Machine Learning (2019) 108:971–992
https://doi.org/10.1007/s10994-019-05787-1

Accelerated gradient boosting

G. Biau[1] · **B. Cadre**[2] · **L. Rouvière**[2]

Received: 5 March 2018 / Accepted: 17 January 2019 / Published online: 4 February 2019

Abstract
Gradient tree boosting is a prediction algorithm that sequentially produces a model in the form of linear combinations of decision trees, by solving an infinite-dimensional optimization problem. We combine gradient boosting and Nesterov's accelerated descent to design a new algorithm, which we call AGB (for Accelerated Gradient Boosting). Substantial numerical evidence is provided on both synthetic and real-life data sets to assess the excellent performance of the method in a large variety of prediction problems. It is empirically shown that AGB is less sensitive to the shrinkage parameter and outputs predictors that are considerably more sparse in the number of trees, while retaining the exceptional performance of gradient boosting.

Keywords Gradient boosting · Nesterov's acceleration · Trees

1 Introduction

Gradient boosting (Friedman et al. 2000; Friedman 2001, 2002) is a learning procedure that combines the outputs of many simple predictors in order to produce a powerful committee with performances improved over the single members. The approach is typically used with decision trees of a fixed size as base learners, and, in this context, is called gradient tree boosting. This machine learning method is widely recognized for providing state-of-the-art results on several challenging data sets, as pointed out by Chen and Guestrin (2016). Boosted decision trees are generally regarded as one of the best off-the-shell prediction algorithms we have today, with performance at the level of the Lasso (Tibshirani 1996) and random forests (Breiman 2001), to name only two competitors.

Editor: Byron Wallace.

✉ G. Biau
gerard.biau@upmc.fr

B. Cadre
benoit.cadre@ens-rennes.fr

L. Rouvière
laurent.rouviere@univ-rennes2.fr

[1] CNRS, LPSM, Sorbonne Université, Paris, France

[2] CNRS, IRMAR, Univ Rennes, Rennes, France

Gradient boosting originates in Freund and Schapire's work (Schapire 1990; Freund 1995; Freund and Schapire 1996, 1997) on weighted iterative classification. It was complemented by several analyses by Breiman (1997, 1998, 1999, 2000, 2004), who made the fundamental observation that Freund and Schapire's AdaBoost is in fact a gradient-descent-type algorithm in a function space, thus identifying boosting at the frontier of numerical optimization and statistical estimation. Explicit regression and classification boosting algorithms were subsequently developed by Friedman (2001, 2002), who coined the name "gradient boosting" and paid a special attention to the case where the individual components are decision trees. Overall, this functional view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification (e.g., Blanchard et al. 2003; Bühlmann and Yu 2003; Lugosi and Vayatis 2004; Zhang and Yu 2005; Bickel et al. 2006; Bühlmann and Hothorn 2007).

In a different direction, the pressing demand of the machine learning community to build accurate prediction mechanisms from massive amounts of high-dimensional data has greatly promoted the theory and practice of accelerated first-order schemes. In this respect, one of the most effective approaches among first-order optimization techniques is the so-called Nesterov's accelerated gradient descent (Nesterov 1983). In a nutshell, if we are interested in minimizing some convex function $g(x)$ over $\mathbb{R}^d$, then Nesterov's descent may take the following form (Beck and Teboulle 2009; Bubeck 2015): starting with $x_0 = y_0$, inductively define

$$\begin{aligned} x_{t+1} &= y_t - w\nabla g(y_t) \\ y_{t+1} &= (1-\gamma_t)x_{t+1} + \gamma_t x_t, \end{aligned} \tag{1}$$

where w is the step size,

$$\lambda_0 = 0, \quad \lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}, \quad \text{and} \quad \gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}.$$

In other words, Nesterov's descent performs a simple step of gradient to go from y_t to x_{t+1}, and then it slides it a little bit further than x_{t+1} in the direction given by the previous point x_t. As acknowledged by Bubeck (2013), the intuition behind the algorithm is quite difficult to grasp. Nonetheless, Nesterov's accelerated gradient descent is an optimal method for smooth convex optimization: the sequence $(x_t)_t$ defined in (1) recovers the minimum of g at a rate of order $1/t^2$ (e.g., Bubeck 2015, Theorem 3.19), in contrast to vanilla gradient descent methods, which have the same computational complexity but can only achieve a rate in $1/t$. Throughout the document, we will use form (1) for Nesterov's descent, keeping in mind that other choices are possible but less adapted to the boosting context, as we will see later. Since the introduction of Nesterov's scheme, there has been much work on first-order accelerated methods (see, e.g., Nesterov 2004, 2005, 2013; Su et al. 2016, for theoretical developments, and Tseng 2008, for a unified analysis of these ideas). Notable applications can be found in sparse linear regression (Beck and Teboulle 2009), compressed sensing (Becker et al. 2011), distributed gradient descent (Qu and Li 2016), and deep and recurrent neural networks (Sutskever et al. 2013).

In this article, we present `AGB` (for `A`ccelerated `G`radient `B`oosting), a new tree boosting algorithm that incorporates Nesterov's mechanism (1) into Friedman's original procedure (Friedman 2001). Substantial numerical evidence is provided on both synthetic and real-life data sets to assess the excellent performance of our method in a large variety of

prediction problems. The striking feature of AGB is that it enjoys the merits of both approaches:

(i) Its predictive performance is comparable to that of standard gradient tree boosting;
(ii) It takes advantage of the accelerated descent to output models which are remarkably much more sparse in their number of components.

Item (ii) is of course a decisive advantage for large-scale learning, when time and storage issues matter. To make the concept clear, we show in Fig. 1 typical test error results by number of iterations and shrinkage (step size), both for the standard (top) and the accelerated (bottom) algorithms. As is often the case with gradient boosting, smaller values of the shrinkage parameter require a larger number of trees for the optimal model, when the test error is at its minimum. However, if both approaches yield similar results in terms of prediction, we see that the optimal number of iterations is at least one order of magnitude smaller for AGB.

The paper is organized as follows. In Sect. 2, we briefly recall the mathematical/statistical context of gradient boosting, and present the principle of the AGB algorithm. Section 3 is devoted to analyzing the results of a battery of experiments on synthetic and real-life data sets. We offer an extensive comparison between the performance of Friedman's gradient tree boosting and AGB, with a special emphasis put on the influence of the learning rate on the size of the optimal models. Section 4 summarizes the main results and discusses research directions for future work. The code used for the simulations and the figures is available at https://github.com/lrouviere/AGB.

