BOOSTING COST-COMPLEXITY PRUNED TREES ON TWEEDIE RESPONSES: THE ABT MACHINE FOR INSURANCE RATEMAKING

Julie Huyghe
Department of Mathematics
Université Libre de Bruxelles (ULB)
Brussels, Belgium

Julien Trufin
Department of Mathematics
Université Libre de Bruxelles (ULB)
Brussels, Belgium

Michel Denuit
Institute of Statistics, Biostatistics and Actuarial Science
UCLouvain
Louvain-la-Neuve, Belgium

September 13, 2023

Abstract

This paper proposes a new boosting machine based on forward stagewise additive modeling with cost-complexity pruned trees. In the Tweedie case, it deals directly with observed responses, not gradients of the loss function. Trees included in the score progressively reduce to the root-node one, in an adaptive way. The proposed Adaptive Boosting Tree (ABT) machine thus automatically stops at that time, avoiding to resort to the time-consuming cross validation approach. Case studies performed on motor third-party liability insurance claim data demonstrate the performances of the proposed ABT machine for ratemaking, in comparison with regular gradient boosting trees.

Keywords: Risk classification, Boosting, Gradient Boosting, Regression Trees, Cost-complexity pruning.

1 Introduction and motivation

Boosting emerged from the field of machine learning and became rapidly popular among insurance analysts due to its outstanding performance. Broadly speaking, boosting is an iterative fitting procedure that sequentially improves the overall model fit by incorporating a new base learner at each step, resulting in an accurate prediction. Boosting works by adding the newly fitted base learner to the previously estimated score in each iteration. For a general introduction to statistical learning in insurance, we refer readers to Denuit et al. (2019a). For an in-depth discussion of boosting in the context of insurance studies, including applications to tree-based methods and neural networks, we refer readers to Denuit et al. (2019b, 2020).

To simplify numerical computations, boosting is often applied on gradients of the loss function. Rather than minimizing the deviance associated with the responses, gradient boosting applies a least-squares principle on its gradients. This approach has made boosting very popular among data analysts, and several open-source software packages now implement highly effective boosting algorithms, such as the Extreme Gradient boosting (XGBoost) algorithm. Applications of gradient boosting in insurance include studies by Guelman (2012), Yang et al. (2018), Lee and Lin (2018), Pesantez-Narvaez et al. (2019), and Henckaerts et al. (2021).

However, boosting is not limited to gradients and can also be regarded as an iteratively re-weighted, or re-offseted procedure applied to the original data, as established by Hainaut et al. (2022). Precisely, it is shown that it is unnecessary to boost gradients in some insurance applications, as boosting can be easily performed directly with responses under Tweedie deviance and log-link. Given that this setting is now widely adopted by actuaries to conduct their analyzes, this result is widely applicable to insurance studies. This extends the results previously obtained by Wüthrich and Buser (2019) in the Poisson case with canonical link function to the whole Tweedie family under log-link. This alternative view to boosting makes the approach very intuitive and better suited to the shape of data commonly encountered in insurance applications, such as low counts for claim numbers in personal lines and highly skewed severity data.

To prevent overfitting, cross validation is used to stop the boosting algorithms when its prediction capabilities start to deteriorate. Early stopping plays a central role in ensuring 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 the out-of-sample test data or as measured by cross validation. The latter technique consists in randomly splitting the training data set into several 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 the goodness of fit and model complexity, as a model too close to the training set often produces worse predictions by reproducing noise in the data or overfitting the training data. However, cross-validation can be computationally expensive.

The standard boosting algorithm has another drawback in that it does not adapt along the sequence of scores produced by the forward stagewise additive procedure. Rather than 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 idea leads to the proposed ABT (Adaptive Boosting Trees) machine, where trees added to the score progressively adapt their complexity to the amount of information left to discover from the data.

The idea behind the proposed ABT machine is based on tree pruning. Decision trees are widely used in machine learning and are appreciated for their interpretability in various domains. For a comprehensive overview of the latest advances in tree-based models, readers are directed to the review by Costa and Pedreira (2022). It is noteworthy that while tree ensemble methods such as bagging trees and boosting trees are conceptually related to trees, they have emerged as a distinct area of study with unique considerations, challenges, and literature. Among the techniques developed for decision trees, pruning has been extensively investigated in the literature. However, the term "pruning" can be ambiguous in the context of tree-related literature, and it is crucial to distinguish between pruning of a single tree in the context of tree literature versus ensemble pruning as discussed in the literature on ensemble techniques. Two pruning approaches for a single tree exist: pre-pruning and post-pruning. For pre-pruning literature, please refer to Quinlan (1986), Wu et al. (2016) and Garcia Leiva et al. (2019). Although post-pruning remains popular, recent innovations have been limited, and Breiman's cost complexity pruning (1984) remains the reference in post-pruning methods. For additional details about tree pruning methods, readers are referred to Windeatt and Ardeshir (2001). The concept of "pruning the ensemble" has been introduced by Margineantu and Dietterich (1997). They investigated the possibility of selecting a subset of decision trees constructed by the Adaboost algorithm to achieve comparable performance. The authors presented five pruning methods, including early stopping and KL-divergence pruning, kappa pruning, kappa error convex hull pruning, and reduced error pruning. Their results demonstrated that, in most cases, the produced tree ensemble can be significantly pruned (reduced). These findings have been widely accepted by the machine learning community, and ensemble pruning has become a trendy topic explored in recent years. As a result, numerous articles have proposed new ensemble pruning methods, including Tamon and Xiang (2000), Hernandez-Lobato et al. (2006), Thompson (1999), Chen et al. (2009) and Vidal and Schiffer (2020).

In this paper, we propose a novel approach to boosting algorithms by incorporating single-tree pruning directly into the algorithm. By doing so, the ABT machine produces an added benefit of having a built-in stopping criterion since the score stops growing when the newly added tree reduces to the root node one. As illustrated in the second case study in Section 4.2, our approach requires significantly fewer trees than traditional boosting algorithms. Therefore, we do not need to use ensemble pruning methods to further reduce the number of trees. This has the potential to improve computational efficiency and reduce the risk of overfitting, making our approach a promising alternative to existing methods in the field.

A small bag fraction can be used to avoid the ABT machine getting trapped in a sub-optimal solution when the size of the trees considered early in the algorithm is too small, as illustrated in Section 4.1.3. The use of a small bag fraction is thus needed only when interaction depth is kept small, which is not required with the ABT machine. Large trees can be fitted in the early stages with ABT machine, as shown in Section 4.2, which gradually simplify until they reach the single-node tree where the ABT machine stops. Fitting large trees early in the ABT algorithm is actually recommended, as illustrated in Section 4.2. The use of a bag fraction is thus not necessary with ABT, provided that the data set comprises

fairly numerous and rich covariates (as in the second case study in Section 4.2) to produce sufficiently large trees. That is why we do not use a bag fraction throughout this paper except in Section 4.1.3 where we consider for illustration purposes small trees for ABT.

The remainder of this paper is organized as follows. Section 2 recalls the boosting principle and its forward stagewise additive modeling. In Section 3, we present the ABT machine that is proposed in this paper, detailing its underlying architecture and functioning. To evaluate the efficiency of the proposed method, we present the results of two case studies conducted using motor insurance data in Section 4. Section 5 summarizes the results and concludes the paper.

2 Boosting

2.1 Supervised learning

The aim of actuarial ratemaking is to evaluate the pure premium as accurately as possible. The target is thus the conditional expectation $\mu(\boldsymbol{X}) = \mathrm{E}[Y|\boldsymbol{X}]$ of the response Y (claim number or claim amount for instance) given the available information summarized in a vector $\boldsymbol{X}$ of features $X_1, X_2, \ldots, X_p$. The feature space $\mathcal{X}$ is a subset of $\mathbb{R}^p$. The function $\boldsymbol{x} \mapsto \mu(\boldsymbol{x}) = \mathrm{E}[Y|\boldsymbol{X} = \boldsymbol{x}]$ is unknown to the actuary and is approximated by a working predictor $\boldsymbol{x} \mapsto \widehat{\mu}(\boldsymbol{x})$ entering premium calculation.

Lack of accuracy for $\widehat{\mu}(\boldsymbol{x})$ is defined by the generalization error

$$\mathrm{Err}(\widehat{\mu}) = \mathrm{E}\left[L(Y, \widehat{\mu}(\boldsymbol{X}))\right], \tag{2.1}$$

where $L(.,.)$ is the loss function measuring the discrepancy between its two arguments and the expected value is over the joint distribution of $(Y, \boldsymbol{X})$. In actuarial studies, loss functions typically correspond to the deviance of distributions within the Exponential dispersion family, to which GLM is applicable. The objective is to find a function $\boldsymbol{x} \mapsto \widehat{\mu}(\boldsymbol{x})$ of the features that minimizes the generalization error.

2.2 Ensemble learning methods

Ensemble techniques assume structural models of the form

$$\mu(\boldsymbol{x}) = g^{-1}\left(\mathrm{score}(\boldsymbol{x})\right) \text{ with } \mathrm{score}(\boldsymbol{x}) = \sum_{m=1}^{M} T(\boldsymbol{x}; \mathbf{a}_m), \tag{2.2}$$

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 $\boldsymbol{x}$, characterized by parameters $\mathbf{a}_m$. In (2.2), the score is the function of features $\boldsymbol{x}$ mapped to $\mu(\boldsymbol{x})$ by the inverse of the link function g.

Let $\mathcal{D} = \{(\nu_1, y_1, \boldsymbol{x}_1), (\nu_2, y_2, \boldsymbol{x}_2), \ldots, (\nu_n, y_n, \boldsymbol{x}_n)\}$ be the set of observations used to fit model (2.2), called training set, where ν_i is the weight associated to observation i. Estimating the score by minimizing the corresponding training sample estimate of the generalization error (2.1), that is,

$$\min_{\{\mathbf{a}_m\}_1^M} \sum_{i=1}^{n} \nu_i L\left(y_i, g^{-1}\left(\sum_{m=1}^{M} T(\boldsymbol{x}_i; \mathbf{a}_m)\right)\right) \tag{2.3}$$

is in general not feasible. This requires computationally intensive numerical optimization technique that can be time-consuming. One way to circumvent this issue is to approximate the solution to (2.3) by using a greedy forward stagewise approach, also known as boosting.

