Annals of Actuarial Science (2024), 1–21
doi:10.1017/S174849952300026X

ACTUARIAL SOFTWARE

Boosted Poisson regression trees: a guide to the BT package in R

Gireg Willame[1], Julien Trufin[2] and Michel Denuit[3]

[1]Actuarial Expert Consultant, Detralytics, Brussels, Belgium; [2]Department of Mathematics, Université Libre de Bruxelles (ULB), Brussels, Belgium; and [3]Institute of Statistics, Biostatistics and Actuarial Science, UCLouvain, Louvain-la-Neuve, Belgium
Corresponding author: Gireg Willame; Email: g.willame@detralytics.eu

(Received 20 February 2023; revised 04 November 2023; accepted 06 November 2023)

Abstract
Thanks to its outstanding performances, boosting has rapidly gained wide acceptance among actuaries. Wüthrich and Buser (Data Analytics for Non-Life Insurance Pricing. Lecture notes available at SSRN. http://dx.doi.org/10.2139/ssrn.2870308, 2019) established that boosting can be conducted directly on the response under Poisson deviance 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. (Boosting cost-complexity pruned trees on Tweedie responses: The ABT machine for insurance ratemaking. Scandinavian Actuarial Journal. https://doi.org/10.1080/03461238.2023.2258135, 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 (Boosting Trees Algorithm. https://cran.r-project.org/package=BT; https://github.com/GiregWillame/BT, 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 weak 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 weak 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 since its introduction in Friedman (2001), and several open-source packages 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. Let us just mention a few of them, to demonstrate the wide applicability of this approach in this context. Guelman (2012) applied gradient boosted trees to predict motor insurance losses. Liu et al. (2014) considered claim frequencies, using multiclass AdaBoost trees. Yang et al. (2018) adapted gradient boosted tree algorithm to Tweedie models. Pesantez-Narvaez et al. (2019) employed XGBoost to predict the occurrence of claims using telematics data. Henckaerts et al. (2021) worked with random forests and boosted trees to develop full tariff plans built from both the frequency and severity of claims.

However, gradient boosting beyond the normal case typically faces the same problem that leads to the adoption of GLMs for insurance applications. In Poisson regression, 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. 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.

Wüthrich and Buser (2019) established that likelihood-based boosting can easily be achieved when trees are used as weak learners under Poisson deviance loss with log-link, provided the responses are reweighted and rescaled at each boosting step. Hence, responses can be used directly, and there is no need to replace them with gradients as long as Poisson deviance 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 suboptimal 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 current paper presents the new `BT` package in `R`, contributed by Willame (2022). This package implements boosting trees for Poisson-distributed responses using log-link function, 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`.

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 illustration 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 $X_1, \ldots, X_p$ gathered in the vector $\boldsymbol{X}$. In this paper, Y is the number of claims reported by a policyholder from the portfolio. The dependence structure inside the random vector $(Y, X_1, \ldots, X_p)$ is exploited to extract the information contained in $\boldsymbol{X}$ about Y. In insurance pricing, the aim is to evaluate the pure premium as accurately as possible.

Here, the target is the conditional expectation $\mu(\boldsymbol{X}) = \mathrm{E}[Y|\boldsymbol{X}]$ of the number of claims Y given the available information $\boldsymbol{X}$, so that $\mu(\boldsymbol{X})$ only refers to one component of the pure premium. In fact, working in the frequency-severity decomposition of insurance losses, $\mu(\boldsymbol{X})$ is the expected number of insured events to be multiplied by the expected claim size, or severity to get the pure premium.

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

Let

$$\mathcal{D} = \{(y_1, \boldsymbol{x}_1, \nu_1), (y_2, \boldsymbol{x}_2, \nu_2), \ldots, (y_n, \boldsymbol{x}_n, \nu_n)\}, \tag{2.1}$$

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

3. Poisson deviance loss functions

In practice, actuaries often use the Poisson distribution together with the log-link function for modeling claim counts. 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 $\widehat{\pi} \geq 0$. This also allows the actuary to re-express boosting as an iterative procedure acting on re-scaled and reweighted responses. The Poisson deviance loss function is given by

$$L(y, \widehat{\pi}(\boldsymbol{x})) = 2\left(y \ln \frac{y}{\widehat{\pi}(\boldsymbol{x})} - (y - \widehat{\pi}(\boldsymbol{x}))\right). \tag{3.1}$$

4. Boosting trees

4.1 *Forward stagewise additive modeling*

Ensemble techniques assume structural models of the form

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

where g is the link function and $T(\boldsymbol{x}; \mathbf{a}_m)$, $m = 1, 2, \ldots, M$, are usually simple functions of the features $\boldsymbol{x}$, typically referred to as weak learners (see, e.g., Friedman, 2001), characterized by parameters $\mathbf{a}_m$. In (4.1), the score is the function of features $\boldsymbol{x}$ mapped to $\pi(\boldsymbol{x})$ by the inverse of the link function g, that is,

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

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

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

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 weak learner 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 that 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 overfits the training data). We refer the reader to Hastie et al. (2008) for more information on cross validation.

