 5.1 Packages related to BT

The BT package implements the boosting approach reviewed in the preceding sections. Each tree in the expansion is built thanks to the `rpart` package. See Therneau and Atkinson (2018) for more information about this R package. The default `tree.control` parameter is set so to grow the biggest tree possible. However, as in other boosting packages, it might happen that some of the built trees do not reach the value of `interaction.depth` specified by users. As we fully rely on `rpart` (we do not have full expansion control), in some specific cases we emphasize that some of the grown trees might not be able to meet the above requirement (i.e. the original `rpart` grown tree contains less than `interaction.depth` internal nodes). Competitors to BT include `gbm3`, `xgboost` and `lightgbm` that are considered in the numerical illustration performed in the next section.

## 5.2 Trees structure in BT

In the (adaptive) boosting tree R-package described in this section, each tree in the expansion is built using the `rpart` R-package. The `rpart` wrapper is perfectly fitted for the adaptive approach. Indeed, at each iteration, the adaptive boosting tree algorithm relies on the minimal cost-complexity pruning procedure widely used within the `rpart` package and amongst users.

Since (A)BT acts as a `rpart` wrapper, some tree building parameters are directly managed by `rpart`. For instance, the size of the trees can be controlled within `rpart` by the parameters `maxdepth` and `minbucket`. Recall that a regression tree with `maxdepth=D` has  $2^D$

terminal nodes, each terminal node having  $D - 1$  ancestors. Note that `rpart` does not allow the user to specify the number of terminal nodes  $J$  nor the interaction depth  $ID$  of the tree.

In our package, the `rpart` parameters `cp` and `xval` are respectively forced to  $-\infty$  and 0, which means that the complexity parameter `cp` is left unspecified and that no cross-validation is performed during the learning process. Notice however that these parameters can be changed by the user if necessary.

Trees with interaction depth  $ID$  correspond to trees with  $J = ID + 1$  terminal nodes such that at most two terminal nodes have  $ID$  ancestors. Therefore, any tree with an interaction depth  $ID$  can be seen as a subtree of a tree with `maxdepth=ID`. That is why, at each iteration, a tree with `maxdepth=ID` is first built with `rpart` without any other constraints. In case  $ID=1$ , the resulting tree corresponds to the selected interaction depth, so that the obtained tree is returned without any other adjustments. Otherwise, when  $ID>1$ ,

- Either the tree contains more than  $ID$  internal nodes. In that case, which is the most likely one, the size of the tree must be reduced in order to get  $ID$  internal nodes.
- Or the tree contains less than  $ID$  internal nodes, which means that the biggest tree that can be built is smaller than expected. In that case, the tree will be returned as such, without any other adjustments.

In the first aforementioned case, a subtree thus still needs to be selected in order to get  $ID$  internal nodes. The selection of this subtree with  $ID$  internal nodes is explained next, in Section 5.3 for the boosting trees procedure and in Section 5.4 for the adaptive version of the boosting trees procedure.

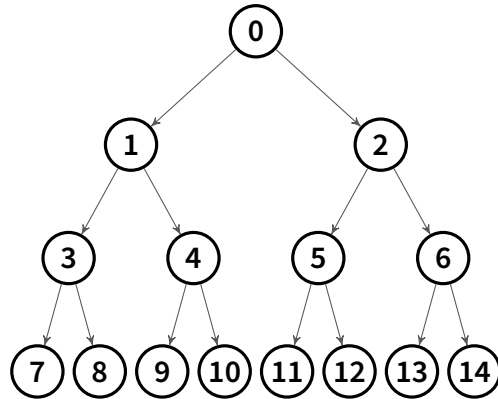
Notice that the tree originally developed is bigger than needed, increasing the computation time. However, it is worth to mention that the tree process is fully handled by external well-developed (and well-known) package, leading to full transparency and easier handling.

### 5.3 Boosting trees

Thanks to `rpart` outputs, one can implement a greedy strategy starting from the root node of the tree, as illustrated in Figure 5.1 for  $ID=3$ . The initial `rpart` object is thus a tree with `maxdepth=ID>1`. The root node 0 is first split into two children nodes 1 and 2, so that the subtree after one split has terminal nodes 1 and 2 and hence  $ID=1$ . Then, node 1 is in turn split into two children nodes 3 and 4 and node 2 is split into nodes 5 and 6. Among those two splits, we select the one that maximizes the decrease of the Poisson deviance, say node 2 as in Figure 5.1, so that the subtree after two splits has terminal nodes 1, 5 and 6 and  $ID=2$ . Again, nodes 1, 5 and 6 are in turn split into children nodes 3 and 4 for node 1, 11 and 12 for node 5 and 13 and 14 for node 6. The third split is then selected among those three splits as the one that maximizes the decrease of the Poisson deviance, say node 5 as in Figure 5.1. The resulting tree has thus now an interaction depth  $ID=3$  with internal nodes 0, 2 and 5. This process is continued until we get the selected  $ID$ . In the example of Figure 5.1, we stop the procedure here since we achieved the selected  $ID$ .

Notice that all this process is made possible thanks to the `snip.rpart` function in `rpart`. From an initially built tree, this allows the user to create a subtree by specifying the branches to be trimmed out. In addition, we emphasize that this function can also be interactively used by clicking on particular nodes within the graphics window.