2 (Accelerated) gradient boosting

2.1 Gradient boosting at a glance

Let $\mathscr{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a sample of i.i.d. observations, taking values in $\mathbb{R}^d \times \mathscr{Y}$. Throughout, $\mathscr{Y} \subset \mathbb{R}$ is either a finite set of labels (for classification) or a subset of $\mathbb{R}$ (for regression). The learning task is to construct a predictor $F : \mathbb{R}^d \to \mathbb{R}$ that assigns a response to each possible value of the independent random observation X_{n+1}. In the context of gradient boosting, this general problem is addressed by considering a class $\mathscr{F}$ of elementary functions $f : \mathbb{R}^d \to \mathbb{R}$ (called the weak or base learners), and by minimizing some empirical risk functional

$$C_n(F) = \frac{1}{n}\sum_{i=1}^{n} \psi(F(X_i), Y_i) \tag{2}$$

over the linear combinations of functions in $\mathscr{F}$. Thus, we are looking for an additive solution of the form $F_n = \sum_{t=0}^{T} \alpha_t f_t$, where $(\alpha_0, \ldots, \alpha_T) \in \mathbb{R}^{T+1}$ and each component f_t is picked in the base class $\mathscr{F}$.

The function $\psi : \mathbb{R} \times \mathscr{Y} \to \mathbb{R}_+$ is called the loss. It is assumed to be convex and continuously differentiable in its first argument. It measures the cost incurred by predicting $F(X_i)$ when the answer is Y_i. For example, in the least squares regression problem, $\psi(x, y) = (y - x)^2$, and

$$C_n(F) = \frac{1}{n}\sum_{i=1}^{n} (Y_i - F(X_i))^2.$$

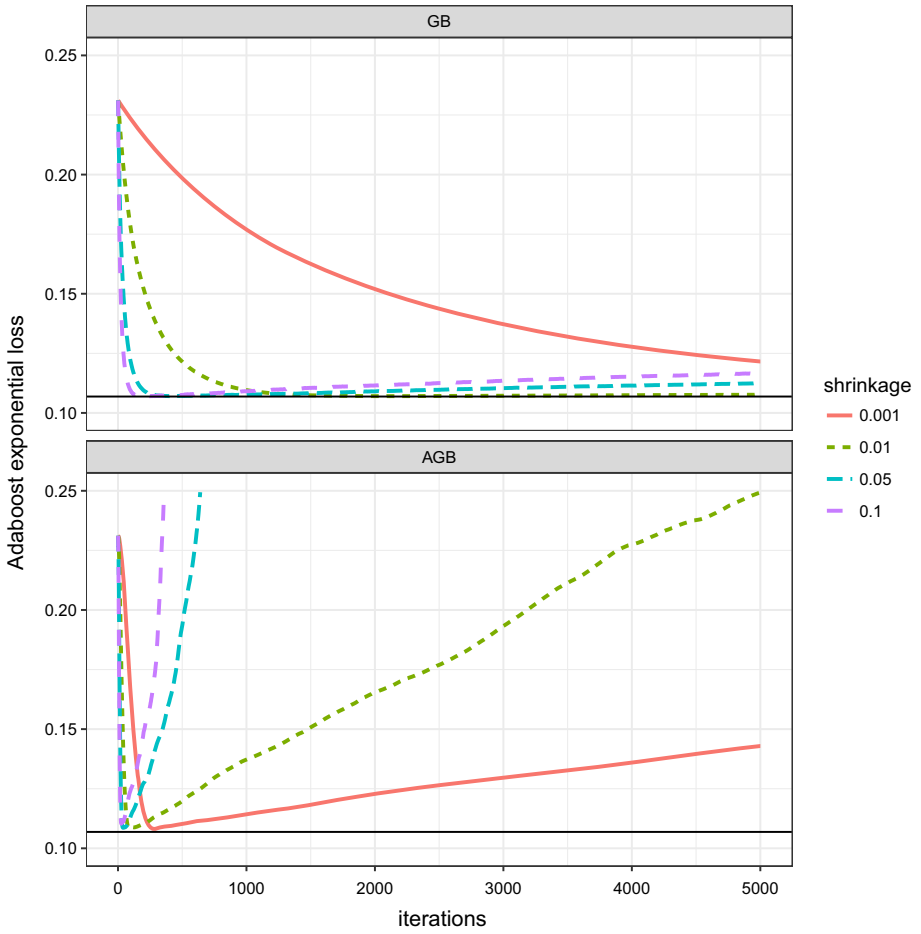

Fig. 1 Adaboost exponential loss (estimated on a test data set) by number of iterations for standard gradient boosting (top) and AGB (bottom). The data are generated according to Model 5 with $n = 5000$ observations (see Sect 3.1 for Model 5)

In the ± 1-classification problem, the final classification rule is $+1$ if $F(x) > 0$ and -1 otherwise. In this context, two classical losses are $\psi(x, y) = e^{-yx}$ (Adaboost exponential loss) and $\psi(x, y) = \log_2(1 + e^{-yx})$ (logit loss, where $\log_2$ is the binary logarithm).

In the present document, we take for $\mathscr{F}$ the collection of all binary decision trees in $\mathbb{R}^d$ using axis parallel cuts with k (small) terminal nodes (or leaves). Let $\mathbb{1}_A$ denote the indicator function of a set A. Then each $f \in \mathscr{F}$ takes the form $f = \sum_{j=1}^{k} \beta_j \mathbb{1}_{A_j}$, where $(\beta_1, \ldots, \beta_k) \in \mathbb{R}^k$ and $\{A_1, \ldots, A_k\}$ is a tree-structured partition of $\mathbb{R}^d$ (Devroye et al. 1996, Chapter 20). An example of regression tree fitted with the R package `rpart.plot` with $k = 3$ leaves in dimension $d = 2$ is shown in Fig. 2.