This leads to the following algorithm:

```
1. Initialization:
        Initialize $\widehat{\text{score}}_0(\boldsymbol{x})$ to be a constant by
            $$\widehat{\text{score}}_0(\boldsymbol{x}) = \underset{s}{\text{argmin}} \sum_{i=1}^{n} \nu_i L(y_i, g^{-1}(s)).$$
2. Main procedure:
        For $m = 1$ to $M$ do
            a) Compute $\widehat{\mathbf{a}}_m$ by solving the subproblem
                $$\widehat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\text{argmin}} \sum_{i=1}^{n} \nu_i L\left(y_i, g^{-1}\left(\widehat{\text{score}}_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \mathbf{a}_m)\right)\right). \tag{4.3}$$
            b) Update $\widehat{\text{score}}_m(\boldsymbol{x}) = \widehat{\text{score}}_{m-1}(\boldsymbol{x}) + \beta T(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$
               for some specified shrinkage coefficient $\beta$.
        End for
3. Output:
            $$\widehat{\pi}_D^{\text{boost}}(\boldsymbol{x}) = g^{-1}\left(\widehat{\text{score}}_{m^\star}(\boldsymbol{x})\right)$$
```

where $m^\star$ is the optimal iteration that can be selected for instance by cross validation.

Boosting is thus an iterative method based on the idea that combining many weak learners should result in a powerful one.

4.2 Binary regression trees as weak learners

In this paper, we use binary regression trees as weak learners. This is the case in the majority of applications of boosting to insurance. Let $\mathrm{I}[\cdot]$ be the indicator function, equal to 1 if the condition appearing in the brackets is fulfilled and to 0 otherwise. Trees only use binary features $\mathrm{I}[X_j < t]$ for ordered features X_j, with arbitrary threshold t or $\mathrm{I}[X_j \in \mathcal{S}]$ for unordered categorical feature X_j with $\mathcal{S}$ an arbitrary subset of the levels. Precisely,

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

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

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

$$B_{jk} = \mathrm{I}[X_j \in \mathcal{S}_k] \text{ for every } \mathcal{S}_k \subset \{\ell_1, \ldots, \ell_m\}, \quad \mathcal{S}_k \neq \emptyset.$$

  Considering all nonempty subsets $\mathcal{S}_k \subset \{\ell_1, \ldots, \ell_m\}$ 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 $\ell_1, \ell_2, \ldots, \ell_{m_j}$ 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 weak learners $T(\boldsymbol{x}; \mathbf{a}_m)$ of the form

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

where $\left\{\chi_t^{(m)}\right\}_{t \in \mathcal{T}_m}$ is the partition of the feature space χ induced by the regression tree $T(\boldsymbol{x}; \mathbf{a}_m)$ 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,

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

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

4.3 Boosting with Poisson deviance loss function under log-link

When we work with the log-link function and a response obeying Poisson distribution [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

$$\mathcal{D}^{(m)} = \left\{(r_{1,m}, \boldsymbol{x}_1, \nu_{1,m}), (r_{2,m}, \boldsymbol{x}_2, \nu_{2,m}), \ldots, (r_{n,m}, \boldsymbol{x}_n, \nu_{n,m})\right\}$$

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

$$\nu_{i,m} = \nu_i \exp\left(\widehat{\text{score}}_{m-1}(\boldsymbol{x}_i)\right) \text{ and } r_{i,m} = \frac{y_i}{\exp\left(\widehat{\text{score}}_{m-1}(\boldsymbol{x}_i)\right)}.$$

In fact, under the Poisson deviance loss function (3.1), (4.3) using the log-link function, that is,

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

can be rewritten as

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

A formal proof can be found in Proposition 3.1 in Hainaut et al. (2022). Boosting can thus be performed on the responses, proceeding in an iterative way, by dividing the response $r_{i,m}$ with $\exp\left(T(\boldsymbol{x}_i;\mathbf{a}_m)\right)$ and multiplying $\nu_{i,m}$ with $\exp\left(T(\boldsymbol{x}_i;\mathbf{a}_m)\right)$ at each step. This shows that gradient boosting introduces an extra step, which is unnecessary and leads to an approximation that can be easily avoided with boosting. This point was initially made by Wüthrich and Buser (2019).

This leads to the following algorithm:

1. Initialization:

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

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

2. Main procedure:

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

a) Calculate weights

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

and ratios

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

b) Calculate

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

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

End for

3. Output:

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

where $m^\star$ is the optimal iteration that can be selected for instance 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 correspond to actual over expected ratios, or AE ratios often used in insurance studies. Weights $\nu_{i,m}$ are estimated expected responses at the preceding iteration $m-1$.

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 of tree $T(\boldsymbol{x};\mathbf{a}_m)$ is defined as

$$R_\alpha(T(\boldsymbol{x};\mathbf{a}_m)) = \sum_{i=1}^{n} \nu_{i,m} L\Big(r_{i,m}, \exp\left(T(\boldsymbol{x}_i;\mathbf{a}_m)\right)\Big) + \alpha|T(\boldsymbol{x};\mathbf{a}_m)|,$$

where the parameter α is a positive real number and $|T(\boldsymbol{x};\mathbf{a}_m)|$ denotes the number of terminal nodes of $T(\boldsymbol{x};\mathbf{a}_m)$, called the complexity of tree $T(\boldsymbol{x};\mathbf{a}_m)$. The cost-complexity measure is thus a combination of the in-sample deviance and a penalty for the complexity of the tree under consideration (more terminal nodes result in a more flexible, and hence complex model). The parameter α is referred to as the regularization parameter and shares the same unit as deviance. For ease of interpretation, it is often normalized by the in-sample deviance of the root tree, denoted D_{root}, which leads to the cost complexity parameter