Initial rpart object



Boosting Tree: Create tree from rpart output

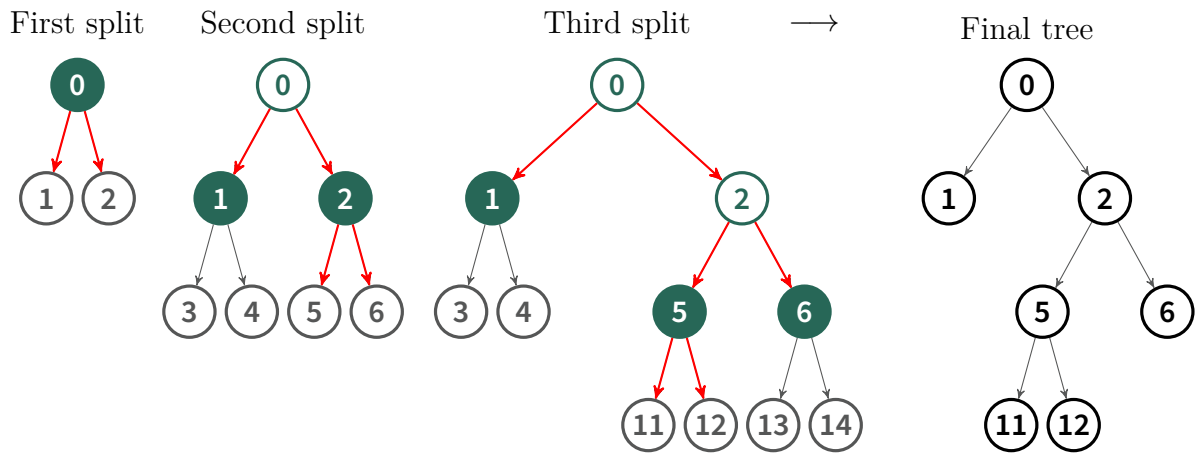


Figure 5.1: Illustration of the construction of the BT tree for ID=3 based on rpart output. At each step, the splitting candidates are highlighted in green and the chosen strategy (best improvement by splitting this node) is highlighted in red.

## 5.4 Adaptive boosting trees

The `rpart` outputs enable us to get the sequence of subtrees corresponding to the minimal cost-complexity pruning procedure recalled in Huyghe et al. (2022) thanks to the `cptable` object. Then, we select among these subtrees the largest one with at most `ID` internal nodes, i.e. the largest subtree satisfying the inequality  $J \leq \text{ID} + 1$ . Compared to boosting trees, the obtained subtree can have this time less than `ID` internal nodes as explained in Huyghe et al. (2022). Moreover, it is interesting to notice that even if the selected tree has exactly `ID` internal nodes, it can be different from the one that would have been obtained with the boosting trees algorithm. Indeed, while the boosting trees procedure uses a greedy strategy, the adaptive boosting trees approach improves this greedy strategy by assessing the goodness of the splits by also looking at those deeper in the tree.

## 5.5 Main arguments of BT

The user needs to define the following parameters:

**formula:** a symbolic description of the model to be fit, similar to what is required by other R functions performing regression analyses. We emphasize that offsets are not supported but that weights are used instead, in line with the approach presented in the preceding section.

**data:** the database on which the computations will be performed.

**tweedie.power:** Tweedie power parameter for the response under consideration.

**ABT:** boolean value to define whether we fit a Boosting Tree (=FALSE) or an Adaptive Boosting Tree (=TRUE).

**n.iter:** number of iterations.

**train.fraction:** percentage of the data used as training set. The remaining part will be used as validation set.

**interaction.depth:** maximum number of splits in each tree.

**shrinkage:** acts as regularization for additional iterations - the smaller the shrinkage generally the better the performance of the fit. However, smaller shrinkage implies that the number of trees may need to be increased to achieve a certain performance.

**bag.fraction:** fraction of the training observations randomly sub-sampled to fit a tree in each iteration. This has the effect of reducing the variance of the boosted model.

**colsample.bytree:** number of variables randomly sampled that will be used to build the next tree in the expansion.

**cv.folds:** number of cross-validation folds to create. If set to 1 (by default), no cross-validation is performed.

**n.cores:** in case of cross-validation, the number of cores used to perform the parallelization. Please note that in the cross-validation context, a call to the `parLapply` function is made (whatever the number of cores). This parameter is originally set to `cv.folds`.

**tree.control:** the proposed algorithm is based on the `rpart` package. This parameter will be used to originally build each tree in the expansion. We emphasize that if `interaction.depth` is set to `NULL`, each tree in the expansion will be built thanks to this parameters with no further treatment. We recommend this parameter usage for advanced user only.

**weights:** a vector representing the weight given to each observation. By default, the same weight (=1) is assigned to each observation.

**seed:** some of the parameters bring randomness during the algorithm. This parameter allows the user to replicate the results.

We emphasize that performing cross-validation produces a first global model trained on the full training set as well as different `cv` related BT models. The former is generally further used while the latter helps to assess performances, for instance.

The `BT_perf` function allows the user to determine the best number of iterations that has to be performed. This one also depends on the type of errors that are available/have been computed during training phase. The `training.error` is automatically computed. In case bagging is used, this corresponds to the in-bag errors (i.e. a random sub-selection of the original training set). Also,

- If a `train.fraction` has properly been defined, a `validation.error` will be computed on the validation set.
- If a `bag.fraction` has properly been defined, an `oob.improvement` vector will be computed.
- If cross-validation parameters have been filled, a `cv.error` will be computed.

These values are stored in the `BTErrors` object.

The optimal number of iterations can be selected in a number of ways, by specifying method which can be set to `validation`, `OOB` or `cv` depending on whether the user wants to use `validation.error`, `oob.improvement` or `cv.error` as previously explained. We emphasize that without specifying the `method` argument a best-guess approach will be performed. If desired, the `BT_perf` function plots the computed errors alongside returning the optimal iteration.

This sections only summarizes the main features of the package. An extensive presentation of the BT package can be found in Willame (2022).

## 6 Numerical illustration

Now that the BT package has been presented, let us perform a numerical illustration to assess its performances on a typical insurance data set. After having presented the database, we

compare the results obtained with the help of the BT package to competitors `gbm3`, `xgboost` and `lightgbm`. All these computations have been performed using R.