Let us get back to the minimization problem (2) and denote by $\text{lin}(\mathscr{F})$ the set of all linear combinations of functions in $\mathscr{F}$, our basic collection of trees. So, each $F \in \text{lin}(\mathscr{F})$ is an additive association of trees, of the form $F_T = \sum_{t=0}^{T} \alpha_t f_t$. Finding the infimum of the functional C_n over $\text{lin}(\mathscr{F})$ is a challenging infinite-dimensional optimization problem,

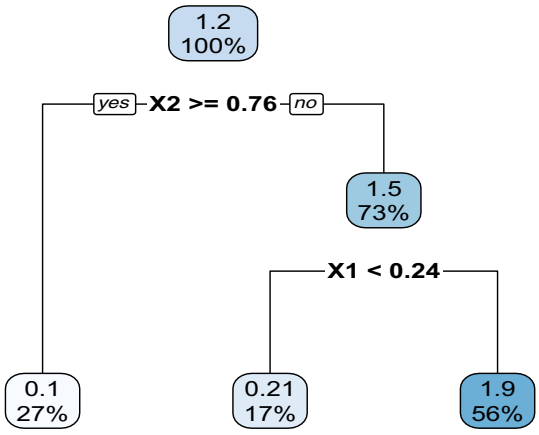

Fig. 2 A regression tree in dimension $d = 2$ with $k = 3$ leaves

which requires an algorithm. This is where gradient boosting comes into play by sequentially constructing a linear combination of trees, adding one new component at each step. This algorithm rests upon a sort of functional gradient descent, which we briefly describe in the next paragraph. We do not go too much into the mathematical details, and refer to Mason et al. (1999, 2000) and Biau and Cadre (2017) for a thorough analysis of the mathematical forces in action.

Let δ_z denote the Dirac measure at z, and let $\mu_n = (1/n)\sum_{i=1}^n \delta_{(X_i,Y_i)}$ be the empirical measure associated with the sample $\mathscr{D}_n$. Clearly,

$$C_n(F) = \mathbb{E}\psi(F(X), Y),$$

where (X, Y) denotes a random pair with distribution μ_n and the symbol $\mathbb{E}$ denotes the expectation with respect to μ_n. The theoretical version of C_n is

$$C(F) = \mathbb{E}\psi(F(X_1), Y_1),$$

where now the expectation is taken with respect to the distribution of (X_1, Y_1). To unify the notation, we let for now (X, Y) be a generic pair of random variables with distribution $\mu_{X,Y}$, keeping in mind that $\mu_{X,Y}$ may be the distribution of (X_1, Y_1) (theoretical risk) or the empirical measure μ_n (empirical risk). We let μ_X be the distribution of X, $L^2(\mu_X)$ the vector space of all measurable functions $f : \mathbb{R}^d \to \mathbb{R}$ such that $\int |f|^2 \mathrm{d}\mu_X < \infty$, and denote by $\langle \cdot, \cdot \rangle_{\mu_X}$ and $\| \cdot \|_{\mu_X}$ the corresponding norm and scalar product. Thus, with this notation, our problem is to minimize the quantity

$$C(F) = \mathbb{E}\psi(F(X), Y)$$

over the linear combinations of functions in the given subset $\mathscr{F}$ of $L^2(\mu_X)$.

As we have seen earlier, it is assumed that, for each $y \in \mathscr{Y}$, the function $\psi(\cdot, y)$ is convex and continuously differentiable. In this setting, the functional C is differentiable at any $F \in L^2(\mu_X)$ in the direction $G \in L^2(\mu_X)$, with differential

$$dC(F; G) = \langle \nabla C(F), G \rangle_{\mu_X},$$

where $\nabla C(F)(x) := \int \partial_x \psi(F(x), y) \mu_{Y|X=x}(\mathrm{d}y)$ is called the gradient of C at F (the symbol ∂_x means partial derivative with respect to the first component and $\mu_{Y|X}$ is the conditional distribution of Y given X).

Suppose now that we have at step t a function $F_t \in \mathrm{lin}(\mathscr{F})$ and wish to find a new $f_{t+1} \in \mathscr{F}$ to add to F_t so that the risk $C(F_t + w f_{t+1})$ decreases at most, for some small value of w. Viewed in function space terms, we are looking for the direction $f_{t+1} \in \mathscr{F}$ such that $C(F_t + w f_{t+1})$ most rapidly decreases. One approach might be to take $f_{t+1}(\cdot) = -\nabla C(F_t)(\cdot)$, the opposite of the gradient of C at F_t (this is a function over $\mathbb{R}^d$), and update according to

$$F_{t+1} = F_t - w \nabla C(F_t). \tag{3}$$

However, since we are restricted to pick our new function in $\mathscr{F}$, this will in general not be a possible choice. The strategy is to choose the new f_{t+1} by a least squares approximation of the function $-\nabla C(F_t)(\cdot)$, i.e., to take

$$f_{t+1} \in \arg\min_{f \in \mathscr{F}} \| - \nabla C(F_t) - f \|^2_{\mu_X}. \tag{4}$$

In the empirical setting, where $C \equiv C_n$ and μ_X is the empirical measure with respect to $X_1, \ldots, X_n$, it is easy to see that the approximation (4) takes the form

$$f_{t+1} \in \arg\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} \big(- \nabla C_n(F_t)(X_i) - f(X_i) \big)^2, \tag{5}$$

where $\nabla C_n(F_t)(X_i) = \partial_x \psi(F_t(X_i), Y_i)$. For example, when $\psi(x, y) = (y - x)^2/2$, then $-\nabla C_n(F_t)(X_i) = Y_i - F_t(X_i)$, and the algorithm simply fits f_{t+1} to the residuals $Y_i - F_t(X_i)$ at step t.

Iteration (3) together with approximation (5) form the core of the gradient boosting principle. After T iterations, the method outputs an additive expansion of the form $F_T = \sum_{t=0}^{T} \alpha_t f_t$, where $(\alpha_0, \ldots, \alpha_T)$ is a sequence of weights and $(f_0, \ldots, f_T)$ is a sequence of trees in $\mathscr{F}$. The parameter w in (3) is the step size of the gradient descent. It is eventually allowed to change at every iteration and should be carefully chosen for convergence guarantees, as shown for example in Biau and Cadre (2017). Importantly, from a practical point of view, finding the optimum in (5) is a non-trivial computational problem, which necessitates a strategy. In Friedman's gradient tree boosting algorithm (Friedman 2001), one uses a CART-style top-down recursive partitioning to compute the minimum at each iteration, together with several regularization techniques to reduce the eventual overfitting. Some of these features are incorporated in our accelerated version, which is presented in the next section.

2.2 The AGB algorithm

The pseudo-code of AGB is summarized in the table below.

AGB algorithm

1: **Require** The data set $\mathscr{D}_n$, $T \geq 1$ (number of iterations), $k \geq 1$ (number of terminal nodes in the trees), $0 < \nu < 1$ (shrinkage parameter).