$$\text{cp} = \frac{\alpha}{D_{\text{root}}}.$$

The cost-complexity measure $R_\alpha(T(\boldsymbol{x}; \mathbf{a}_m))$ can thus be rewritten as

$$R_\alpha(T(\boldsymbol{x}; \mathbf{a}_m)) = \sum_{i=1}^{n} \nu_{i,m} L\Big(r_{i,m}, \exp\big(T(\boldsymbol{x}_i; \mathbf{a}_m)\big)\Big) + \text{cp}\, D_{\text{root}} |T(\boldsymbol{x}; \mathbf{a}_m)|.$$

Note that `rpart` provides complexity parameters rather than regularization parameters. 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 in the Poisson case. Indeed, at each iteration, the adaptive boosting tree algorithm relies on the minimal cost-complexity pruning procedure widely used within the `rpart` package and among users. Moreover, `rpart` supports the Poisson deviance loss (which is not the case for general Tweedie deviance losses).

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 $2^{\mathtt{D}}$ terminal nodes, each terminal node having `D` ancestors. More precisely, each terminal node of a regression tree with `maxdepth` = `D` belongs to generation `D` + 1, the first generation being the root node (node 0 in Figure 1), the second generation being the two child nodes of the root node (nodes 1 and 2 in Figure 1), the third generation being nodes 3, 4, 5, and 6 in Figure 1, and so on. 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.

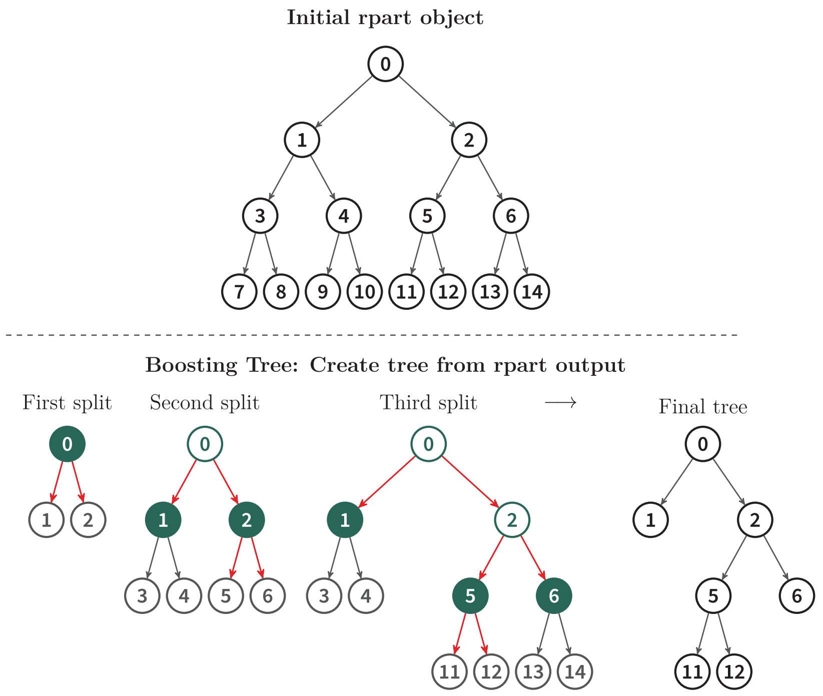

Figure 1. Illustration of the construction of the BT tree for $\mathtt{ID} = 3$ based on `rpart` output. At each step, the splitting candidates are highlighted in green, and the chosen strategy (best improvement by splitting this node) is highlighted in red.

Trees with interaction depth ID correspond to trees with $J = \mathtt{ID} + 1$ terminal nodes such that at most two terminal nodes have ID ancestors. Unlike a tree with $\mathtt{maxdepth} = \mathtt{D}$, which has $J = 2^{\mathtt{D}}$ terminal nodes, a tree with $\mathtt{ID} = \mathtt{D}$ contains $J = \mathtt{D} + 1$ terminal nodes. Therefore, any tree with an interaction depth ID can be seen as a subtree of a tree with $\mathtt{maxdepth} = \mathtt{ID}$. That is why, at each iteration, a tree with $\mathtt{maxdepth} = \mathtt{ID}$ is first built with `rpart` without any other constraints. In case $\mathtt{ID} = 1$, the resulting tree corresponds to the selected interaction depth, so that the obtained tree is returned without any other adjustments. Otherwise, when $\mathtt{ID} > 1$,

- 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.

This leads to the following update of the algorithm's main procedure explained in Section 4.3:

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

a) Calculate weights

$$\nu_{i,m} = \nu_i \exp\left(\widehat{\mathrm{score}}_{m-1}(\boldsymbol{x}_i)\right)$$

```
    and ratios
```

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

```
    b) Fit a regression tree $T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$ with maxdepth = ID on the working training set
```

$$\mathcal{D}^{(m)} = \left\{(\nu_{i,m}, r_{i,m}, \boldsymbol{x}_i), i = 1, \ldots, n\right\}.$$

```
    c) If ID = 1, then $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m) = T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$.
       If ID > 1, then prune $T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$ (as explained in Sections 5.3 and 5.4)
       to get $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$, unless $|T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)| \leq \text{ID} + 1$ in which case $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m) = T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$.
    d) Update $\widehat{\text{score}}_m(\boldsymbol{x}) = \widehat{\text{score}}_{m-1}(\boldsymbol{x}) + T(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$.
End for
```

Notice that the tree $T_0(\boldsymbol{x}; \widehat{\mathbf{a}}_m)$ 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 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 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 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 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.