## 6.1 Descriptive statistics

The numerical illustration uses the `freMTPLfreq` database contained in the `CASDatasets` package contributed by Dutang and Charpentier (2020). It comprises features collected for 413,169 motor third-party liability insurance policies together with the number of claims reported to the insurer. This database has often been used to illustrate insurance pricing methods.

The considered data set contains the following information:

**PolicyID:** The policy ID (used to link with the claims dataset).

**ClaimNb:** Number of claims during the exposure period.

**Exposure:** The period of exposure for a policy, in years.

**Power:** The power of the car (ordered categorical).

**CarAge:** The vehicle age, in years.

**DriverAge:** The driver age, in years (in France, people can drive a car at 18).

**Brand:** The car brand divided in the following groups:

- Renault Nissan or Citroen.
- Volkswagen, Audi, Skoda or Seat.
- Opel, General Motors or Ford.
- Fiat.
- Mercedes, Chrysler or BMW.
- Japanese (except Nissan) or Korean.
- Other.

**Gas:** The car gas, Diesel or Regular.

**Region:** The policy region in France (based on the 1970-2015 classification).

**Density:** The density of inhabitants (number of inhabitants per km<sup>2</sup>) in the city where the policyholder lives.

The claim frequency defined as  $\text{ClaimFreq} = \frac{\text{ClaimNb}}{\text{Exposure}}$  has been computed and added to the database. Before running the BT package, some records have been dropped: records with an exposure larger than 1 year have been deleted, only drivers below the age of 90 are considered, and only cars with vehicle age below 30 years are considered. Applying these choices, we ended up with 411,305 records which represents 99.55% of the original number of records.



The working database's total exposure and claim number respectively amount to 230,319 years and 16,128 with a claim frequency of around 7%. Descriptive statistics are displayed in Figure 6.1. Notice that the barplot represents the total exposure in the considered segment and should be read on the left axis. The curve represents the claims frequency and its value can be read on the right axis. Some observations can be drawn from these plots:

**CarAge:** As already mentioned, majority of cars have less than 30 years. The exposure above 20 years is quite limited. Regarding the claims frequency, it seems relatively stable up to 12 years. A slight decrease can however be seen from this point on but should be tempered by the low exposure observed above 20 years old.

**DriverAge:** Drivers are mainly aged between 30 and 55 years old. A clear trend is visible in the claims frequency where younger drivers appear to be the most dangerous ones. This observation can also be made for older drivers but has to be balanced by the low exposure in that category.

**Brand:** Almost all the cars in the database are either Renault, Nissan or Citroen. This fact can easily be explained as the database gathers information from French insurers. On the claims frequency side, no specific effect is visible.

**Power:** As this variable is ordered categorical, one can interpret the categories higher than "k" as the most powerful cars. We can observe that this specific segment is not the most represented in the database. Moreover, the claims frequency tends to increase with the car power.

**Density:** Largest exposures correspond to small density values, showing that a large part of the database is coming from rural areas or small cities. One can also underline a clear increasing trend in the claim frequency.

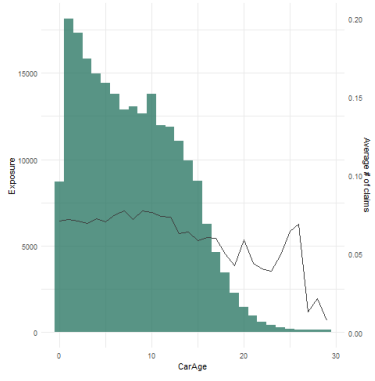
**Region:** Most of the policyholders are coming from the "Centre" region, the most populated area in France. It is followed by the "Ile de France" and "Bretagne" ones. The largest claims frequency is observed for the "Ile de France" region, corresponding to Paris area.

**Gas:** The database is well balanced between Diesel and Regular car gas. A slightly better claim frequency can be noted for the Regular car gas.

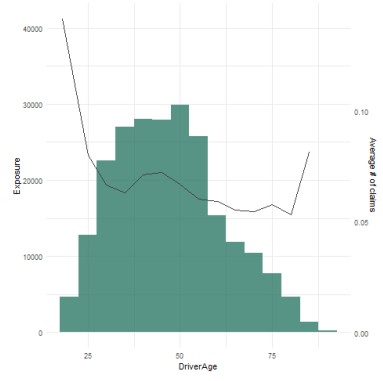
Now that the database has been thoroughly presented, let us continue with model fitting. To this end, we split the database in three parts:

- a *full training set* which contains 80% of the randomly sampled records, further split into:
  - a *training set* containing the 80% first *full training set*'s records.
  - a *validation set* containing the 20% last records.
- a *testing set* which contains the remaining 20% records.

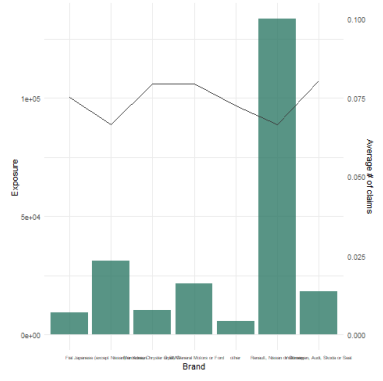
The testing set is reserved to model comparison. The claims frequency observed in these 3 sets are respectively of 6.99%, 7.11% and 6.97%.