2: **Initialize** $F_0 = G_0 \in \arg\min_z \sum_{i=1}^n \psi(z, Y_i)$, $\lambda_0 = 0$, $\gamma_0 = 1$.

3: **for** $t = 0$ to $(T-1)$ **do**

4: For $i = 1, \ldots, n$, **compute** the negative gradient instances

$$Z_{i,t+1} = -\nabla C_n(G_t)(X_i).$$

5: **Fit** a regression tree to the pairs $(X_i, Z_{i,t+1})$, giving terminal nodes $R_{j,t+1}$, $1 \leq j \leq k$.

6: For $j = 1, \ldots, k$, **compute**

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,t+1}} \psi(G_t(X_i) + w, Y_i).$$

7: **Update**

(a) $F_{t+1} = G_t + \nu \sum_{j=1}^k w_{j,t+1} \mathbb{1}_{R_{j,t+1}}$.

(b) $G_{t+1} = (1-\gamma_t) F_{t+1} + \gamma_t F_t$.

(c) $\lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$, $\lambda_{t+1} = \frac{1+\sqrt{1+4\lambda_t^2}}{2}$.

(d) $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}$.

8: **end for**

9: **Output** F_T.

We see that the algorithm has two inner functional components, $(F_t)_t$ and $(G_t)_t$, which correspond respectively to the vectorial sequences $(x_t)_t$ and $(y_t)_t$ of Nesterov's acceleration scheme (1). Observe that the sequence $(G_t)_t$ is internal to the procedure while the linear combination output by the algorithm after T iterations is F_T. Line 2 initializes to the optimal constant model. As in Friedman's original approach, the algorithm selects at each iteration, by least-squares fitting, a particular tree that is in most agreement with the descent direction (the "gradient"), and then performs an update of G_t. The essential difference is the presence of the companion function sequence $(G_t)_t$, which slides the iterates $(F_t)_t$ according to the recursive parameters λ_t and γ_t (lines 7 (b)-(d)).

Let $f_{t+1} = \sum_{j=1}^k \beta_{j,t+1} \mathbb{1}_{R_{j,t+1}}$ be the approximate-gradient tree output in line 6 of the algorithm. The next logical step is to perform a line search to find the step size and update the model accordingly, as follows:

$$w_{t+1} \in \arg\min_{w>0} \sum_{i=1}^n \psi(G_t(X_i) + w f_{t+1}(X_i), Y_i), \quad F_{t+1} = G_t + w_{t+1} f_{t+1}.$$

However, following Friedman's gradient tree boosting (Friedman 2001), a separate optimal value $w_{j,t+1}$ is chosen for each of the tree's regions, instead of a single w_{t+1} for the whole

tree. This operation gives more latitude to the additive model and is known to usually improve the quality of the fit. Thus, the coefficients $\beta_{j,t+1}$ from the tree-fitting procedure can be then simply discarded, and the model update rule at epoch t becomes, for each $j = 1, \ldots, k$,

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,t+1}} \psi(G_t(X_i) + w, Y_i), \quad F_{t+1} = G_t + \nu \sum_{j=1}^{k} w_{j,t+1} \mathbb{1}_{R_{j,t+1}}$$

(lines 6 and 7 (a)). We also note that the contribution of the approximate gradient is scaled by a factor $0 < \nu < 1$ when it is added to the current approximation. The parameter ν can be regarded as controlling the learning rate of the boosting procedure. Smaller values of ν (more shrinkage) usually lead to larger values of T for the same training risk. Therefore, in order to reduce the number of trees composing the boosting estimate, large values for ν are required. However, too large values of ν may break the gradient descent dynamic, as shown for example in Biau and Cadre (2017, Lemma 3.2). Indeed, this lemma indicates that for a sufficiently smooth loss function $\psi(x, y)$ the difference $C(F_t) - C(F_{t+1})$ decreases as soon as ν is small enough. All in all, both ν and T control prediction risk on the training data and these parameters do not operate independently. This tradeoff issue is thoroughly explored in the next section.

3 Numerical studies

This section is devoted to illustrating the potential of our `AGB` algorithm and to highlighting the benefits of Nesterov's acceleration scheme in the boosting process. Synthetic models and real-life data are considered, and an exhaustive comparison with standard gradient tree boosting is performed. For the implementation of Friedman's boosting, we used the `R` package `gbm`, a description of which can be found in Ridgeway (2007). These two boosting algorithms are compared in the last subsection with the Lasso (Tibshirani 1996) and random forests (Breiman 2001) methods, respectively implemented with the packages `glmnet` and `randomForest`.

3.1 Description of the data sets

The algorithms were benchmarked on both simulated and real-life data sets. For each of the simulated models, we consider two designs for $X = (X_1, \ldots, X_d)$: Uniform over $(-1, 1)^d$ ("Uncorrelated design") and Gaussian with mean 0 and $d \times d$ covariance matrix Σ such that $\Sigma_{ij} = 2^{-|i-j|}$ ("Correlated design"). The following five models cover a wide spectrum of regression and classification problems. Models 1–3 and 5 come from Biau et al. (2016). Model 4 is a slight variation of a benchmark model in Hastie et al. (2009). Models 1–3 are regression problems, while Model 4 and 5 are ± 1-classification tasks. Models 2–4 are additive, while Model 1 and 5 include some interactions. Model 3 can be seen as a sparse high-dimensional problem. We denote by Z_{μ,σ^2} a Gaussian random variable with mean μ and variance σ^2.

Model 1 $n = 1\,000, d = 100, Y = X_1X_2 + X_3^2 - X_4X_7 + X_8X_{10} - X_6^2 + Z_{0,0.5}$.

Model 2 $n = 800, d = 100, Y = -\sin(2X_1) + X_2^2 + X_3 - \exp(-X_4) + Z_{0,0.5}$.

Table 1 Main characteristics of the five real-life data sets used in the experiments

Data set	n	d	Output Y
Adult	30,162	14	Binary
Advert.	2359	1431	Binary
Crime	1993	102	Continuous
Spam	4601	57	Binary
Wine	1559	11	Continuous

Model 3 $n = 1\,000, d = 500, Y = X_1 + 3X_3^2 - 2\exp(-X_5) + X_6.$

Model 4 $n = 2\,000, d = 30,$

$$Y = \begin{cases} 2\,\mathbb{1}_{\sum_{j=1}^{10} X_j^2 > 3.5} - 1 & \text{for uncorrelated design} \\ 2\,\mathbb{1}_{\sum_{j=1}^{10} X_j^2 > 9.34} - 1 & \text{for correlated design.} \end{cases}$$

Model 5 $n = 1\,500, d = 50, Y = 2\,\mathbb{1}_{X_1 + X_4^3 + X_9 + \sin(X_{12}X_{18}) + Z_{0,0.1} > 0.38} - 1.$

We also considered the following real-life data sets from the `UCI Machine Learning repository`: Adult, Internet Advertisements, Communities and Crime, Spam, and Wine. Their main characteristics are summarized in Table 1 (a more complete description is available at the address https://archive.ics.uci.edu/ml/datasets.html).