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 ABT 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 (that are the risk exposures) in line with the approach presented in the preceding section. Moreover, it is worth noticing that the response variable is not the number of claims but the claim frequency (number of claims divided by the exposure-to-risk).

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

tweedie.power: Tweedie power parameter for the response under consideration, which must be set to 1 for the Poisson distribution. Moreover, only the latter distribution is currently available.

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 subsampled 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. This parameter is generally set to cv.folds -1.

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 subselection 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^2) 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 2. 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.

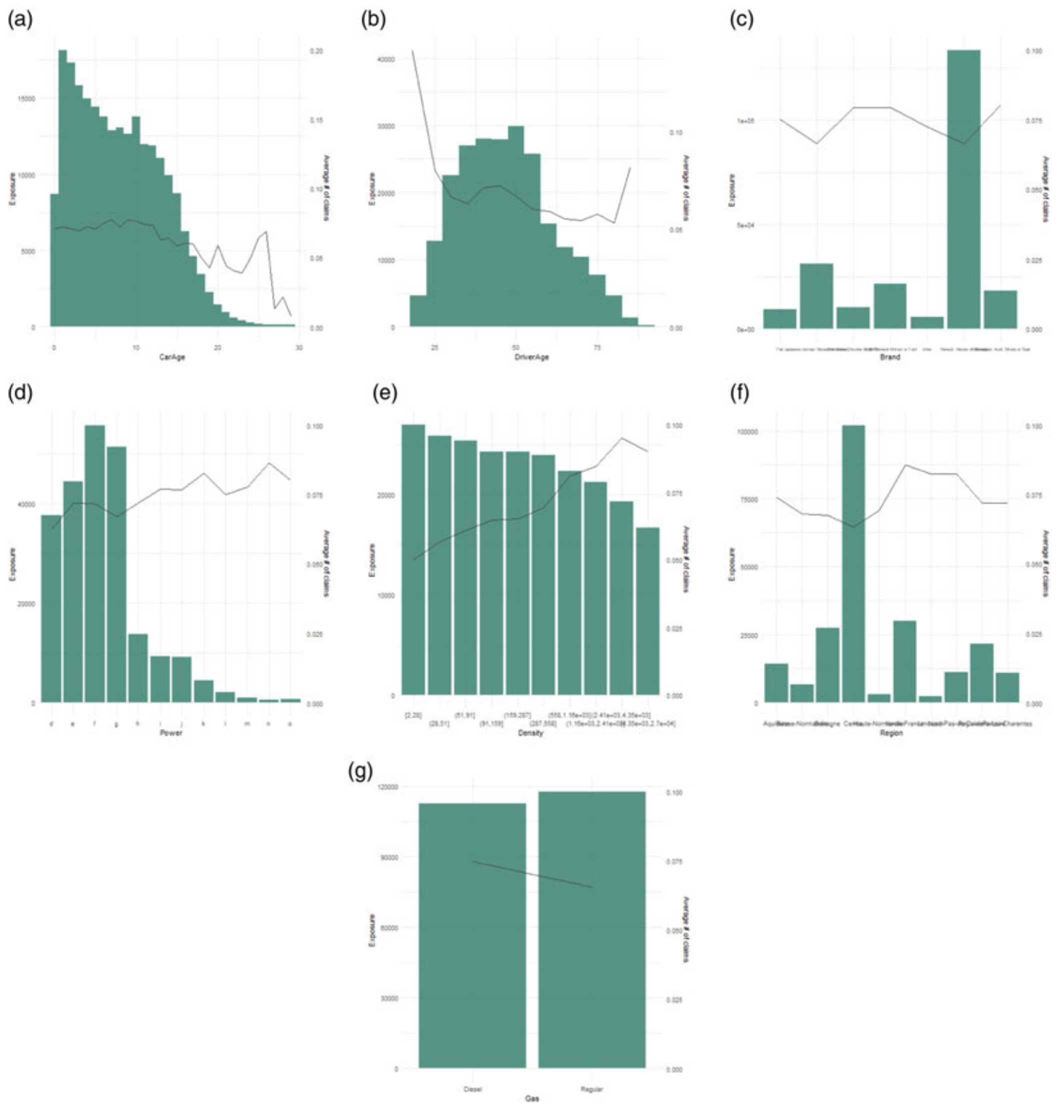

Figure 2. Univariate descriptive analysis. (a) CarAge. (b) DriverAge. (c) Brand. (d) Power. (e) Density. (f) Region. (g) Gas.

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* that 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* that contains the remaining 20% records.

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

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 nonterminal 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, three folds were used.
- For all fitted model, 5,000 boosting iterations have been performed.

For each competitor, seven 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 fourth 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 = (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 be 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 is 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.[1] We also mention that the represented total set deviance is obtained as the sum of the individual deviances divided by the number of records within the considered set.