2.3 Forward stagewise additive modeling

Boosting, or forward stagewise additive modeling, involves fitting a single function sequentially and adding it to the expansion of previously fitted terms. Unlike a stepwise approach, where previous terms are readjusted each time a new term is added, each fitted term in boosting is not readjusted as new terms are added to the expansion. Specifically, we start by computing

$$\widehat{\mathbf{a}}_1 = \underset{\mathbf{a}_1}{\operatorname{argmin}} \sum_{i=1}^{n} \nu_i L\left(y_i, g^{-1}\left(\widehat{\operatorname{score}}_0(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \mathbf{a}_1)\right)\right) \tag{2.4}$$

where $\widehat{\operatorname{score}}_0(\boldsymbol{x})$ is an initial guess (for instance, just an intercept). Then, at each iteration $m \geq 2$, we solve the subproblem

$$\widehat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\operatorname{argmin}} \sum_{i=1}^{n} \nu_i L\left(y_i, g^{-1}\left(\widehat{\operatorname{score}}_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \mathbf{a}_m)\right)\right) \tag{2.5}$$

with

$$\widehat{\operatorname{score}}_{m-1}(\boldsymbol{x}) = \widehat{\operatorname{score}}_{m-2}(\boldsymbol{x}) + T(\boldsymbol{x}; \widehat{\mathbf{a}}_{m-1}).$$

By adopting a meaningful stopping rule, we end up at iteration M with estimates of the form $\widehat{\mu}_{\mathcal{D}}^{\text{boost}}(\boldsymbol{x}) = g^{-1}\left(\widehat{\operatorname{score}}_M(\boldsymbol{x})\right)$, which can then be used in pure premium calculations.

The boosting algorithm is presented in Algorithm 1.

Algorithm 1 Forward Stagewise Additive Modeling.

---

1. Initialization :

Initialize $\widehat{\operatorname{score}}_0(\boldsymbol{x})$ to be a constant. For instance:

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

2. Main procedure :

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

(2.1) Compute $\widehat{\mathbf{a}}_m$ as defined in Equation (2.5).

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

End for

3. **Output:**

$$\widehat{\mu}_{\mathcal{D}}^{\text{boost}}(\boldsymbol{x}) = g^{-1}\left(\widehat{\operatorname{score}}_M(\boldsymbol{x})\right).$$

	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 sums with Gamma-distributed severities
$\xi = 2$	Continuous, positive	Gamma
$2 < \xi < 3$	Continuous, positive	-
$\xi = 3$	Continuous, positive	Inverse Gaussian
$\xi > 3$	Continuous, positive	-

Table 1: Tweedie distributions.

2.4 Tweedie deviance loss function under log-link

2.4.1 Tweedie family of distributions

In practice, actuaries often use distributions from the Tweedie class along with the log-link function to model responses. The Tweedie class comprises members of the Exponential Dispersion family of distributions that have power variance functions $V(\mu) = \mu^\xi$ for some ξ.

Table 1 provides a list of all Tweedie distributions, with negative values of ξ corresponding to continuous distributions across the entire real axis. For $0 < \xi < 1$, no member of the Exponential Dispersion family exists and only cases where $\xi \geq 1$ are of interest for insurance applications. For the sake of completeness, we also consider the square loss function, thereby allowing $\xi = 0$.

2.4.2 Tweedie deviance-based boosting under log-link

Tweedie deviance loss function is given by

$$L(y, \widehat{\mu}(\boldsymbol{x})) = \begin{cases} (y - \widehat{\mu}(\boldsymbol{x}))^2 \text{ for } \xi = 0 \\ 2\left(y \ln \frac{y}{\widehat{\mu}(\boldsymbol{x})} - (y - \widehat{\mu}(\boldsymbol{x}))\right) \text{ for } \xi = 1 \\ 2\left(-\ln \frac{y}{\widehat{\mu}(\boldsymbol{x})} + \frac{y}{\widehat{\mu}(\boldsymbol{x})} - 1\right) \text{ for } \xi = 2 \\ 2\left(\frac{\max\{y,0\}^{2-\xi}}{(1-\xi)(2-\xi)} - \frac{y\widehat{\mu}(\boldsymbol{x})^{1-\xi}}{1-\xi} + \frac{\widehat{\mu}(\boldsymbol{x})^{2-\xi}}{2-\xi}\right) \text{ for } \xi > 1 \text{ and } \xi \neq 2. \end{cases} \tag{2.6}$$

For $\xi = 0$, the square loss function is recovered, whereas $\xi = 1$ and 2 correspond to the Poisson and Gamma deviance functions, respectively.

In insurance studies, gradient boosting has often been used, where the response is replaced with the gradient of the loss function to expedite calculations. However, this approach can obscure the analysis and is unnecessary in the Tweedie case, as explained next. Define the working weight

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

and the working response

$$\tilde{r}_{mi} = \frac{y_i}{\exp\left(\widehat{\text{score}}_{m-1}(\boldsymbol{x}_i)\right)}$$

at iteration m. For any loss function (2.6) and log-link, Hainaut et al. (2022) established that

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

can be rewritten as

$$\widehat{\mathbf{a}}_m = \underset{\mathbf{a}_m}{\operatorname{argmin}} \sum_{i=1}^{n} \nu_{mi} L\Big(\tilde{r}_{mi}, \exp\left(T(\boldsymbol{x}_i; \mathbf{a}_m)\right)\Big), \tag{2.7}$$

so that solving (2.7) amounts to fit $T(\boldsymbol{x}_i; \mathbf{a}_m)$ on the working training set

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

Algorithm 1 simplifies to Algorithm 2 in the Tweedie case under log-link.

Algorithm 2 Tweedie Deviance-based Boosting under Log-link.

---

1. Initialization :

Initialize $\widehat{\operatorname{score}}_0(\boldsymbol{x})$ to be a constant. For instance:

$$\widehat{\operatorname{score}}_0(\boldsymbol{x}) = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \nu_i L(y_i, \exp(\beta)).$$

2. Main procedure :

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

(2.1) Fit $T(\boldsymbol{x}; \mathbf{a}_m)$ on the working training set

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

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

End for

3. **Output:**

$$\widehat{\mu}_{\mathcal{D}}^{\text{boost}}(\boldsymbol{x}) = \exp\left(\widehat{\operatorname{score}}_M(\boldsymbol{x})\right).$$

In the Tweedie case, boosting should be preferred to gradient boosting since the latter procedure introduces an extra step which is unnecessary and leads to an approximation that can be easily avoided. This extends the point made by Wüthrich and Buser (2019) for the Poisson distribution under canonical link to the whole Tweedie class under log-link. The gradient boosting applies a least-squares principle on the gradients of the deviance. As the sum of least squares corresponds to the log-likelihood of a Gaussian model, choosing this optimization criterion is equivalent to assume that gradients are realizations of a Normal random variable. But under Tweedie deviance loss function and log-link, equation (3.7) in Hainaut et al. (2022) shows that the gradient inherits from the distribution of the response Y that is scaled and translated. Except in the normal case ($\xi = 1$), applying the least

squares principle is then ineffective and slows down the convergence compared to the Tweedie boosting as illustrated in Hainaut et al. (2022). In the rest of the paper, we work with Tweedie loss functions under log-link.

2.5 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. That is, we consider base learners $T(\boldsymbol{x}; \mathbf{a}_m)$ of the form

$$T(\boldsymbol{x}; \mathbf{a}_m) = \sum_{t \in \mathcal{T}_m} c_{tm} \mathrm{I}\left[\boldsymbol{x} \in \chi_t^{(m)}\right], \tag{2.8}$$

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 predictions in the terminal nodes, that is,

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

For details on the recursive partitioning algorithm used to build binary regression trees, we refer the reader to the seminal book by Breiman et al. (1984). For an overview of tree-based methods applied to insurance, please refer to Denuit et al. (2020).

With regression trees, a shrinkage parameter $0 \leq \gamma < 1$ is often used to slow the learning rate of the training procedure, so that instead of adding tree $T(\boldsymbol{x}; \mathbf{a}_m)$ to the score in Step 2.2 of Algorithm 2, we rather add $\gamma T(\boldsymbol{x}; \mathbf{a}_m)$. Small values for γ work best but result in larger computation time since more regression trees are needed. Empirically, values for the shrinkage parameter γ smaller than 10% yield dramatic improvements for regression estimation. Typically, γ is fixed at the lowest possible value within the computational constraints (see Friedman, 2001). In this paper, we run the models of Section 4 with shrinkage parameter $\gamma = 1\%$.