For each data set, simulated or real, the sample is divided into a training set (50%) $\mathscr{D}_{\text{train}}$ to fit the method; a validation set (25%) $\mathscr{D}_{\text{val}}$ to select the hyperparameters of the algorithms; and a test set (25%) $\mathscr{D}_{\text{test}}$ on which the predictive performance is evaluated. We considered two loss functions for both standard boosting and `AGB`: the least squares loss $\psi(x, y) = (y - x)^2$ for regression and the Adaboost loss $\psi(x, y) = e^{-yx}$ for ± 1-classification. We also tested the logit loss function $\psi(x, y) = \log_2(1 + e^{-yx})$. Since the results are similar to the Adaboost loss they are not reported.

In the boosting algorithms, the validation set is used to select the number of components of the model, i.e., the number of iterations performed by the algorithm. Thus, denoting by F_T the boosting predictor after T iterations fitted on $\mathscr{D}_{\text{train}}$, we select the $T^\star$ that minimizes

$$\frac{1}{\sharp \mathscr{D}_{\text{val}}} \sum_{i \in \mathscr{D}_{\text{val}}} \psi(F_T(X_i), Y_i). \tag{6}$$

For both standard gradient tree boosting and `AGB`, we fit regression trees with two terminal nodes. We considered five fixed values for the shrinkage parameter ν (1e − 05, 0.001, 0.01, 0.1, and 0.5), and fixed an arbitrary (large) limit of $T = 10\,000$ iterations for the standard boosting and $T = 2\,500$ for `AGB`. All results are averaged over 100 replications for simulated examples, and over 20 independent permutations of the sample for the real-life data.

3.2 Gradient boosting versus accelerated gradient boosting

In this subsection, we compare the standard gradient tree boosting and `AGB` algorithms in terms of minimization of the empirical risk (2) and selected number of components $T^\star$. Figure 3 shows the training and validation errors for Friedman's boosting and `AGB`, as a function of the number of iterations.

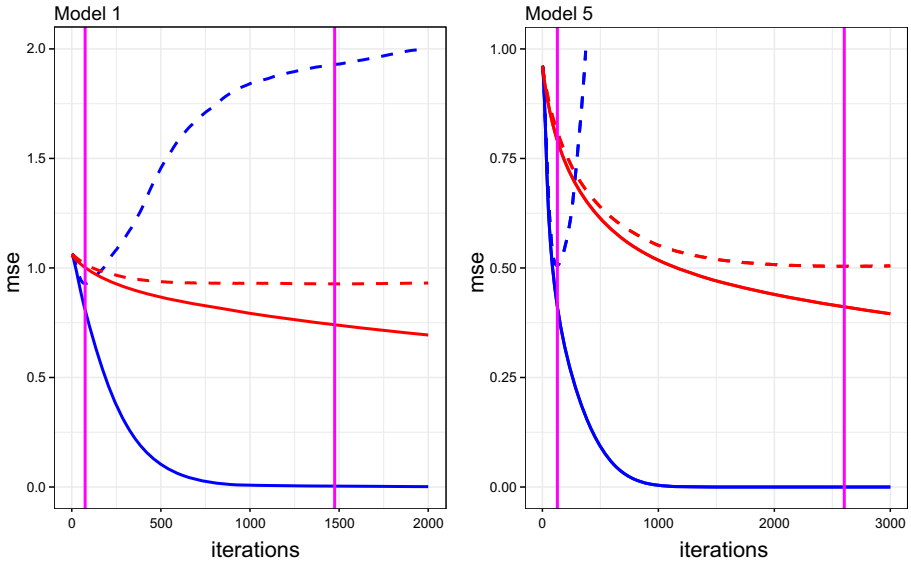

Fig. 3 Training (solid lines) and validation (dashed lines) errors for standard gradient boosting (red) and AGB (blue) for Model 1 (left) and Model 5 (right). Shrinkage parameter ν is fixed to 0.01 (Color figure online)

As it is generally the case for gradient boosting (e.g., Ridgeway 2007), the validation error decreases until predictive performance is at its best and then starts increasing again. The vertical magenta line shows the optimal number of iterations $T^\star$, selected by minimizing (6). We see that the validation rates at the optimal $T^\star$ are comparable for AGB and the original algorithm. However, AGB outperforms gradient boosting in terms of number of components of the output model, which is much smaller for AGB. This is a direct consequence of Nesterov's acceleration scheme. This remarkable behavior is confirmed by Figs. 4, 5, and 6, where we plotted the relationship between predictive performance, the number of iterations, and the shrinkage parameter. On the left side of each figure, we show the boxplots of the test errors of the selected predictors $F_{T^\star}$, i.e.,

$$\frac{1}{\sharp\mathscr{D}_{\text{test}}} \sum_{i \in \mathscr{D}_{\text{test}}} \psi(F_{T^\star}(X_i), Y_i), \tag{7}$$

as a function of the shrinkage parameter ν. The right sides depict the boxplots of the optimal number of components $T^\star$.

These three figures convey several messages. First of all, we notice that the predictive performances of the two methods are close to each other, independently of the data sets (simulated or real). Moreover, in line with the comments of Hastie et al. (2009, Chapter 10), smaller values of the shrinkage parameter ν favor better test error. Indeed, for all examples we observe that the best test errors are achieved for ν smaller than 0.1. However, for such values of ν, it seems difficult for standard boosting to reach the optimal $T^\star$ in a reasonable number of iterations, and 10 000 iterations are generally not sufficient as soon as ν is less than 0.01. The accelerated algorithm allows to circumvent this problem since, for each value of ν, the optimal model is achieved after a number of iterations considerably smaller than with standard boosting. Besides, AGB is less sensitive to the choice of ν. These two features are clear advantages since, in practice, one has no or few a priori information on the reasonable