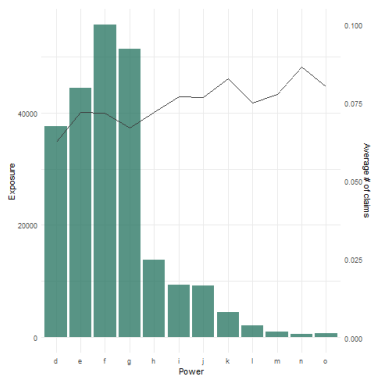
(a) CarAge



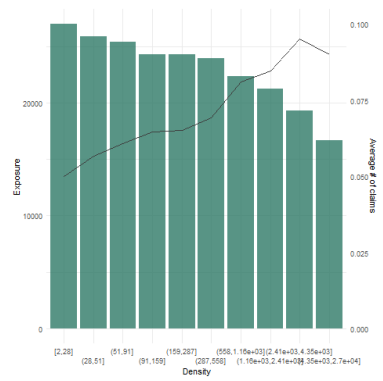
(b) DriverAge



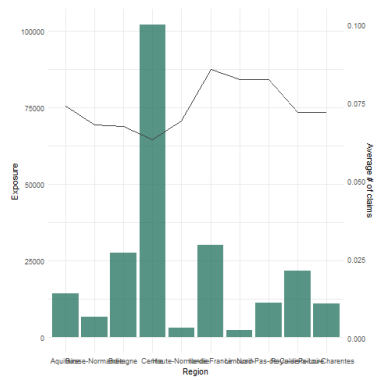
(c) Brand



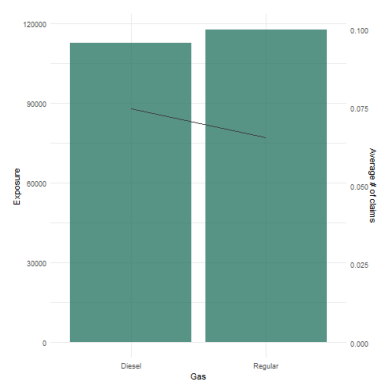
(d) Power



(e) Density



(f) Region



(g) Gas

Figure 6.1: Univariate descriptive analysis

## 6.2 Comparison

We now aim to compare the performances of competitor models, namely BT with and without adaptive approach, `gbm3`, `xgboost` and `lightgbm`. To this end, a grid search has been applied for each competitor model. A set of possible parameters combination (so-called grid precisely defined below) has therefore been created. In each case, every competitor model has been fitted and its performances have been assessed using either validation set or cross-validation approach. Moreover, the test set performances have also been computed for each fitted model using the optimal number of iterations obtained with the two approaches. Notice that the same predefined grid has been used for all competitors. In other words, we looked for the best models inside this grid of parameters only; all the other models' specific parameters were therefore left to their default values. We acknowledge that allowing for a deeper and/or competitor specific grid of parameters might have led to better models and to different models comparison's conclusion. The following grid of parameters has been used for the present application:

- The interaction depth varies between 1 and 7. When possible, this parameter has been interpreted as in BT package, that is the number of non-terminal nodes. In particular,
  - the `lightgbm` package does not have such parameter but we rather used the number of terminal leaves.
  - in the same way, the `xgboost` package had to be tweaked to match with other approaches. We therefore used the `approx` method to build the tree along with the `maximum leaves` parameter.
- A bagging fraction of 50% has been applied with a bagging frequency of 1.
- A shrinkage of 1% has been used across all models.
- The minimal number of observations in a node has been set to 2.
- When cross-validation was performed, 3 folds were used.
- For all fitted model, 5,000 boosting iterations have been performed.

For each competitor, 7 combinations of parameters have been tried out and multiple performances have been assessed. As one can notice, these possibilities only differ in the interaction depth parameter. In the following, we refer to "run  $x$ " to mean that the parameters set corresponds to the interaction depth  $x$ . In particular, the 4<sup>th</sup> BT fitted model can be reproduced thanks to the following pseudo code (its Adaptive Boosting counterpart is easily obtained by setting `ABT=TRUE`):

```
BT(  
  formula = as.formula("ClaimFreq ~  
    Power + CarAge + DriverAge + Brand + Gas + Region + Density"  
  ),  
  data = full training set,  
  tweedie.power = 1,  
)
```

```

ABT = FALSE,
weights = Exposure,
keep.data = FALSE,
cv.folds = 3,
train.fraction = 0.8,
n.iter = 5000,
interaction.depth = 4,
shrinkage = 0.01,
bag.fraction = 0.5
)

```

Before diving into the results, the following technical aspects have to be stressed:

- As already mentioned, the BT package does not support offset. However, in our specific framework considering offset is equivalent to normalizing the response variable by the corresponding weight value. Thus, it corresponds to working with ClaimFreq as response variable and weights equal to Exposure. This approach has also been retained for `xgboost` as well as `lightgbm`. In the `gbm3` case, usual offsetting has been applied.
- For this example, the idea was to compute validation set performances as well as cross-validation ones. We however emphasize that the former is not needed once the latter is used. Moreover, it reduces the training set size to 64% of the working database rather than 80% (i.e. no split of the *full training set*). That being said, the size of the training set is deemed acceptable to provide meaningful results.

Depending on the package, either the *full training set* with a training fraction of 80% (meaning that the first 80% records are used as train set and the remaining as validation set) or the previously defined sets have been used. In any case, every model fitting was performed so that the underlying sets remain similar.

- As described in the previous section, Adaptive Boosting does not necessarily need to perform cross-validation. This conclusion is rather interesting to limit the computational cost. To keep everything comparable, the single-root convergence has not been used as stopping criterion but we applied the same cross-validation process for all the competing models.
- We emphasize that the `xgboost` package required one-hot encoding for categorical variables. This approach has also been tested for the `lightgbm` package. In the latter case, the retained option was however to use sparse matrix and point out the categorical variables to the algorithm.
- Model performances were assessed thanks to Poisson deviance. Nonetheless, the Poisson loss function used across the different competitor packages are not fully identical. For example, it often uses the negative Poisson log-likelihood. To make graphs comparable, we therefore needed to re-scale the furnished outputs to obtain the Poisson deviance <sup>1</sup>. We also mention that the represented total set deviance is obtained as the

---

<sup>1</sup>In the `lightgbm` and `xgboost` cross-validation plots, it corresponds to a small proxy which supposes that the total exposure within each fold is equal.