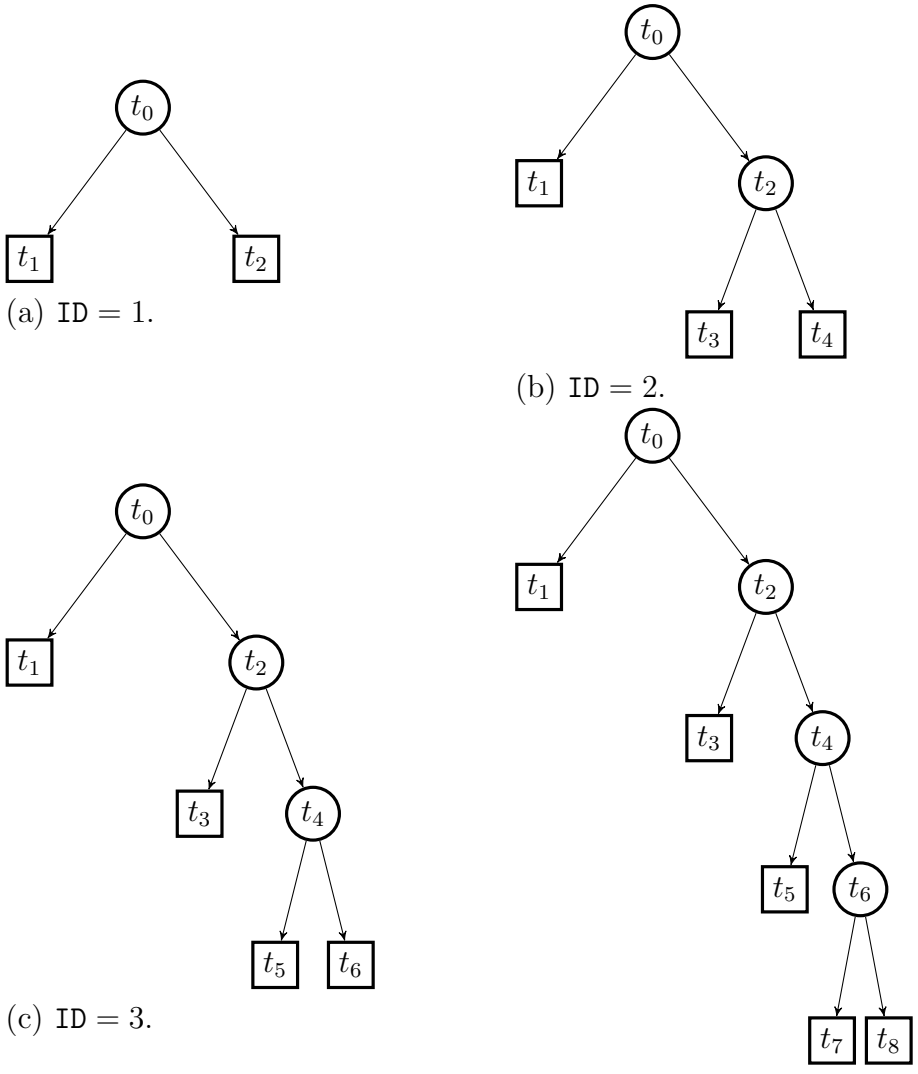
2.6 Size of the trees

Boosting trees involve two significant parameters that need to be tuned, namely the number of trees M and the size of the trees. The number of trees M is determined by a stopping criterion checked at each iteration of the boosting algorithm, while the size of the trees is selected by the analyst before the boosting process begins. The size of trees can be specified in various ways, such as the number of terminal nodes J or the depth of the tree `D`. A regression tree with depth `D` has $2^{\mathtt{D}}$ terminal nodes, each with `D` ancestors.

In the context of boosting, the size of the trees is typically controlled by the interaction depth parameter `ID`. Each subsequent split represents a higher level of interaction with the previous split features. Trees with an interaction depth of `ID` correspond to trees with $J = \mathtt{ID}+1$ terminal nodes. By setting $\mathtt{ID} = 1$, only single-split regression trees are produced, which allows capturing only the main effects of the features in the score. On the other hand, for $\mathtt{ID} = 2$, two-way interactions are permitted, and for $\mathtt{ID} = 3$, three-way interactions are

allowed, and so on. Thus, the value of `ID` determines the level of interactions permitted in the score, and a larger value of `ID` allows the model to learn deeper patterns in the data, such as illustrated in Figure 1.

In practice, the appropriate level of interaction required is often unknown, and thus, `ID` is a tuning parameter that needs to be selected carefully by considering different values and selecting the one that minimizes the generalization error estimated on a validation set or through cross-validation.



(a) `ID` = 1.

(b) `ID` = 2.

(c) `ID` = 3.

(d) `ID` = 4.

Figure 1: Example of binary regression trees with `ID` $= 1, 2, 3, 4$. Circles indicate non-terminal nodes and square boxes represent terminal nodes.

3 ABT machine

Now that the machinery behind boosting trees has been recalled, and that can be summarized by Algortihm 3, we are now prepared to describe the new machine learning tool proposed in this paper.

Algorithm 3 Tweedie Deviance-based Boosting Trees under Log-link.

---

1. Initialization :

Initialize $\widehat{\text{score}_0}(\boldsymbol{x})$ to be a constant. For instance:

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

2. Main procedure :

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

(2.1) Fit a regression tree $T(\boldsymbol{x}; \mathbf{a}_m)$ of size `ID` on the working training set

$$\mathcal{D}^{(m)} = \{(\nu_{mi}, \tilde{r}_{mi}, \boldsymbol{x}_i), i = 1, \ldots, n\}.$$

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

End for

3. **Output:**

$$\widehat{\mu}_{\mathcal{D}}^{\text{BT}}(\boldsymbol{x}) = \exp\left(\widehat{\text{score}}_M(\boldsymbol{x})\right).$$

Our approach involves fitting cost-complexity pruned trees in a forward stagewise and additive way, hence the acronym ABT for Adaptive Boosting Trees. The algorithm begins by fitting larger trees in the early stages, which then gradually simplify until they reach the single-node tree where the ABT machine stops. The stopping criterion is built into the ABT algorithm, eliminating the need for computationally-intensive cross-validation steps. Moreover, the stopping criterion embedded in the ABT machine allows the algorithm to stop automatically once the single-node tree is reached, so that there is not one tree too many that will be estimated with ABT. Additionally, `ID` or J can vary among the trees included in the score, allowing the model complexity to adapt to the data structure not yet captured by the ABT machine. The next sections carefully explain how the ABT algorithm proceeds.

3.1 Cost-complexity measure

The cost-complexity measure of tree $T(\boldsymbol{x}; \mathbf{a}_m)$ is defined as

$$R_\alpha(T(\boldsymbol{x}; \mathbf{a}_m)) = \sum_{i=1}^{n} \nu_{mi} L\Big(\tilde{r}_{mi}, \exp\left(T(\boldsymbol{x}_i; \mathbf{a}_m)\right)\Big) + \alpha |T(\boldsymbol{x}; \mathbf{a}_m)|, \tag{3.1}$$

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. As more terminal nodes result in a more flexible, and hence complex model, increasing the number of terminal nodes of a tree $T(\boldsymbol{x};\mathbf{a}_m)$ by splitting one of its terminal nodes into two children nodes will reduce the in-sample deviance of the resulting tree $T(\boldsymbol{x};\mathbf{a}'_m)$ compared to the original tree $T(\boldsymbol{x};\mathbf{a}_m)$. Therefore, the in-sample deviance always favors the more complex tree $T(\boldsymbol{x};\mathbf{a}'_m)$ over $T(\boldsymbol{x};\mathbf{a}_m)$. However, by introducing a penalty for the complexity of the tree, the cost-complexity measure may now prefer the original tree $T(\boldsymbol{x};\mathbf{a}_m)$ over the more complex one $T(\boldsymbol{x};\mathbf{a}'_m)$. $R_\alpha(T(\boldsymbol{x};\mathbf{a}'_m)) \geq R_\alpha(T(\boldsymbol{x};\mathbf{a}_m))$ if, and only if,

$$\alpha \geq \sum_{i=1}^{n} \nu_{mi} L\Big(\tilde{r}_{mi}, \exp\big(T(\boldsymbol{x}_i;\mathbf{a}_m)\big)\Big) - \sum_{i=1}^{n} \nu_{mi} L\Big(\tilde{r}_{mi}, \exp\big(T(\boldsymbol{x}_i;\mathbf{a}'_m)\big)\Big). \tag{3.2}$$

In fact, $R_\alpha(T(\boldsymbol{x};\mathbf{a}'_m)) \geq R_\alpha(T(\boldsymbol{x};\mathbf{a}_m))$ if, and only if, the deviance reduction that we get by producing tree $T(\boldsymbol{x};\mathbf{a}'_m)$ is smaller than the increase in the penalty for having one more terminal node. In situations where the reduction in deviance is insufficient to compensate the corresponding increase in penalty, we do not switch from the simpler model $T(\boldsymbol{x};\mathbf{a}_m)$ to the more intricate $T(\boldsymbol{x};\mathbf{a}'_m)$. The parameter α can be interpreted as the increase in the penalty for having one more terminal node.

3.2 Minimal cost-complexity pruning

A tree $T(\boldsymbol{x};\mathbf{a}_m)$ obtained by pruning branches from $T(\boldsymbol{x};\mathbf{a}'_m)$ is referred to as a pruned subtree, or simply a subtree, of $T(\boldsymbol{x};\mathbf{a}'_m)$. This is denoted as $T(\boldsymbol{x};\mathbf{a}_m) \preceq T(\boldsymbol{x};\mathbf{a}'_m)$.

The pruning process of an initial tree T_{init} consists of constructing a sequence of progressively smaller trees

$$T_{init}, T_{|T_{init}|-1}, \ldots, T_1, \tag{3.3}$$

where T_k is a subtree of T_{init} with k terminal nodes, $k = 1, \cdots, |T_{init}| - 1$. In particular, T_1 is only composed of the root node t_0 of T_{init}. If we denote by $\mathcal{C}(T_{init}, k)$ the class of all subtrees of T_{init} having k terminal nodes, an intuitive procedure to produce the sequence of trees (3.3) is then to select for every $k = 1, \cdots, |T_{init}| - 1$, the tree $T_k \in \mathcal{C}(T_{init}, k)$ which minimises the in-sample deviance. This procedure may generate subtrees of T_{init} that are not nested, meaning that subtree T_k may not necessarily be a subtree of T_{k+1}. Therefore, a node t from T_{init} may reappear in tree T_k even though it was previously cut off in tree T_{k+1}. That is why the minimal cost-complexity pruning described next is often preferred over the latter approach.

Consider tree $T(\boldsymbol{x};\mathbf{a}_m)$. For a fixed value of α, we define $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$ as the subtree of $T(\boldsymbol{x};\mathbf{a}_m)$ that minimises the cost-complexity measure $R_\alpha(.)$, namely

$$R_\alpha(T(\boldsymbol{x};\mathbf{a}_m(\alpha))) = \min_{T(\boldsymbol{x};\mathbf{a}) \preceq T(\boldsymbol{x};\mathbf{a}_m)} R_\alpha(T(\boldsymbol{x};\mathbf{a})). \tag{3.4}$$

Hence, at this value of α, there is no subtree of $T(\boldsymbol{x};\mathbf{a}_m)$ with lower cost-complexity measure than $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$. Because there can be more than one subtree of $T(\boldsymbol{x};\mathbf{a}_m)$ minimising

$R_\alpha(.)$, we supplement condition (3.4) with

$$R_\alpha(T(\boldsymbol{x};\mathbf{a})) = R_\alpha(T(\boldsymbol{x};\mathbf{a}_m(\alpha))) \Rightarrow T(\boldsymbol{x};\mathbf{a}_m(\alpha)) \preceq T(\boldsymbol{x};\mathbf{a}). \tag{3.5}$$

To break ties in the cost-complexity measure $R_\alpha(.)$ when multiple subtrees of $T(\boldsymbol{x};\mathbf{a}_m)$ have the same minimum value, an additional condition is used. We select the smallest tree among these subtrees that satisfies condition (3.4). These smallest subtrees $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$ are called the smallest minimising subtrees. While there is at least one subtree of $T(\boldsymbol{x};\mathbf{a}_m)$ that minimises $R_\alpha(.)$ for every value of α (since there are only finitely many pruned subtrees of $T(\boldsymbol{x};\mathbf{a}_m)$), it is not clear if the additional condition (3.5) can be satisfied for all values of α. In other words, it is not clear whether there can exist two subtrees that minimize $R_\alpha(.)$ and are not subtrees of each other. However, Breiman et al. (1984, Theorem 10.7) proved that the additional condition (3.5) is indeed valid, meaning that there exists a smallest minimising subtree $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$ for every value of α.