In Figure 3 (resp. Figure 4), the validation set errors (resp. cross-validation errors) are shown for each competitor model across the seven parameters combination. The vertical dotted line represents the iteration which minimizes the error across all the seven runs. For clarity, the minimal validation set and cross-validation errors are also summarized on Figure 5. 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. 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., fifth run for ABT, seventh run for BT, second 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 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

[1]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.

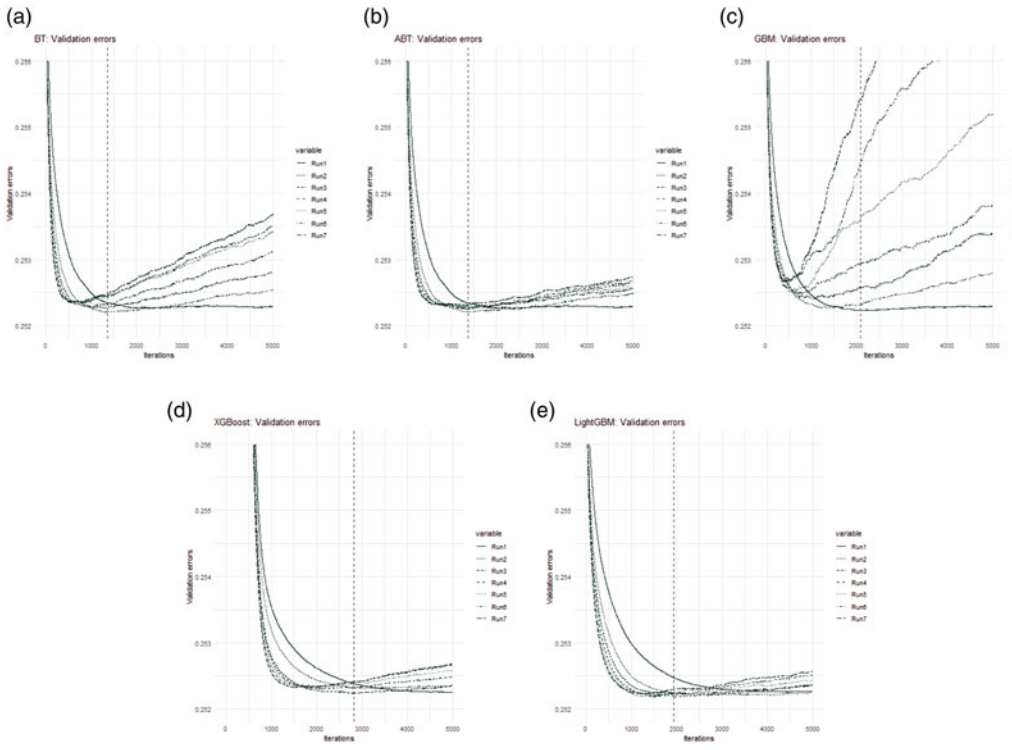

Figure 3. Validation set deviance. (a) `BT`. (b) `ABT`. (c) `gbm3`. (d) `xgboost`. (e) `lightgbm`.

number of iterations in the cross-validation case are considered. Sometimes, only the results for the best run are displayed to avoid increasing the number of results.

The left-hand side of Figure 7 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 7 where these five initial bins are divided into twenty equidistant bins. It clearly seems that the predictions' distributions are similar for all models.

Following Tevert (2013), simple lift charts are represented in Figure 8. These plots are easily obtained via the following process:

1. Compute the model predictions on the test set.
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 overestimate the highest frequencies and underestimate the lowest ones, with a breaking point around the fourth decile. Notice that this may suggest the need for an additional auto-calibration step. We refer the interested reader to Ciatto et al. (2023).

The concentration curve and the Lorenz curve have also been investigated. More precisely, we focused on two resulting metrics, namely the area bBetween 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. (2019), the Lorenz curve at a given level α represents the share of predicted claims