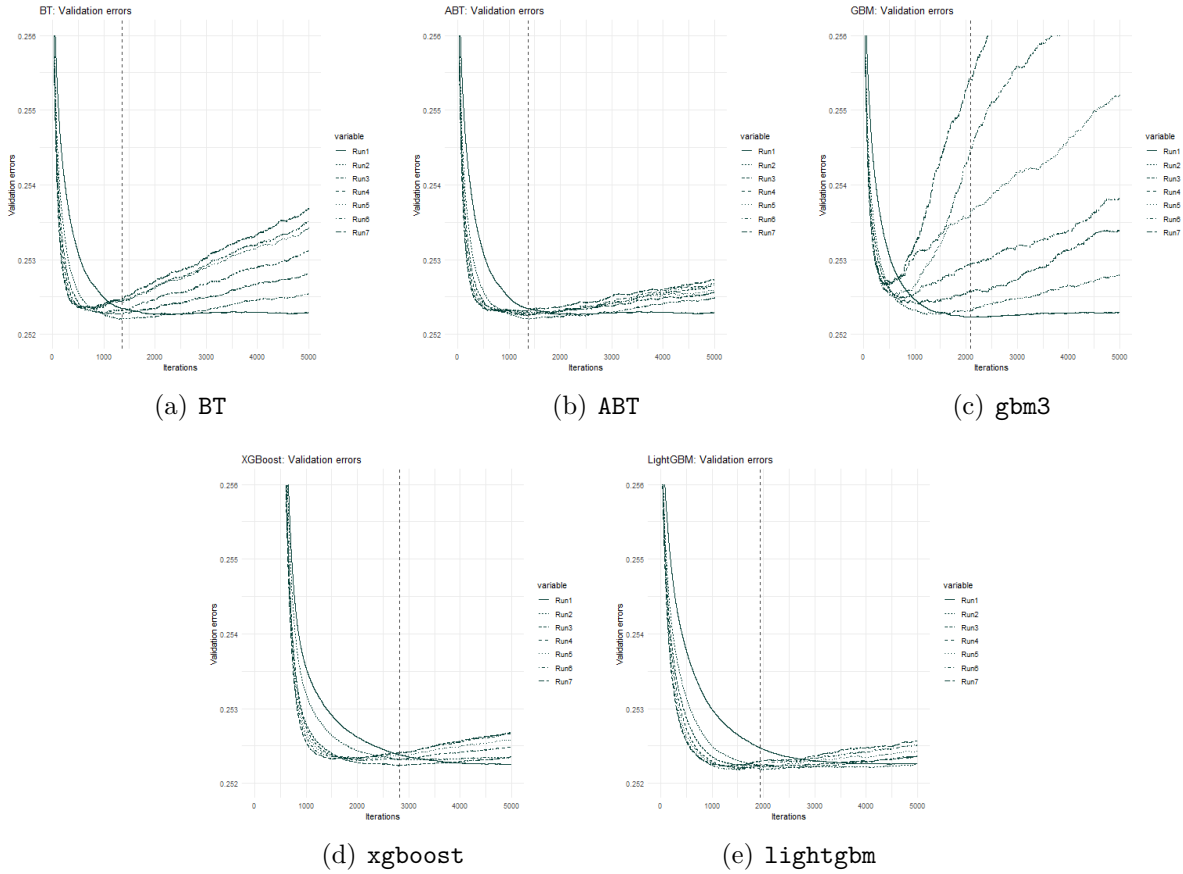


Figure 6.2: Validation set deviance

sum of the individual deviances divided by the number of records within the considered set.

On Figure 6.2 (resp. Figure 6.3) the validation set errors (resp. cross-validation errors) are shown for each competitor model across the 7 parameters combination. The vertical dotted line represents the iteration which minimizes the error across all the 7 runs. For clarity, the minimal validation set and cross-validation errors are also summarized on Figure 6.4. As one can note, the performances shown by the Adaptive Boosting and classical Boosting tend to outperform its competitors for the cross-validation assessment. The results are obviously a bit more volatile using the validation errors.

Using the validation set (resp. cross-validation) criterion, the optimal number of iterations is obtained for each fitted model. Thanks to this information, the test set predictions as well as the total set deviance are computed for every model under consideration. These deviances are displayed in Figure 6.5. This assessment tends to show that the Adaptive Boosting is globally better generalizing than its boosting competitors. That being said, if one solely sticks with the best models obtained via cross-validation i.e. 5<sup>th</sup> run for ABT, 7<sup>th</sup> run for BT, 2<sup>nd</sup> run for xgboost, ... the lightgbm model is slightly better. Using the validation criterion, the xgboost model seems a bit better. We finally note that the three competitors generalization performances are basically almost equivalent. It furthermore demonstrates