When $\alpha = 0$, $R_\alpha(.)$ coincides with the in-sample deviance, the biggest tree is chosen because the complexity penalty term is essentially dropped. This leads to the selection of the tree $T(\boldsymbol{x};\mathbf{a}_m)$ which minimizes $R_\alpha(.)$. As the parameter α increases, the penalty for having a large tree increases, resulting in subtrees $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$ with fewer terminal nodes. For sufficiently large values of α, the subtree $T(\boldsymbol{x};\mathbf{a}_m(\alpha))$ consists of the root node only. Since tree $T(\boldsymbol{x};\mathbf{a}_m)$ has only a finite number of subtrees, the set of the smallest minimizing subtrees $\{T(\boldsymbol{x};\mathbf{a}_m(\alpha))\}_{\alpha\geq 0}$ contains only a finite number of subtrees of $T(\boldsymbol{x};\mathbf{a}_m)$ even if α varies continuously from zero to infinity. Breiman et al. (1984) established in Section 10.2 that an increasing sequence $0 = \alpha_0 < \alpha_1 < \ldots < \alpha_{\kappa_m} < \alpha_{\kappa_m+1} = \infty$ exists, such that, for all $k = 0, 1, \ldots, \kappa_m$

$$T(\boldsymbol{x};\mathbf{a}_m(\alpha)) = T(\boldsymbol{x};\mathbf{a}_m(\alpha_k)) \quad \text{for all } \alpha \in [\alpha_k, \alpha_{k+1})$$

with $T(\boldsymbol{x};\mathbf{a}_m(\alpha_0)) = T(\boldsymbol{x};\mathbf{a}_m)$ and $|T(\boldsymbol{x};\mathbf{a}_m(\alpha_{\kappa_m}))| = 1$. Moreover,

$$T(\boldsymbol{x};\mathbf{a}_m(\alpha_{\kappa_m})) \preceq T(\boldsymbol{x};\mathbf{a}_m(\alpha_{\kappa_m-1})) \preceq \ldots \preceq T(\boldsymbol{x};\mathbf{a}_m(\alpha_1)) \preceq T(\boldsymbol{x};\mathbf{a}_m(\alpha_0)).$$

The sequence of subtrees is found to be nested, indicating that any subtree $T(\boldsymbol{x};\mathbf{a}_m(\alpha_k))$ in the sequence $\{T(\boldsymbol{x};\mathbf{a}_m(\alpha_k))\}_{k=1,\ldots,\kappa_m}$ can be obtained by pruning the previous subtree $T(\boldsymbol{x};\mathbf{a}_m(\alpha_{k-1}))$. This sequence corresponds to the so-called minimal cost-complexity pruning procedure.

Remark 3.1. 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

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

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

$$R_\alpha(T(\boldsymbol{x};\mathbf{a}_m)) = \sum_{i=1}^{n} \nu_{mi} L\Big(\tilde{r}_{mi}, \exp\big(T(\boldsymbol{x}_i;\mathbf{a}_m)\big)\Big) + cp\, D_{root} |T(\boldsymbol{x};\mathbf{a}_m)|, \tag{3.6}$$

and the sequence $\{\alpha_k\}_{k=1,\ldots,\kappa_m}$ of regularization parameters defines the sequence $\{cp_k\}_{k=1,\ldots,\kappa_m}$ of cost-complexity parameters, with $cp_k = \dfrac{\alpha_k}{D_{root}}$. Note that the function `rpart` from the `R` package `rpart` used in Section 4 to produce the sequence of nested trees $\{T(\boldsymbol{x}; \mathbf{a}_m(\alpha_k))\}_{k=1,\ldots,\kappa_m}$ provides the cost-complexity parameters $\{cp_k\}_{k=1,\ldots,\kappa_m}$ rather than the regularization parameters $\{\alpha_k\}_{k=1,\ldots,\kappa_m}$.

3.3 ABT algorithm

Algorithm 3 generates decision trees of a fixed size across all iterations, where the size is determined by the interaction depth `ID` or equivalently by the number of terminal nodes J. We propose an adaptation of Algorithm 3 that allows the model complexity to adapt to the data structure not yet captured by the model fitted so far.

At iteration m, we first fit a tree with depth $\mathtt{D} = J - 1$, denoted as $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$, on the working training set $\mathcal{D}^{(m)} = \{(\nu_{mi}, \tilde{r}_{mi}, \boldsymbol{x}_i), i = 1, \ldots, n\}$. Note that the case $J = 1$ is not considered since it corresponds to fitting root node trees at each iteration (which is obviously useless once the initial score $\widehat{\text{score}_0}(\boldsymbol{x}) = \text{argmin}_\beta \sum_{i=1}^n \nu_i L(y_i, g^{-1}(\beta))$ has been estimated), so that we assume $J \geq 2$. Tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$ is the smallest tree guaranteeing that any tree with J terminal nodes solution of Step 2.2 in Algorithm 3 is a subtree of it. Any tree solution of Step 2.2 in Algorithm 3 can thus be obtained by pruning $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$. It is interesting to notice that the selected tree with J terminal nodes at iteration m obtained by pruning $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$ can be different from the tree that would have been obtained by using a greedy strategy typically used in the boosting trees algorithm. Here, we improve this greedy strategy by assessing the goodness of the splits by also looking at those deeper in the tree.

Then, we prune $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$ following the minimal cost-complexity pruning recalled in Section 3.2 up to get tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$ defined as the subtree of the nested sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ with J terminal nodes, that is,

$$|T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))| = J.$$

However, the nested sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ does not necessarily contain a tree with J terminal nodes. The mth tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$ in the score has thus J terminal nodes unless there is no tree in the sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ of that size, in which case tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$ has a smaller size and is such that

$$|T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*-1}))| > J \text{ and } |T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))| \leq J.$$

The ABT approach can thus be summarized as follows:

Algorithm 4 Adaptive Boosting Trees under Log-link.

1. Initialization :

Initialize $\widehat{\text{score}}_0(\boldsymbol{x})$ to be a constant. For instance:

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

2. Main procedure :

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

(2.1) Fit a regression tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$ with depth $\mathtt{D} = J - 1$ on the working training set

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

(2.2) Prune $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_0))$ following the minimal cost-complexity pruning up to get tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$ which is such that

$$|T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*-1}))| > J \text{ and } |T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))| \leq J.$$

(2.3) Update $\widehat{\text{score}}_m(\boldsymbol{x}) = \widehat{\text{score}}_{m-1}(\boldsymbol{x}) + \gamma T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$.

End for

3. **Output:**

$$\widehat{\mu}_{\mathcal{D}}^{\text{ABT}}(\boldsymbol{x}) = \exp\left(\widehat{\text{score}}_M(\boldsymbol{x})\right).$$

This adaptive boosting algorithm enables to reduce overfitting, as shown in the case study worked out in the next section, without requiring to perform cross validation at each step, which can be very time-consuming in practice.

4 Numerical illustrations

To end this paper, we illustrate the performance of the proposed ABT machine by presenting two case studies conducted using motor insurance data. The case studies both deal with motor liability claim numbers. However, the ABT algorithm works for any type of Tweedie response under log-link.

4.1 Case study 1

4.1.1 Data set

First, we consider the motor third-party liability insurance portfolio used in Denuit et al. (2020). This portfolio pertains to an insurance company operating in the EU which was observed over the course of one year. The portfolio comprises 160 944 insurance policies. Each policy, denoted as i, includes information on the number of claims Y_i filed by policyholder i,

Number of claims	Exposure-to-risk
0	126 499.7
1	15 160.4
2	1424.9
3	145.4
4	14.3
5	1.4
≥ 6	0

Table 2: Descriptive statistics for the number of claims.

the corresponding exposure-to-risk $e_i \leq 1$ (expressed in policy-year), and the eight features $\boldsymbol{X}_i = (X_{i1}, \ldots, X_{i8})$, namely

- X_{i1} = AgePh: policyholder's age;
- X_{i2} = AgeCar: age of the car;
- X_{i3} = Fuel: fuel of the car, with two categories (gas or diesel);
- X_{i4} = Split: splitting of the premium, with four categories (annually, semi-annually, quarterly or monthly);
- X_{i5} = Cover: extent of the coverage, with three categories (from compulsory third-party liability cover to comprehensive);
- X_{i6} = Gender: policyholder's gender, with two categories (female or male);
- X_{i7} = Use: use of the car, with two categories (private or professional);
- X_{i8} = PowerCat: the engine's power, with five categories.

Figure 2 displays the exposure-to-risk by category/value for each of the eight features and Table 5 shows the observed numbers of claims with corresponding exposures-to-risk. We refer the reader to Denuit et al. (2020) for a detailed presentation of the data set.

4.1.2 Results produced by the ABT machine

In this study, we adopt the Poisson deviance as the loss function when working with claim counts. Additionally, we utilize the log-link function, which is the canonical link in the Poisson case.