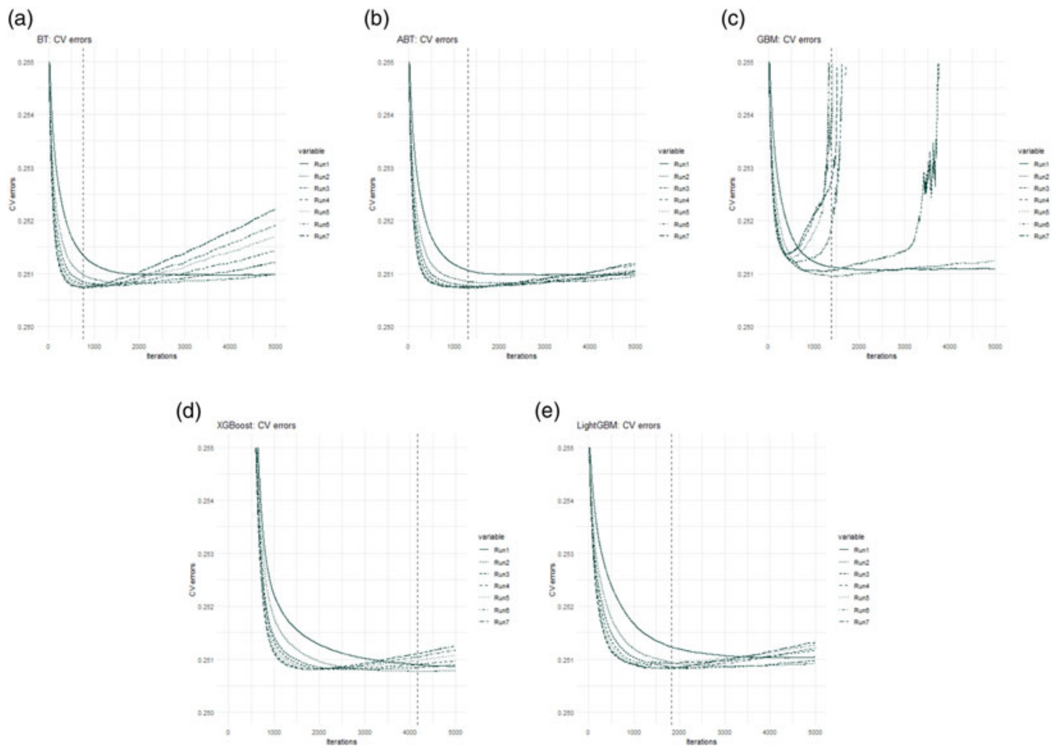

Figure 4. Cross-validation deviance. (a) `BT`. (b) `ABT`. (c) `gbm3`. (d) `xgboost`. (e) `lightgbm`.

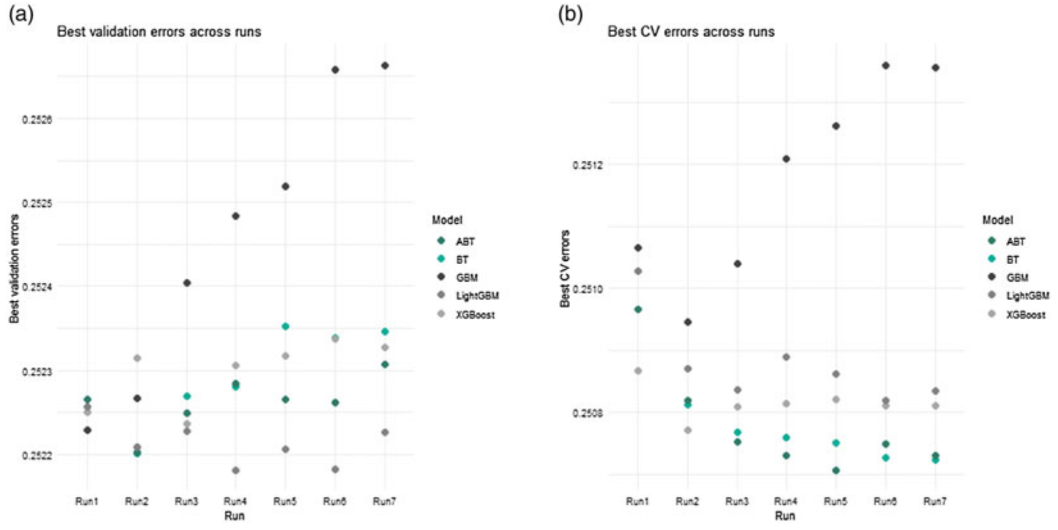

Figure 5. Minimal deviance for each run. (a) Validation set. (b) Cross-validation.

corresponding to the α% 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:

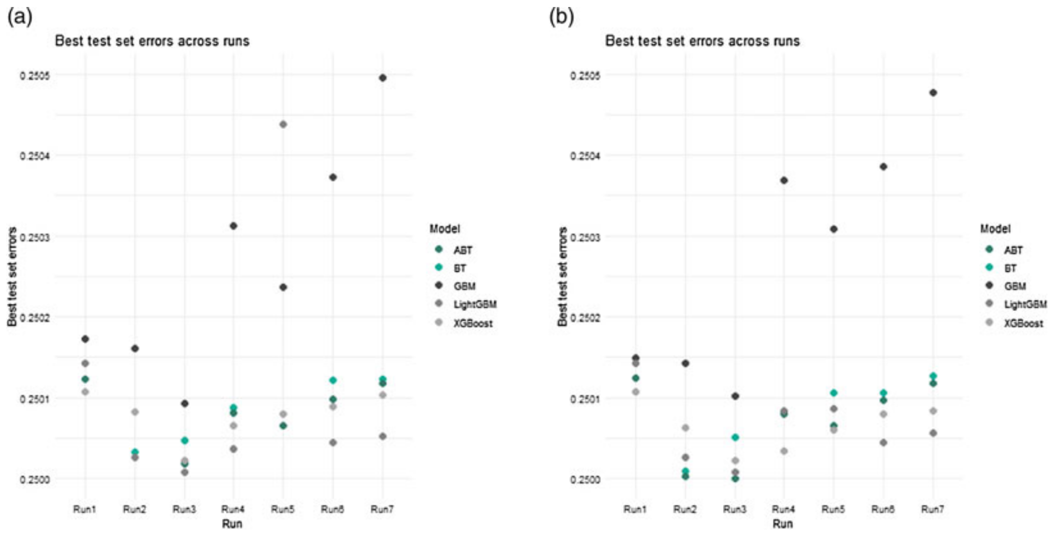

Figure 6. Minimal test set deviance for each run. (a) Validation set. (b) Cross-validation.