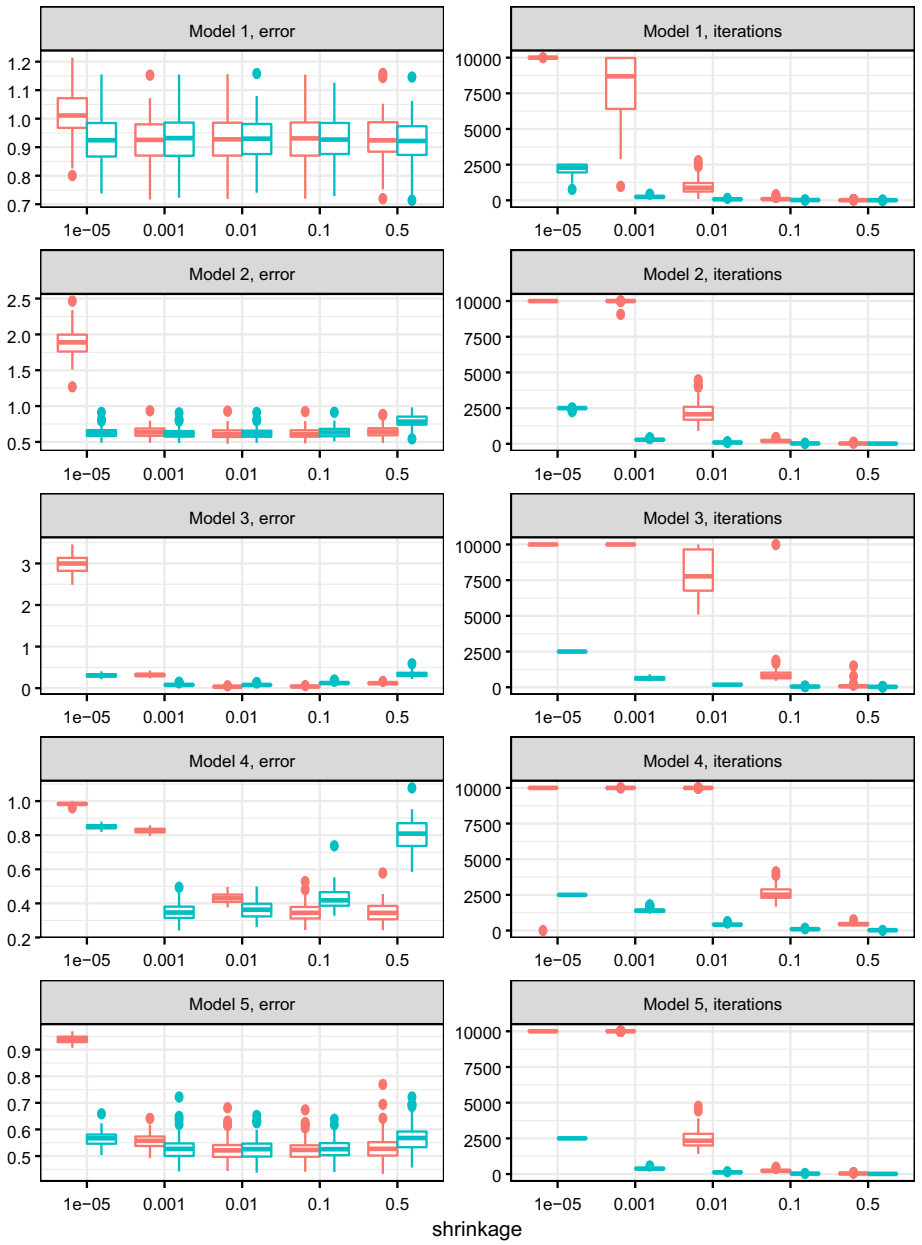

Fig. 4 Boxplots of the test error (7) (left) and selected numbers of iterations (right), as a function of the shrinkage parameter ν for standard gradient boosting (red, left) and AGB (blue, right). Results are presented for simulated models with uncorrelated (Color figure online)

value of ν, and the usual strategy is to try several (often, small) values of the shrinkage parameter until the validation error is the lowest. Let us finally note that it may be surprising, at first sight, to see the validation error rise so quickly once the optimum $T^\star$ is reached (Fig. 3). However, this rapid increase must be appreciated in view of the equally rapid decrease of