To produce each of the M constituent trees, we use the `R` functions `rpart` and `printcp` from the `R` package `rpart`. At iteration m, the trees of the sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ are first produced with the function `rpart`. Then, the right tree $T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_{k_m^*}))$ is selected from $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ by using the function `printcp` which provides the trees of the sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ together with their corresponding number of splits (and hence their number of terminal nodes) and cost-complexity parameters $cp_k = \dfrac{\alpha_k}{D_{root}}$. Notice

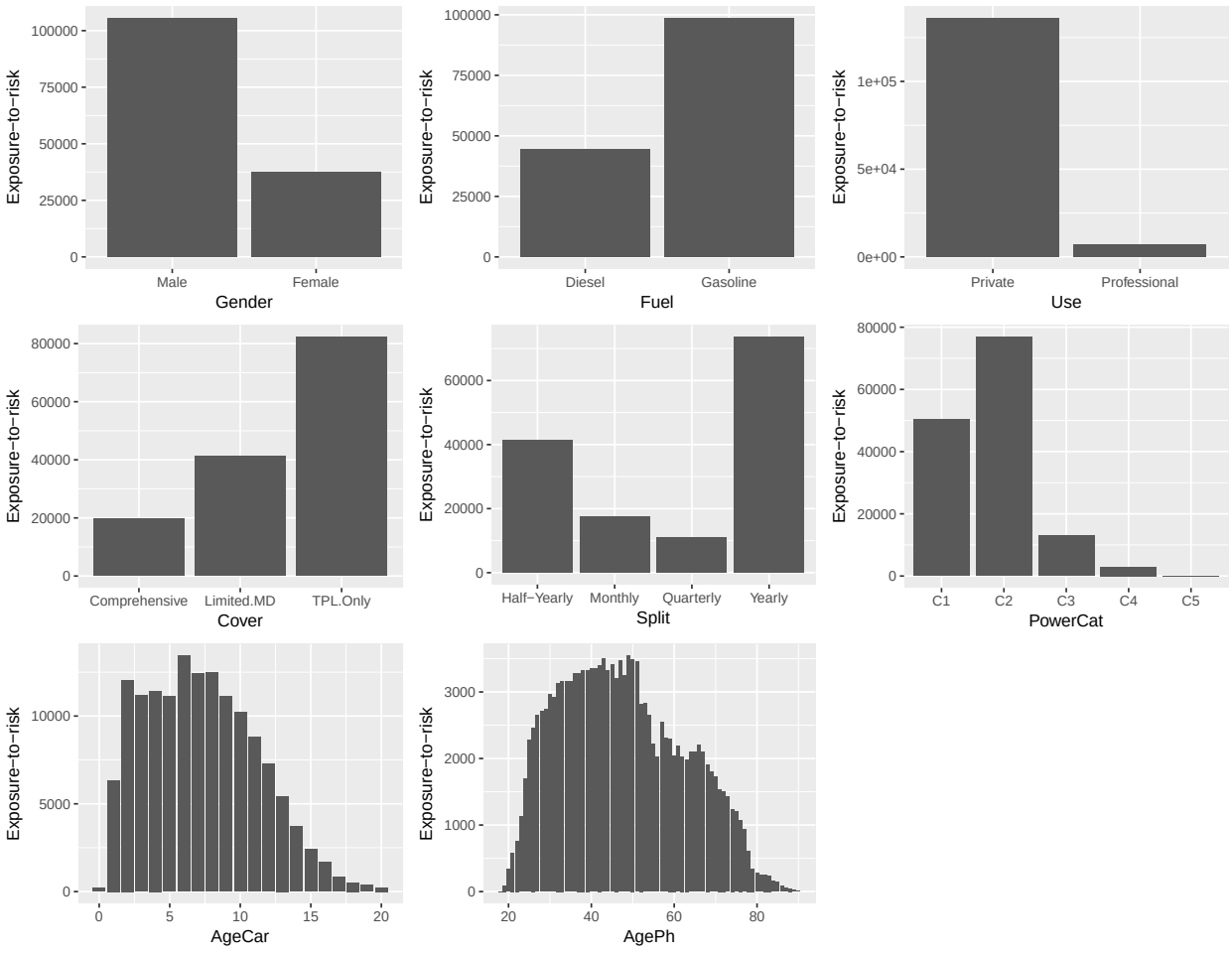


Figure 2: Levels/values of the features and corresponding exposures-to-risk.

that the function `rpart` automatically produces the sequence $\{T(\boldsymbol{x}; \widehat{\mathbf{a}}_m(\alpha_k))\}_{k=0,\ldots,\kappa_m}$ so that the computational time remains unaffected compared to boosting trees with depth $\texttt{D} = J-1$.

We run the ABT machine on a training set comprising 80% of the data set with $J = 20, 15, 10, 5$ and shrinkage coefficient $\gamma = 1\%$. The remaining 20% of the observations are used to compute the out-of-sample estimates of the generalization error. We also build gradient boosting trees (GBT) algorithm using the Poisson deviance as loss function and the log-link function on the training set with the `R` package `gbm`. The size of the trees is controlled by the interaction depth $\texttt{ID} = J-1$. Figure 3 displays in-sample and out-of-sample estimates of the generalization error with respect to the number of iterations for $J = 20, 15, 10, 5$, respectively. The corresponding number of terminal nodes of the constituent trees for the ABT machine are presented in Figure 4.

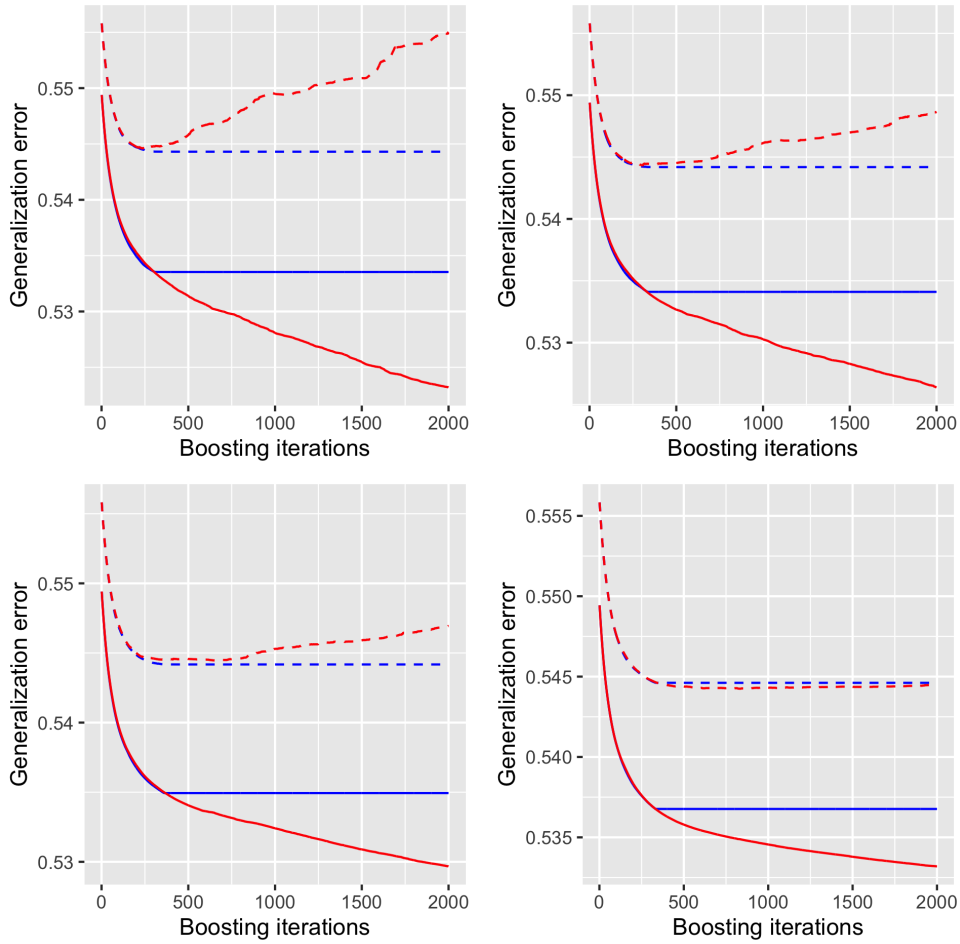


Figure 3: In-sample (solid blue line for ABT, solid red line for GBT) and out-of-sample (dotted blue line for ABT, dotted red line for GBT) estimates of the generalization error for $J = 20$ (upper left panel), $J = 15$ (upper right panel), $J = 10$ (bottom left panel), and $J = 5$ (bottom right panel).

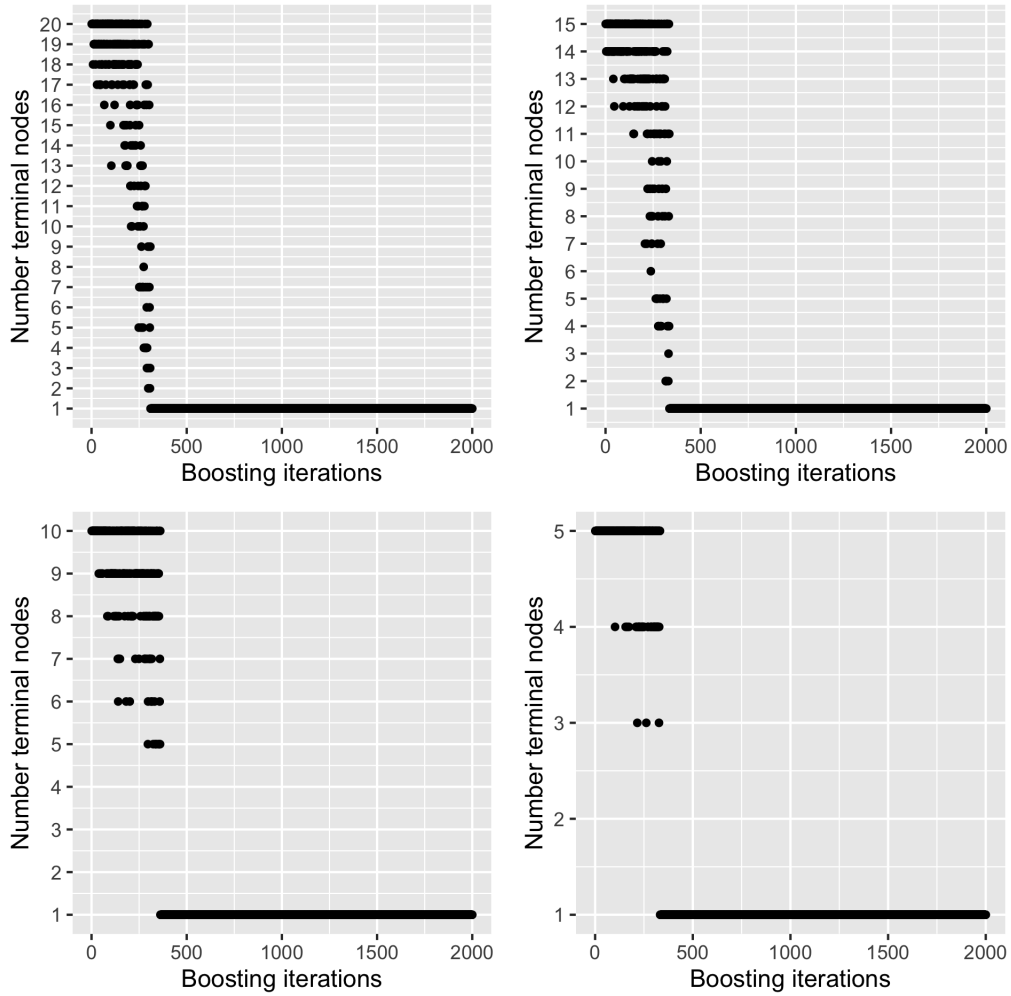


Figure 4: Number of terminal nodes of the constituent trees for the ABT machine with $J = 20$ (upper left panel), $J = 15$ (upper right panel), $J = 10$ (bottom left panel), and $J = 5$ (bottom right panel).