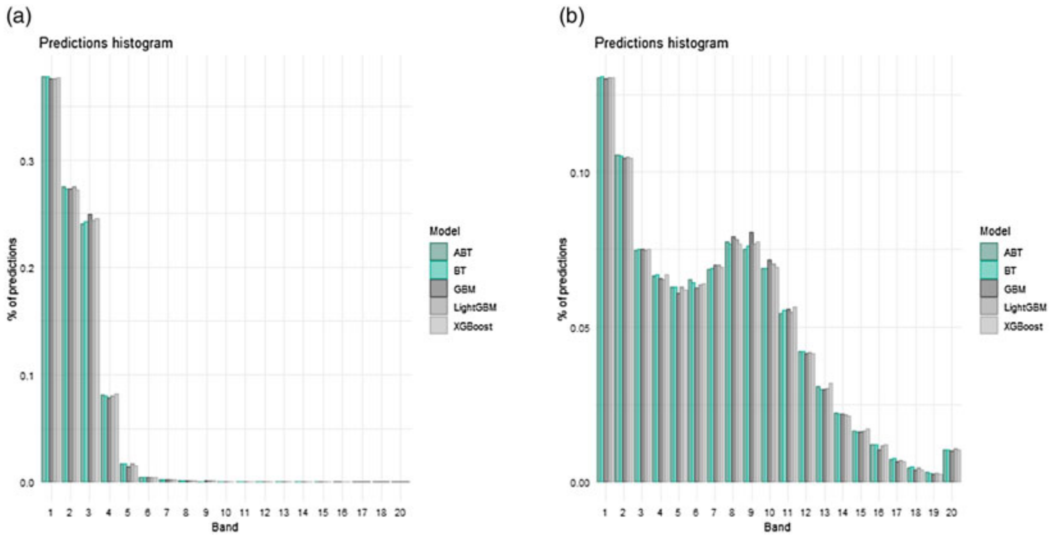

Figure 7. Predictions' histogram – Min. predicted value = 7.52×10^{-5}, band length = 0.025 for the left graph (0.006 on the right graph). (a) All bands. (b) Zoom on first five bands.

- 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 in Figure 9 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 that 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

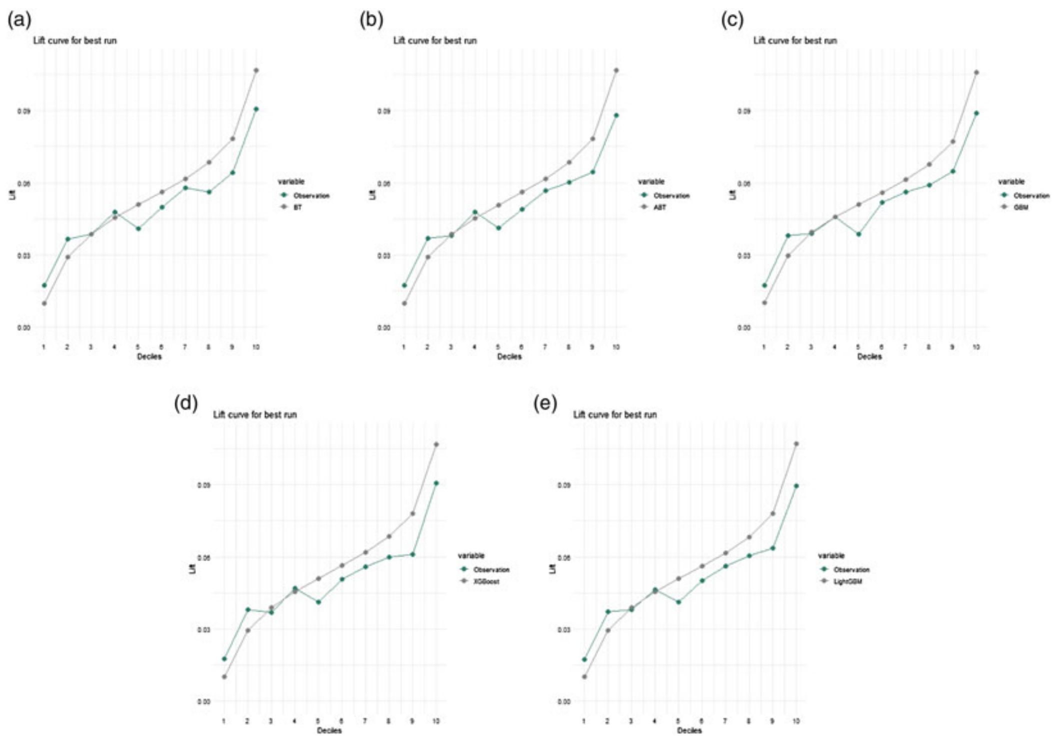

Figure 8. Simple lift chart for best CV-runs. (a) `BT`. (b) `ABT`. (c) `gbm3`. (d) `xgboost`. (e) `lightgbm`.