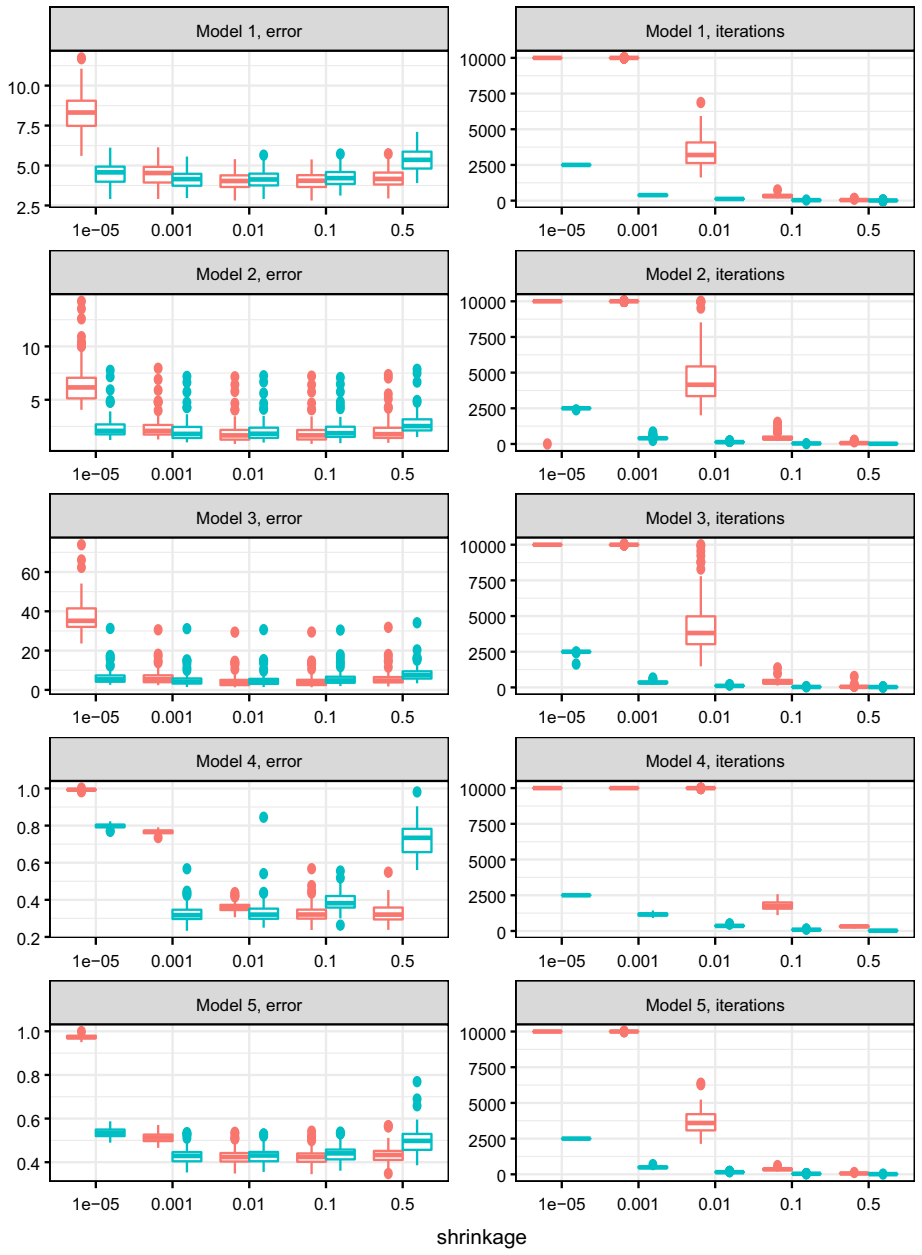

Fig. 5 Boxplots of the test error (7) (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and AGB (blue, right). Results are presented for simulated models with correlated design (Color figure online)

the training error. In fact, AGB overfits extremely fast and the validation error, for fixed ν, increases just as quickly.

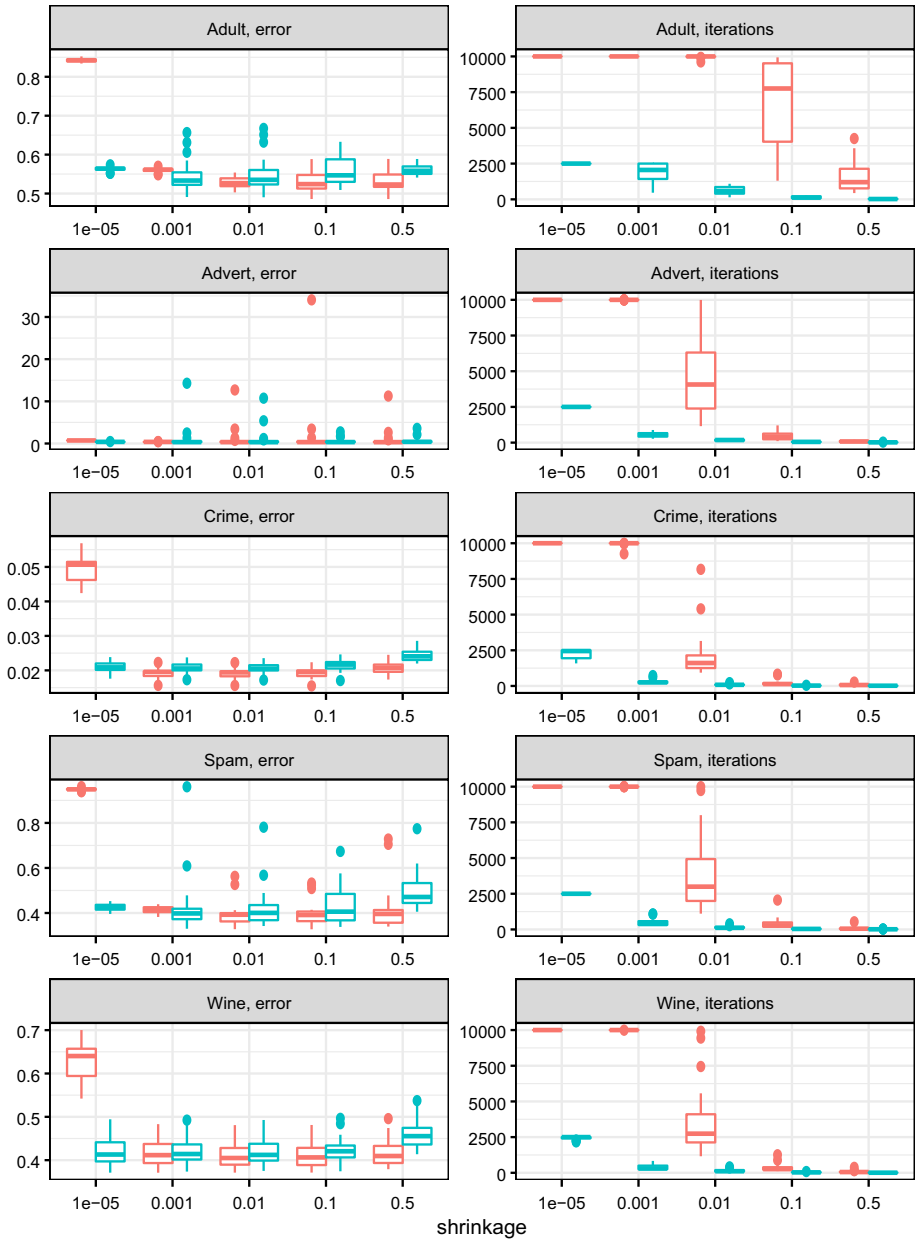

Fig. 6 Boxplots of the test error (7) (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and `AGB` (blue, right). Results are presented for real-life data sets (Color figure online)

Of course, the benefit of having sparser models is striking when we are faced with large-scale data, i.e., when iterations have a computational price. To illustrate this point, Table 2 provides the computation times to fit a tree with the default parameters of the `rpart` package. The computations have been performed on a laptop with 2.8 GHz processor and 16 Gb of

Table 2 Duration in seconds to fit a tree with `rpart` as a function of n (sample size) and d (dimension). Results are averaged over 100 repetitions

n	d		
	10	100	1000
1000	0.02	0.18	1.98
10,000	0.20	2.35	26.73
100,000	3.81	39.23	412.34

RAM memory. We clearly see that it is more and more expensive to fit a tree as the sample size and/or the dimension of the ambient space increase. It is in this large-scale context that `AGB` can have a decisive advantage over regular gradient boosting.