The proposed ABT machine has an advantage over GBT in that it avoids overfitting. Since the size of the trees does not adapt to the data with GBT, the model tends to overfit the training data, leading to a higher generalization error on the test sample. In contrast, the ABT machine adapts the size of the trees by gradually decreasing the number of terminal nodes to reach a root node tree. This progressive decrease is visible in Figure 4, which shows the corresponding number of terminal nodes for the ABT machine. To illustrate the progression leading to a single-node tree with the ABT machine, we closely examine the last five iterations for ABT with $J = 20$, namely iterations 305 to 309. At iteration 305, the `printcp` command of the `rpart` package yields the information summarized in Table 3. Specifically, as no tree within the list possesses exactly 19 splits, the algorithm selects here the tree with 5 splits, which corresponds to a cost-complexity parameter of 8.0927e-05. The resultant pruned tree is represented in Figure 5. Each rectangle represents a node of the tree. In each node, one can see the weighted mean of the working response in that node, the number of claims, the number of observations as well as the proportion of the training

CP	nsplit	rel error
1.0159e-04	0	1.00000
9.9330e-05	2	0.99980
8.0927e-05	5	0.99950
7.7784e-05	21	0.99818
...	...	...

Table 3: Trees produced at iteration 305.

set in the node. The darker the grey of a node, the higher the estimated expected working response in that node.

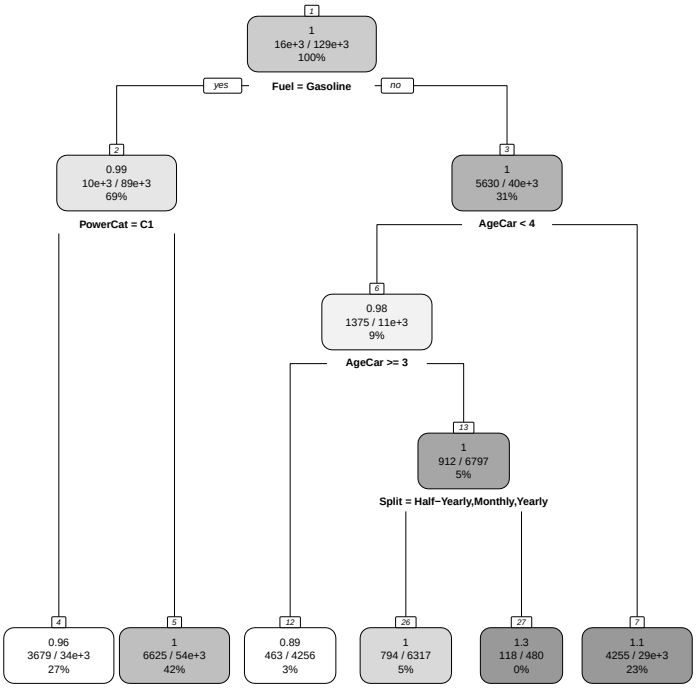


Figure 5: Tree at iteration 305.

The same results for iterations 306 to 309 are shown in Table 4 and Figure 6. The ABT algorithm finally stops at iteration 309 where a single-node tree is obtained for the first time.

CP	nsplit	rel error
9.3350e-05	0	1.00000
8.5623e-05	4	0.99962
7.9060e-05	22	0.99799
7.7079e-05	24	0.99784
...	...	...

CP	nsplit	rel error
9.3075e-05	0	1.00000
8.4815e-05	1	0.99991
7.9002e-05	31	0.99728
7.4593e-05	43	0.99630
...	...	...

CP	nsplit	rel error
9.2538e-05	0	1.00000
7.9623e-05	1	0.99991
7.9054e-05	2	0.99983
7.8639e-05	41	0.99657
...	...	...

CP	nsplit	rel error
9.1242e-05	0	1.00000
8.6894e-05	21	0.99797
8.6179e-05	23	0.99780
8.0010e-05	25	0.99762
...	...	...

Table 4: Trees produced at iteration 306 (top-left), 307 (top-right), 308 (bottom-left) and 309 (bottom-right).

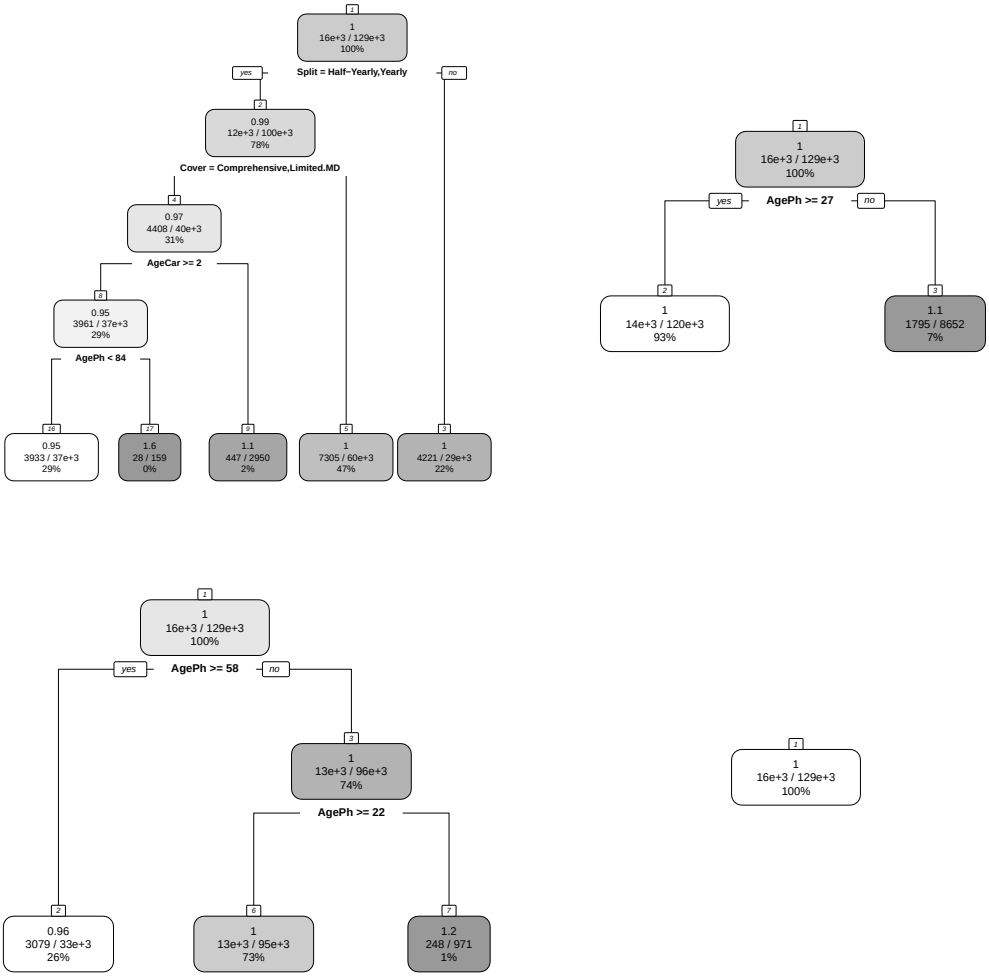


Figure 6: Trees at iteration 306 (top-left), 307 (top-right), 308 (bottom-left) and 309 (bottom-right).

The stopping criterion embedded in the ABT machine enables the avoidance of the time-consuming cross-validation step, thus saving computational time. Moreover, the ABT machine reduces overfitting compared to GBT, as illustrated in Figure 3, where in-sample and out-of-sample estimates of the generalization error with respect to the number of iterations are presented for both methods. While the size of the first trees for ABT is equal to J, it decreases on average with the number of iterations.

The proposed ABT machine is a promising alternative to GBT due to its similar or slightly better performance in terms of out-of-sample deviance or generalization error. One advantage of ABT is that it does not require cross-validation and operates directly on the response variables rather than gradients, making it a more efficient approach.

The only exception is with $J = 5$, where the proposed learning procedure stops too quickly. The larger the value of J, the more the selection of the root node tree in the sequence reveals that there is no more information to capture in the training set. This is because there is no larger tree with at most J terminal nodes that is more relevant than the root node tree. It is more likely that the first root node tree in the ensemble appears early for small J than for larger ones, which can be an issue since the learning procedure automatically stops. A way to overcome this issue is proposed next.

4.1.3 ABT with bagging fraction

The stopping criterion embedded within the ABT machine may sometimes result in stopping too early, especially for small J. To prevent this from occurring, it may be helpful to allow for a bagging fraction, exactly as with standard boosting procedure. The bagging fraction β represents the fraction of observations randomly selected from the training set to fit the next tree in the expansion. Therefore, $\beta\%$ of the training sample is used at each iteration with a common value of 50% or higher (if the training sample is small). For $\beta = 100\%$, we recover the results obtained in the preceding section. Randomizing the data used to fit the new tree then avoids that ABT gets trapped too early in the single-root node tree when the number of terminal nodes J is small.