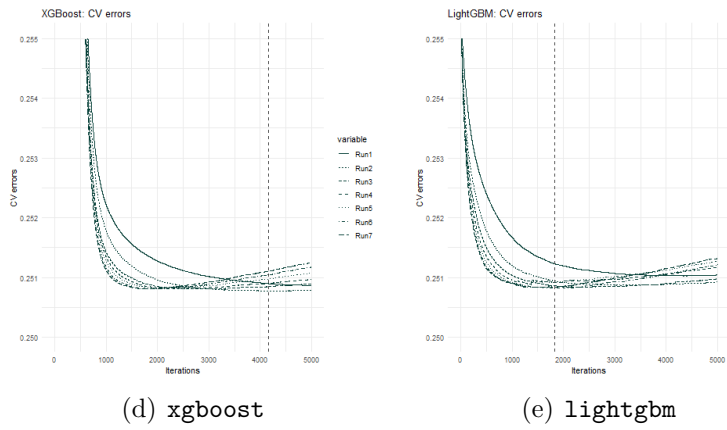
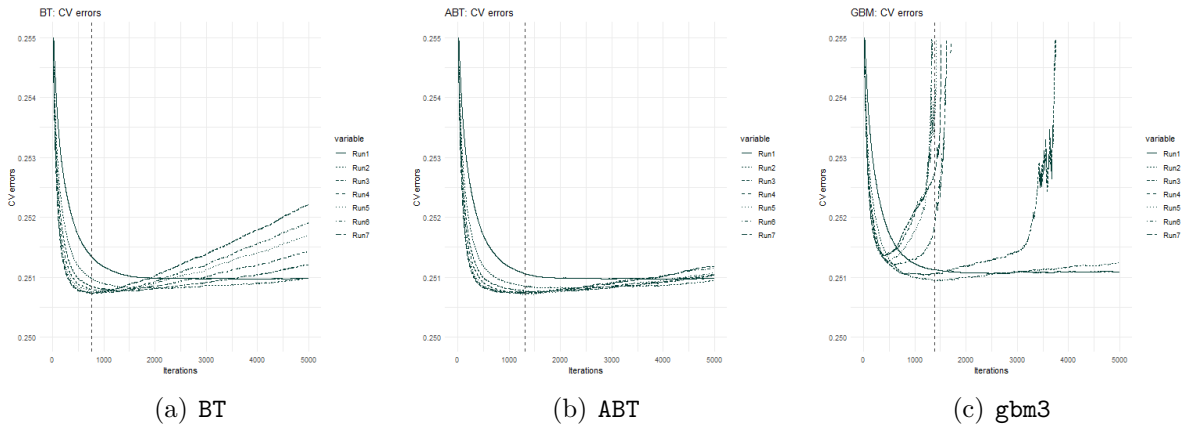


Figure 6.3: Cross-validation deviance

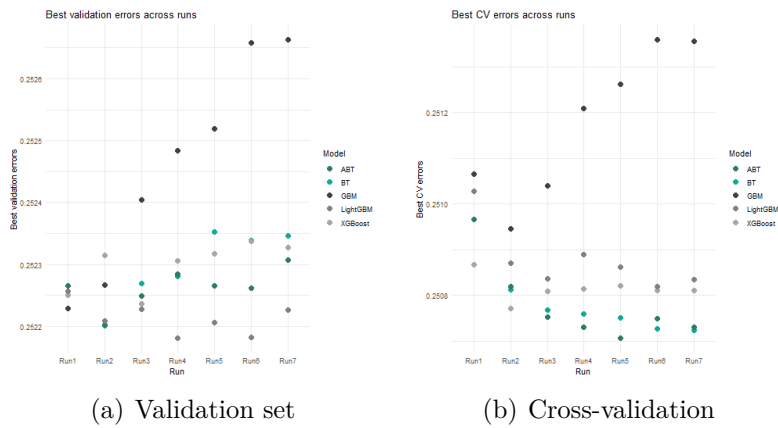


Figure 6.4: Minimal deviance for each run

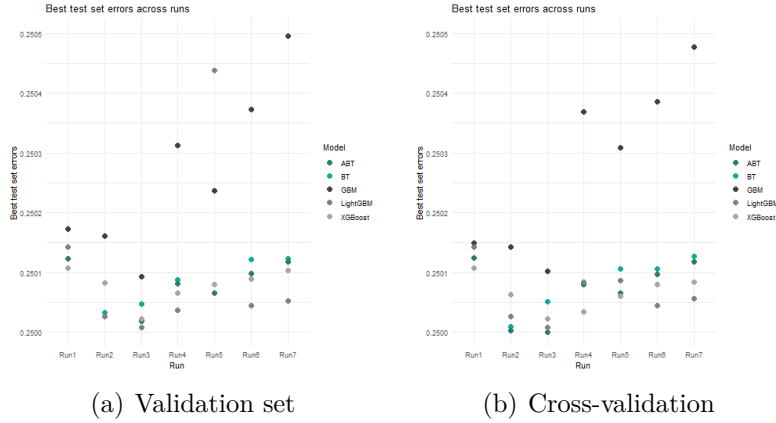


Figure 6.5: Minimal test set deviance for each run

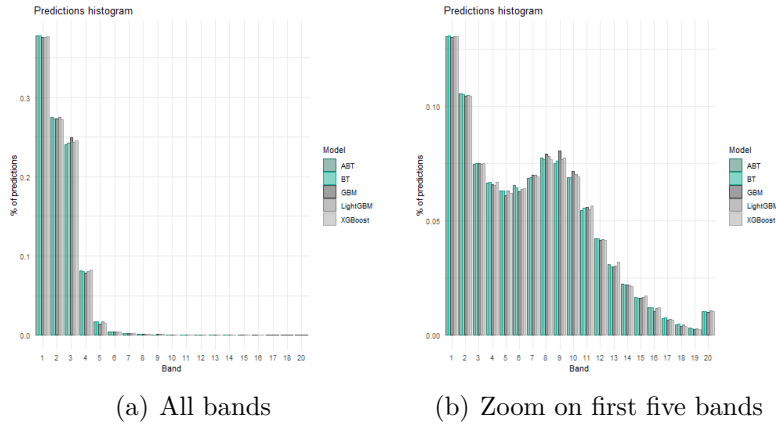


Figure 6.6: Predictions' histogram - Min. predicted value =  $7.52 \times 10^{-5}$ , band length = 0.025 for the left graph (0.006 on the right graph)

the relevance of BT package for Poisson distributed response variable.