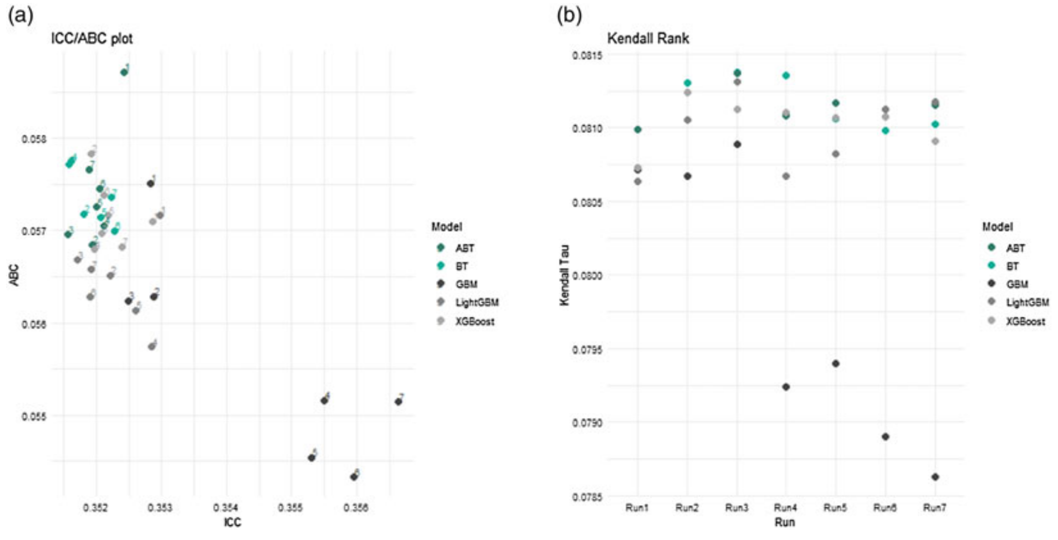

Figure 9. Rank performances. (a) ICC/ABC. (b) Kendall Tau's.

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 9. This computation seems to heavily favor the `BT` approach. As

Table 6.1. Variable importance (in %)

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

pointed out by Wüthrich (2023), it is worth mentioning that ICC and ABC are not consistent scoring rules for the mean. In the class of autocalibrated predictors, ICC is a consistent scoring rule and ABC is always zero. However, the models considered in our illustration are not necessarily autocalibrated.

We finally end up this comparison by discussing the so-called variable importance. To that end, the one-hot encoded importance has been summed up for the xgboost package. The results are displayed in Table 6.1. 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 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. Moreover, the convergence to a root node seems to be the natural stopping criterion for the Adaptive Boosting, and this approach still has be implemented. Finally, an extension of the package to all members of the Tweedie family would be highly relevant to actuarial science applications. In particular, such an extension would require modifications to the rpart package to enable it to handle Tweedie distributions. We are currently working on such an extension to our BT package.

Data availability statement. To replicate the results, the BT package used in this article can be downloaded via the DOI link https://doi.org/10.5281/zenodo.10071474 as well as via the Comprehensive R Archive Network (CRAN) https://cran.r-project.org/package=BT (version v0.4).

Financial support. This research received no specific grant from any funding agency, commercial or not-for-profit sector.

Competing interests. Authors Gireg Willame, Julien Trufin and Michel Denuit declare none.

References

Ciatto, N., Denuit, M., Trufin, J. & **Verelst, H.** (2023). Does autocalibration improve goodness of lift? *European Actuarial Journal*, **13**, 479–486.

Denuit, M., Hainaut, D. & **Trufin, J.** (2020). *Effective statistical learning methods for actuaries II: Tree-based methods and extensions*. Springer actuarial lecture notes series. Springer.

Denuit, M., Sznajder, D. & **Trufin, J.** (2019). 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). CASDatasets: Insurance datasets. https://github.com/dutangc/CASdatasets

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, **29**, 1189–1232.

Guelman, L. (2012). Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications*, **39**, 3659–3667.

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*, **841**, 2022–2866.

Hastie, T., Tibshirani, R. & **Friedman, J.** (2008). *The elements of statistical learning* (2nd ed.). Springer.

Henckaerts, R., Cote, M.-P., Antonio, K. & **Verbelen, R.** (2021). Boosting insights in insurance tariff plans with tree-based machine learning methods. *North American Actuarial Journal*, **25**, 255–285.

Huyghe, J., Trufin, J. & **Denuit, M.** (2022). Boosting cost-complexity pruned trees on Tweedie responses: The ABT machine for insurance ratemaking. *Scandinavian Actuarial Journal*. https://doi.org/10.1080/03461238.2023.2258135

Lee, S. C. & **Lin, S.** (2018). Delta boosting machine with application to general insurance. *North American Actuarial Journal*, **22**, 405–425.

Liu, Y., Wang, B. & **Lv, S.** (2014). Using multi-class AdaBoost tree for prediction frequency of auto insurance. *Journal of Applied Finance and Banking*, **4**, 45–53.

Pesantez-Narvaez, J., Guillen, M. & **Alcaniz, M.** (2019). Predicting motor insurance claims using telematics data- XGBoost versus logistic regression. *Risks*, **7**, 1–16.

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. (2023). Model selection with Gini indices under auto-calibration. *European Actuarial Journal*, **13**, 469–477.

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

Yang, Y., Qian, W. & **Zou, H.** (2018). Insurance premium prediction via gradient tree-boosted Tweedie compound Poisson models. *Journal of Business & Economic Statistics*, **36**, 456–470.

Cite this article: Willame G, Trufin J and Denuit M (2024). Boosted Poisson regression trees: a guide to the BT package in R, *Annals of Actuarial Science*, 1–21. https://doi.org/10.1017/S174849952300026X