Figure 7 displays the results obtained with $\beta \in \{50\%, 70\%, 90\%\}$ for the different values of J. Results for $\beta = 100\%$ are those displayed in Figure 3. For any value of J, the larger the bagging fraction β, the more ABT prevents overfitting. When $\beta = 100\%$, ABT stops once a root-node tree is reached. When $\beta < 100\%$, a root-node tree can be followed by a larger tree since the data set changes. In fact, it is enough to consider β slightly smaller than one, which almost avoids overfitting while outperforming GBT, as illustrated in Figure 8 where we consider $\beta = 99\%$. It is worth mentioning that when $\beta < 100\%$, the stopping criterion is not embedded within the ABT machine anymore, so that an out-of-sample estimate of the generalization error must be used as in classical boosting or gradient boosting. As shown in Figures 7 and 8, the ABT machine with $\beta < 100\%$ is much less prone to overfitting than the classical gradient boosting widely used by practitioners.

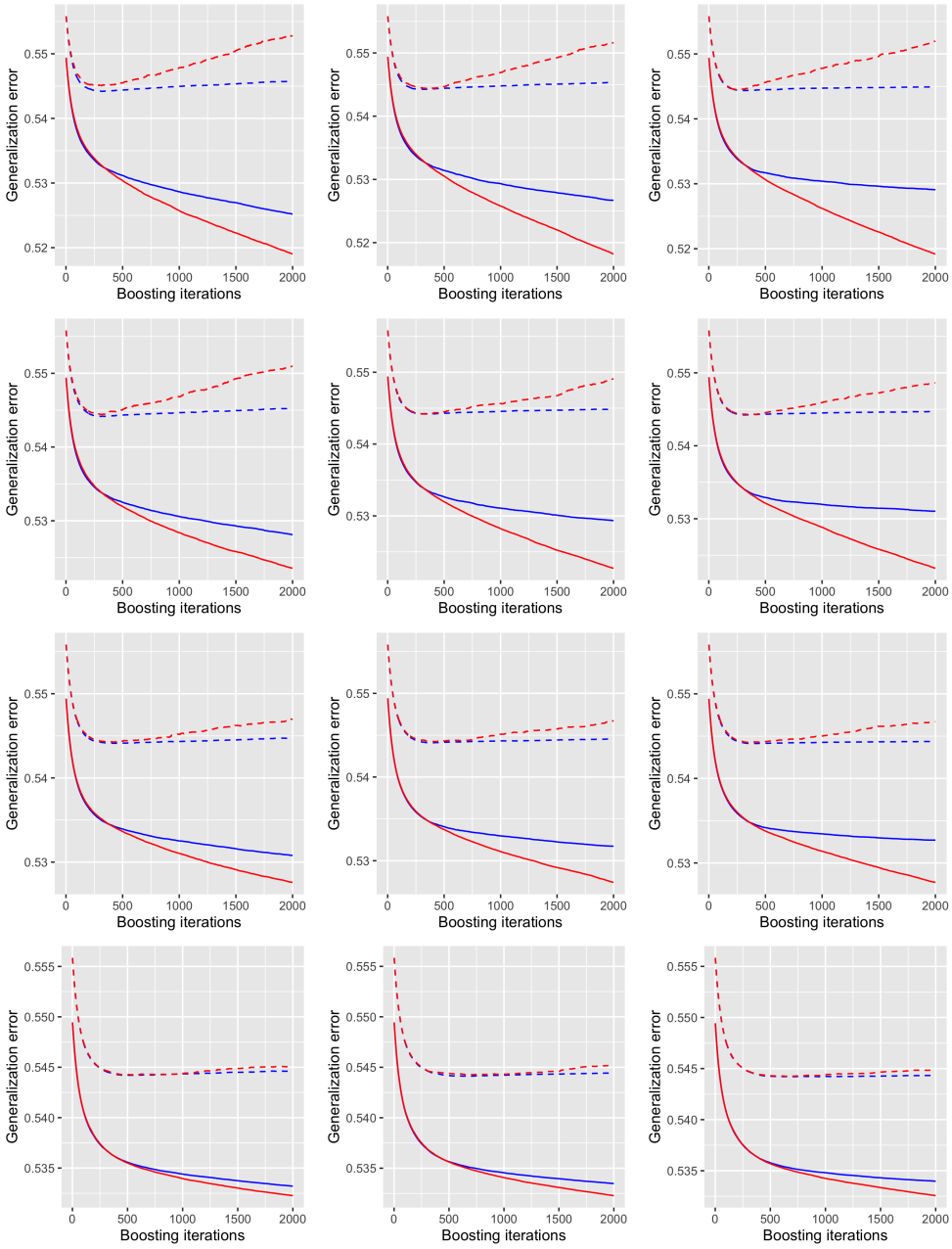


Figure 7: In-sample (solid blue line for ABT, solid red line for GBT) and out-of-sample (dotted blue line for ABT, dotted red line for GBT) estimates of the generalization error. From up to bottom: $J = 20$, $J = 15$, $J = 10$, and $J = 5$ (bottom panel). From left to right: $\beta = 50\%$, $\beta = 70\%$, and $\beta = 90\%$.

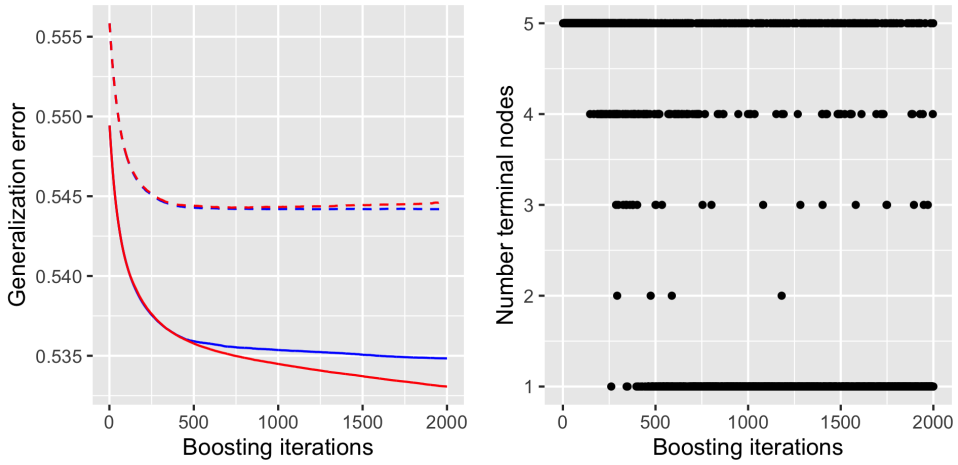


Figure 8: In-sample (solid blue line for ABT, solid red line for GBT) and out-of-sample (dotted blue line for ABT, dotted red line for GBT) estimates of the generalization error for $J = 5$ and $\beta = 99\%$ (left panel) and number of terminal nodes of the constituent trees for ABT (right panel).

The data set considered in this first case study is actually quite simple in the sense that six of the eight features are categorical variables with at most four categories. The next case study is more realistic for practitioners, the data set comprising features with much more categories. Considering large trees at the early stage of the ABT algorithm makes more sense in the next case study since the number of possible splits at each node of a tree is much higher in the next case study.

4.2 Case study 2

4.2.1 Data set

Finally, we consider a data set corresponding to a French motor third-party liability insurance portfolio available in the `CASdatasets` package in `R`. Specifically, we look at the data set `freMTPL2freq` which contains 678 013 insurance policies. For each policy i, the data set describes the numbers of claims Y_i filed by policyholder i, the corresponding exposure-to-risk $e_i \leq 1$ and the following features $\mathbf{X}_i = (X_{i1}, \ldots, X_{i8})$:

- X_{i1} = DrivAge: policyholder's age;
- X_{i2} = VehAge: age of the car in years;
- X_{i3} = VehPower: power of the car, with 12 categories;
- X_{i4} = VehBrand: car brand, with 11 categories;
- X_{i5} = VehGas : Gas or diesel, with 2 categories;
- X_{i6} = Area: area code, with 6 categories;

- X_{i7} = Region: region in France, with 21 categories;

- X_{i8} = Density: density of inhabitants per km2 in the city of the living place of the driver.