While deviance is one of the most used criteria, it might be interesting to look at other performance measures. For clarity purpose, only the test set predictions obtained with the optimal number of iterations in the cross-validation case are considered. Sometimes, only the results for the best run are displayed to avoid to multiply results.

The left-hand side of Figure 6.6 displays histograms for predictions according to each best model. Please note that the twenty bins have been obtained by uniformly dividing the intervals containing all the predictions. As the predictions' are concentrated on the lower tail, a zoom on the first five bins is performed on the right-hand side of Figure 6.6 where these five initial bins are divided into twenty equidistant bins. It clearly seems that the predictions' distributions are similar for all models.

Following Tevet (2013), simple lift charts are represented in Figure 6.7. These plots are easily obtained via the following process:

1. Compute the model predictions on the test set.

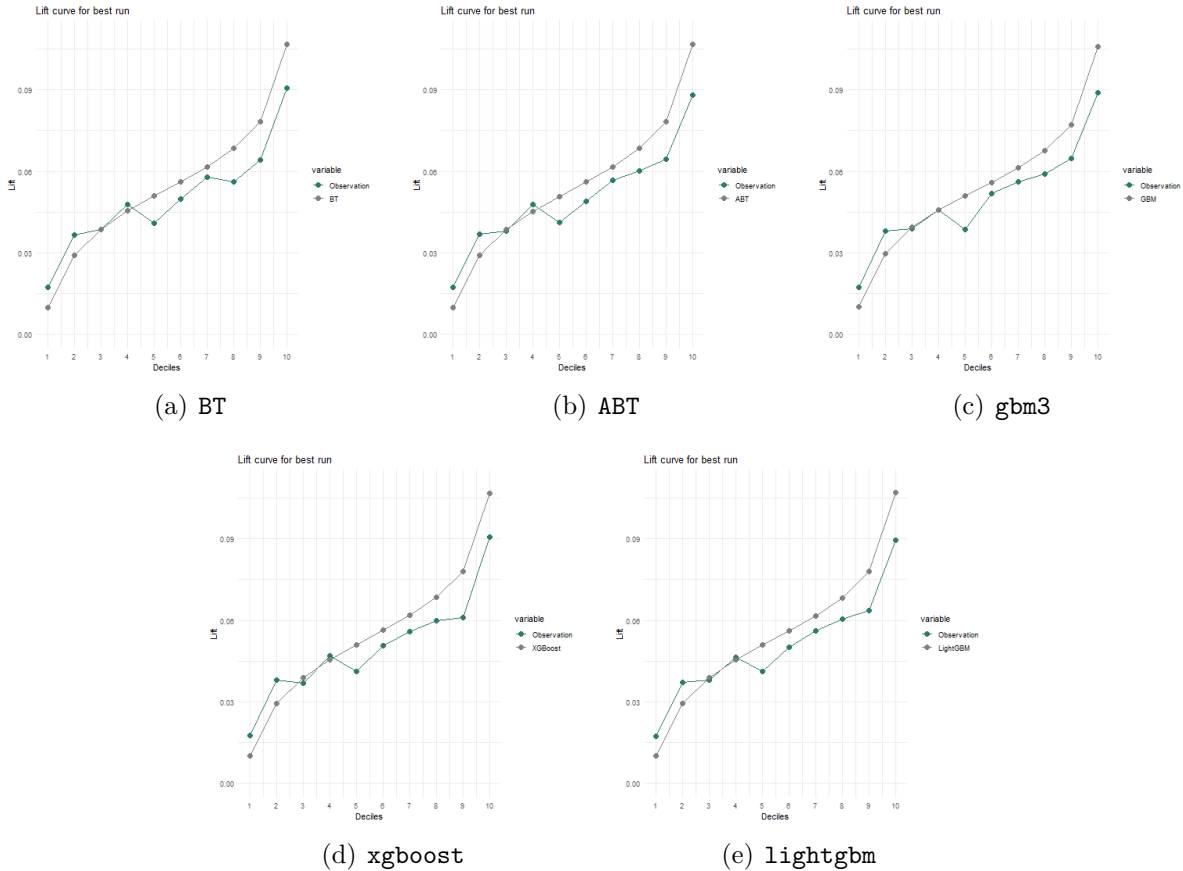


Figure 6.7: Simple lift chart for best CV-runs

2. Sort the test set according to the obtained predictions (from lowest to highest).
3. Split the ranked observations into 10 buckets so that each band has the same exposure.
4. For each bucket, compute the average of observed ClaimNb and average of predicted values.

The results are globally similar. The models over-estimate the highest frequencies and under-estimate the lowest ones, with a breaking point around the 4<sup>th</sup> decile. Notice that this may suggest the need for an additional auto-calibration step. We refer the interested reader to Ciatto et al. (2022).