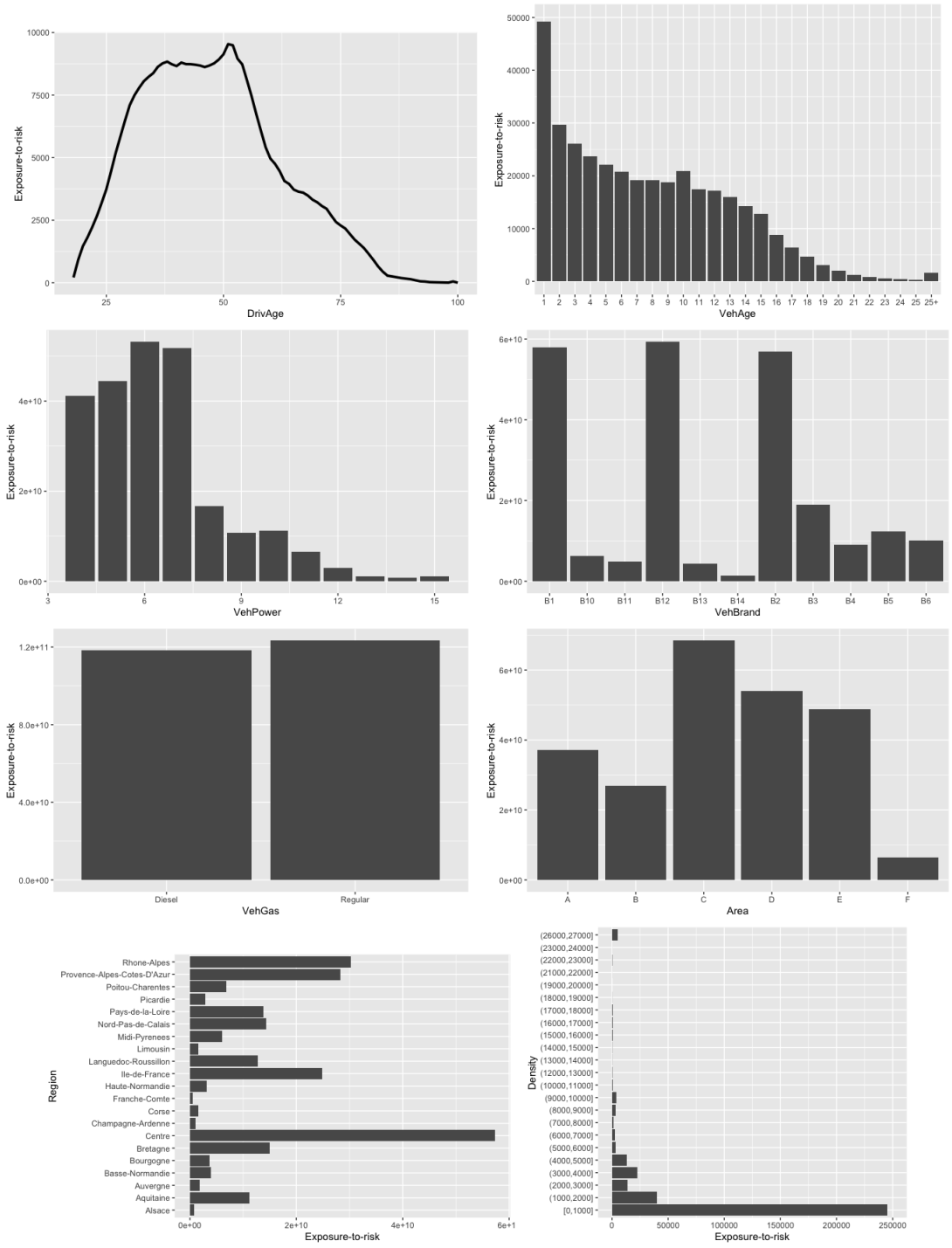


Figure 9: Levels/values of the features and corresponding exposures-to-risk.

Number of claims	Exposure-to-risk
0	335 311.2
1	20 615.6
2	1 150.1
3	52.8
4	3.0
5	1.1
≥ 6	2.3

Table 5: Descriptive statistics for the number of claims.

Figure 7 displays the exposure-to-risk by category/value for each of the eight features and Table 3 shows the observed numbers of claims with corresponding exposures-to-risk. This data set has been used with different statistical and machine learning techniques by several authors, including Noll et al. (2018). We refer the reader to the latter reference for an accurate description of the data set.

4.2.2 Results produced by the ABT machine

Firstly, we train the ABT machine using 80% of the data set with different values of $\texttt{ID} = J - 1 = 15, 20, 25, 30$, along with a shrinkage coefficient $\gamma = 1\%$. The remaining 20% of the observations was used to compute the out-of-sample estimates of the generalization error. We begin by fitting large trees in the early stages ($\texttt{ID} = 15, 20, 25, 30$), which then gradually simplify until they reach the single-node tree where the ABT machine stops. Next, we build GBT on the training set with the `R` package `gbm` using the Poisson deviance as loss function and the log-link function. The size of the trees is controlled by the interaction depth $\texttt{ID} = J - 1$. We consider $\texttt{ID} = 1, 2, 3, 4, 5, 6$ with GBT, which are typical values for GBT, and we use a bag fraction of 50%, which is also a typical value for the bag fraction with GBT.

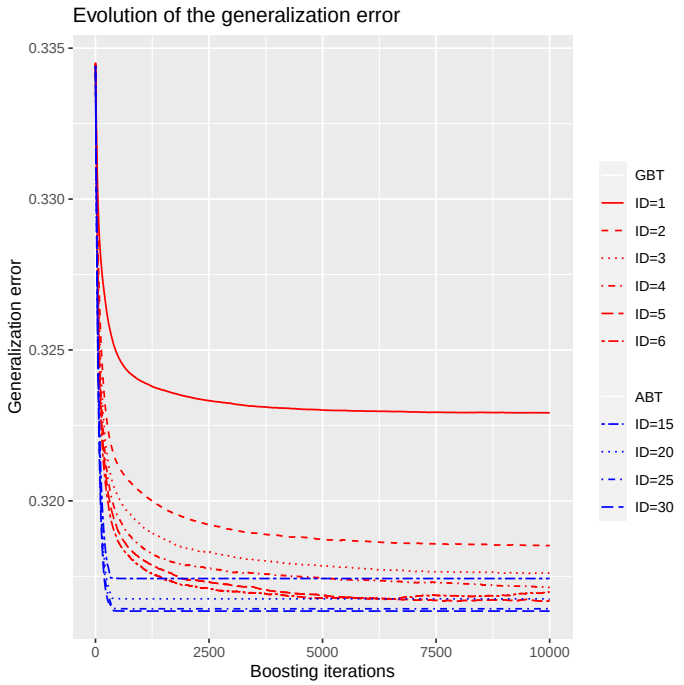


Figure 10: Out-of-sample estimates of the generalization error for GBT in red ($\texttt{ID} = 1, 2, 3, 4, 5, 6$) and ABT in blue ($\texttt{ID} = 15, 20, 25, 30$).

The results are depicted in Figure 10. Larger initial trees for ABT lead to lower out-of-sample estimates of the generalization error (0.3174263 for $\texttt{ID} = 15$, 0.3167534 for $\texttt{ID} = 20$, 0.3164238 for $\texttt{ID} = 25$ and 0.3163478 for $\texttt{ID} = 30$). ABT with $\texttt{ID} = 25$ and ABT with $\texttt{ID} = 30$ perform quite similarly in terms of predictive accuracy. Increasing the interaction depth by 5 again (and hence considering ABT with $\texttt{ID} = 35$) produces similar results than the ones obtained with $\texttt{ID} = 30$ so that we did not show the errors in Figure 10 for $\texttt{ID} = 35$. Using ABT with sufficiently large trees at the early stage of the algorithm appears to be effective in this example, the trees progressively simplify during the algorithm. The out-of sample estimate of the generalization error for the best ABT model with $\texttt{ID} = 30$ is significantly lower than the error for the best GBM with $\texttt{ID} = 5$. Moreover, we can see that the ABT method with large initial trees requires significantly fewer trees than the GBT approach. Specifically, the error stabilizes after more or less 700 trees for ABT with $\texttt{ID} = 30$, whereas the lowest error for GBM with $\texttt{ID} = 5$ is achieved after more or less 8000 trees. These findings suggest that our ABT method shows promise in terms of both prediction accuracy and computation time.

5 Conclusion

To simplify numerical computations, boosting is often applied to gradients of the loss function. Instead of minimizing the deviance associated with the responses, gradient boosting applies a least-squares principle on its gradients, which has made it a popular choice among data analysts. Open-source software packages such as the Extreme Gradient Boosting (XG-

Boost) algorithm have implemented highly effective boosting algorithms.

However, boosting is not limited to gradients and can be regarded as an iteratively re-weighted or re-offsetted procedure applied to the original data. Hainaut et al. (2022) have shown that it is often unnecessary to boost gradients in insurance applications and that boosting can be performed directly with responses under Tweedie deviance and log-link.

To prevent overfitting, cross-validation is commonly used to stop the boosting algorithm when prediction capabilities begin to deteriorate. Early stopping is crucial in ensuring a sparse model with optimal performance on new data. However, cross-validation can be computationally expensive.

This paper presents a new procedure called "ABT" for adaptive boosting trees. The idea behind ABT 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 progressively simplify until the ABT machine stops at a single-node tree. The stopping criterion is built within the ABT algorithm, eliminating the need for computationally-intensive cross-validation.

Using two datasets from motor insurance, we have shown that the ABT machine allows for better adaptation to the data at each iteration, improving the prediction and reducing overfitting. The numerical analysis results clearly demonstrate the effectiveness of the algorithm in terms of predictive performance.

In conclusion, our paper presents an easily interpretable and implementable boosting algorithm that integrates the stopping criterion within the algorithm. The combination of results from Hainaut et al. (2022) and the pruning process offers a better prediction and prevents overfitting.

References

- Breiman, L., Friedman, J. H. (1988). Tree-structured classification via generalized discriminant analysis. Journal of the American Statistical Association 83(403), 725-727.

- Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J. (1984). Classification and Regression Trees. Wadsworth Statistics/Probability Series.

- Chen, H., Tino, P., Yao, X. (2009). Predictive ensemble pruning by expectation propagation. IEEE Transactions on Knowledge and Data Engineering, 21(7), 999-1013.

- Costa, V.G., Pedreira, C.E. (2022). Recent advances in decision trees: an updated survey. Artificial Intelligence Review.

- 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. (2019b). Effective Statistical Learning Methods for Actuaries III: Neural Networks 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.

- Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics 29(5), 1189-1232.

- Garcia Leiva, R., Fernandez Anta, A., Mancuso, V. (2019). A novel hyperparameter-free approach to decision tree construction that avoids overfitting by design. IEEE Access.

- 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 2022(10), 841-866.

- 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(2), 255-285.

- Hernandez-Lobato, D., Hernandez-Lobato, J. M., Ruiz-Torrubiano, R., Valle, A. (2006). Pruning adaptive boosting ensembles by means of a genetic algorithm. Intelligent Data Engineering and Automated Learning. Proceedings 7, 322-329. Springer Berlin Heidelberg.

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

- Margineantu, D. D., Dietterich, T. G. (1997). Pruning adaptive boosting. In International Conference on Machine Learning 97, 211-218.

- Noll, A., Salzmann, R., Wüthrich, M. (2018). Case study: French motor third-party liability claims. Available at SSRN: https://ssrn.com/abstract=3164764.

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

- Quinlan J.R. (1986). Induction of decision trees. Machine Learning 1, 81-106.

- Tamon, C.,Xiang, J. (2000). On the boosting pruning problem. European Conference on Machine Learning, CProceedings 11, 404-412. Springer Berlin Heidelberg.

- Thompson, S. (1999). Pruning boosted classifiers with a real valued genetic algorithm. Knowledge-Based Systems, 12(5-6), 277-284.

- Vidal, T., Schiffer, M. (2020). Born-again tree ensembles. International conference on machine learning 9743-9753.

- Windeatt, T., Ardeshir, G. (2001). An empirical comparison of pruning methods for ensemble classifiers. Advances in Intelligent Data Analysis: 4th International Conference. Springer Berlin Heidelberg.

- Wu, C.C. , Chen, Y.L., Liu, Y.H. (2016). Decision tree induction with a constrained number of leaf nodes. Applied Intelligence 45(3), 673-685.

- 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.