The Concentration Curve and the Lorenz Curve have also been investigated. More precisely, we focused on two resulting metrics, namely the Area Between the Curves (ABC) and the Integrated Concentration Curve (ICC). The former is defined as the area between the two performance curves while the latter is defined as the area under the concentration curve. According to Denuit et al. (2019b), the Lorenz Curve at a given level  $\alpha$  represents the share of predicted claims corresponding to the  $\alpha\%$  of policies from the portfolio with the lowest predicted values. The Concentration Curve at this level gives the corresponding share of the true claims that should have been predicted from this sub-portfolio. The ABC and ICC metrics can therefore be interpreted as follows:



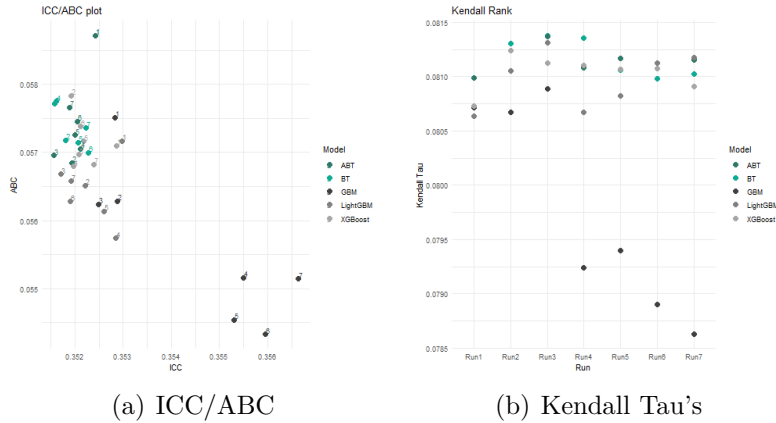


Figure 6.8: Rank performances

- A large difference between the two curves suggests that the considered predictor poorly approximates the observation. We therefore want to minimize the ABC metric which is the area contained between the Concentration Curve and the Lorenz Curve.
- The more convex the Concentration Curve, the better. In fact, a convex curve will result in a better classification (lower risk will be less charged compared to higher ones). Due to its properties this is equivalent to minimize the ICC.

One can observe on Figure 6.8 that there is no model jointly minimizing the two metrics. In fact, the BT package appears to be better in terms of ICC while `gbm3` has a lower ABC. If we ignore the latter package which previously showed the worst performances, the BT models along with `lightgbm` tend to be a good trade-off. In particular, the third Adaptive Boosting run appears to be an appropriate candidate. Kendall's rank correlation coefficients have also been computed. This well-known metric allows the user to measure the concordance between observations and predictions. Obviously, this assessment is closely related to previous metrics. The results are displayed on the right-hand side of Figure 6.8. This computation seems to heavily favor the BT approach.

We finally end up this comparison by discussing the so-called variable importance. To that end, the one-hot encoded importance have been summed up for the `xgboost` package. The results are displayed in Table 6.2. Importances are globally similar across the different competitors. Moreover, it clearly seems that the Density and the DriverAge are way more important compared to the other ones. This observation is aligned with the descriptive analysis where clear trends were observed for these features. One can also underline the fact that these two variables are continuous and that the tree-based approach typically favors such features.

## 7 Discussion

As shown in the numerical illustration, the performances obtained with the help of the BT package are among the most powerful ones for all considered metrics. Of course, these

Variable	BT	ABT	gbm3	xgboost	lightgbm
Brand	6	7	6	6	7
CarAge	8	8	5	9	9
Density	25	24	22	28	25
DriverAge	41	39	39	41	40
Gas	4	4	3	4	4
Power	11	12	14	8	10
Region	6	7	11	4	5

Table 6.1: Variable importance (in %).

findings relate to a specific database and different results could be obtained with other ones. Also, creating different testing set and/or using other seeds across runs might have led to different ranking. We refer to Denuit et al. (2020) for a thorough explanation. We acknowledge that the numerical illustration has been performed with a rather limited grid of parameters and that the BT computational time was the longest one.

The BT package is still under development. In its current version, NA values are automatically dropped from the input database. In the future, we could rely on the NA treatment given by the `rpart` package which is based on surrogate variables. Finally, the convergence to a root node seems to be the natural stopping criterion for the Adaptive Boosting and this approach still has to be implemented.

## References

- Ciatto, N., Denuit, M., Trufin, J., Verelst, H. (2022). Does autocalibration improve goodness of fit ? *European Actuarial Journal*.
- DeLong, L., Lindholm, M., Wüthrich, M. V. (2021). Making Tweedie’s compound Poisson model more accessible. *European Actuarial Journal* 11, 185-226.
- Denuit, M., Hainaut, D., Trufin, J. (2019a). *Effective Statistical Learning Methods for Actuaries I: GLM and Extensions*. Springer Actuarial Lecture Notes Series.
- Denuit, M., Hainaut, D., Trufin, J. (2020). *Effective Statistical Learning Methods for Actuaries II: Tree-based Methods and Extensions*. Springer Actuarial Lecture Notes Series.
- Denuit, M., Sznajder, D., Trufin, J. (2019b). Model selection based on Lorenz and concentration curves, gini indices and convex order. *Insurance: Mathematics and Economics*, 89:128-139.
- Dutang, C., Charpentier, A. (2020). Package `CASdatasets`.
- Hainaut, D., Trufin, J., Denuit, M. (2022). Response versus gradient boosting trees, GLMs and neural networks under Tweedie loss and log-link. *Scandinavian Actuarial Journal* 2022, 841-866.

- Huyghe, J., Trufin, J., Denuit, M. (2022). Boosting cost-complexity pruned trees on Tweedie responses: The ABT machine for insurance ratemaking.
- Lee, S.C., Lin, S. (2018). Delta boosting machine with application to general insurance. North American Actuarial Journal 22, 405-425.
- Therneau, T. M., Atkinson, B. (2018). rpart: Recursive partitioning and regression trees. <https://cran.r-project.org/package=rpart>.
- Tevert, D (2013). Exploring model lift: Is your model worth implementing. Actuarial Review 40, 10-13.
- Willame, G. (2022). BT: (Adaptive) Boosting Trees Algorithm. <https://cran.r-project.org/package=BT> and <https://github.com/GiregWillame/BT>.
- Wüthrich, M. V., Buser, C. (2019). Data Analytics for Non-Life Insurance Pricing. Lecture notes available at SSRN <http://dx.doi.org/10.2139/ssrn.2870308>.



People drive actuarial innovation