3.3 Time-varying versus fixed weights

We briefly discuss in this subsection the influence of the weights $(\gamma_t)_t$ (line (7)-(d) of the algorithm). As we have seen in the introduction, the current choice guarantees a fast convergence rate in $1/t^2$ when the function g to be minimized is convex and sufficiently smooth (Bubeck 2015, Theorem 3.19). Remarkably, if g is assumed to be strongly convex, then Nesterov's method can achieve a much faster (exponential and optimal) rate of convergence, using however a different set of weights *independent* of the iteration t (Bubeck 2015, Theorem 3.18). It turns out however that most of the losses $\psi(x, y)$ used in gradient boosting are convex but not strongly convex in x (e.g., the Adaboost exponential and the logit losses), and it is therefore a safe choice to use the "generic" weights $(\gamma_t)_t$ defined in (7)-(d). This option is all the more recommended as the `AGB` algorithm is not underpinned by any mathematical theory to date (see the discussion in Sect. 4). However, for regression problems, the least squares loss $\psi(x, y) = (y - x)^2$ is strongly convex, and it is thus tempting to operate with the fixed weights of Bubeck (2015, Theorem 3.18). This simply changes step (7)-(b) of the `AGB` algorithm into

$$G_{t+1} = \left(1 + \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right) F_{t+1} - \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} F_t,$$

where $\kappa = 1/(2\nu)$. We call this new algorithm `AGB2`. Figure 7 presents results for the simulated regression Models 1–3. We see that the estimated errors are of the same order for `AGB` and `AGB2`. However, `AGB` requires considerably less iterations to minimize the validation error (6), thus drastically reducing the model complexity.

3.4 Comparison with the Lasso and random forests

We compare in this last subsection the performance of the standard and accelerated boosting algorithms with that of the Lasso and random forests, respectively implemented with the `R` packages `glmnet` and `randomForest`. As above, the number of components $T^\star$ of the boosting predictors is selected by minimizing (6). The shrinkage parameter of the Lasso (parameter `lambda` in `glmnet`) and the number of variables randomly sampled as candidates at each split for the trees of the random forests (parameter `mtry` in `randomForest`) are selected by minimizing the mean squared error (regression) and the misclassification

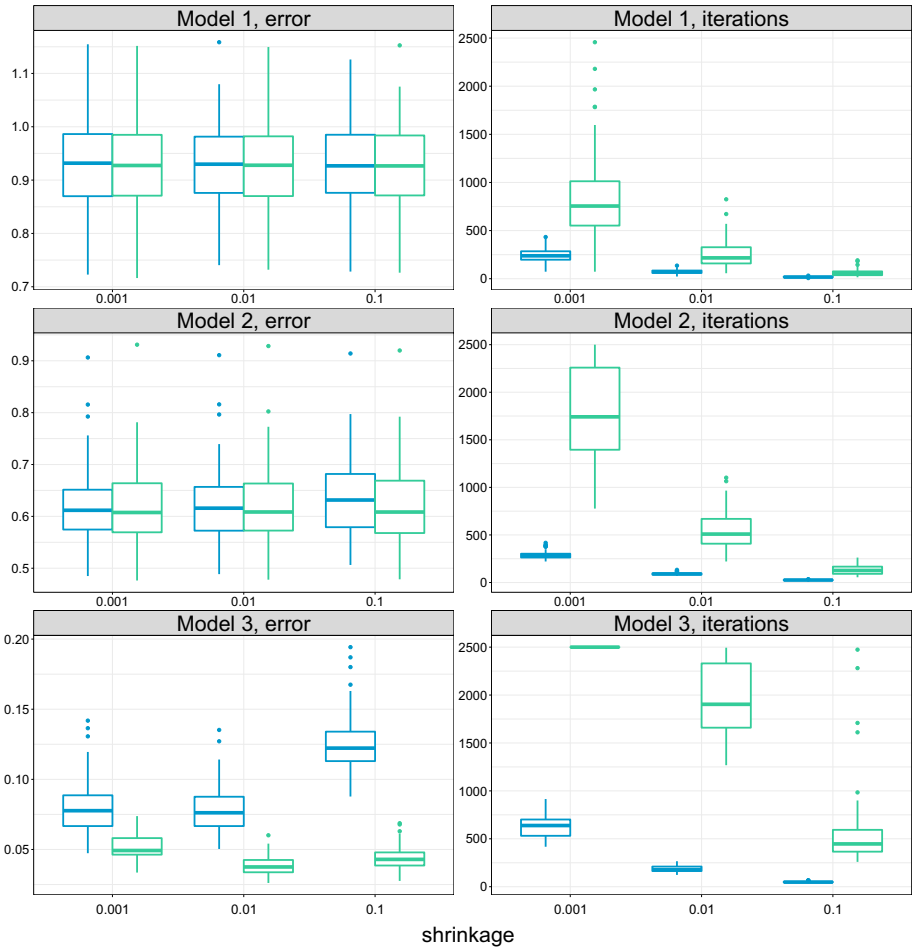

Fig. 7 Boxplots of the test error (7) (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for `AGB` (blue, left) and `AGB2` (green, right). Results are presented for simulated Models 1-3, with uncorrelated design (Color figure online)

error (classification) computed on the validation set. The `R`-package `caret` was used to conduct these minimization problems. The prediction performance of each predictor F was assessed on the test set by the mean squared error $\frac{1}{\sharp\mathscr{D}_{\text{test}}}\sum_{i\in\mathscr{D}_{\text{test}}}(Y_i - F(X_i))^2$ for regression problems, and (i) the misclassification error $\frac{1}{\sharp\mathscr{D}_{\text{test}}}\sum_{i\in\mathscr{D}_{\text{test}}}\mathbb{1}_{F(X_i)\neq Y_i}$ and (ii) the area under ROC curve (AUC) for classification problems (computed on the test set).

Table 3 shows the test errors for the regression problems, while Tables 4 and 5 display misclassification errors and AUC for classification tasks. All results are averaged over 100 replications for simulated examples and over 20 permutations of the sample for real-life data sets.

As might be expected, the results depend on the data sets, with an advantage to boosting algorithms, which are often the first and perform uniformly well. Besides, even if there is no clear winner between traditional boosting and `AGB`, we still find that `AGB` is weakly sensitive to the choice of ν and leads to more parsimonious models ($T^\star$ in the tables) for both regression and classification problems, and independently of the data set.

Table 3 Mean (m.) and standard deviation (SD) of the mean squared test error for the regression problems. Also shown for the boosting algorithms is the mean over all replications of the optimal number of components ($T^\star$). Results are averaged over 100 independent replications for simulated examples and over 20 independent permutations of the sample for real-life data sets

		GB					AGB					Lasso	RF
	ν	1e-05	0.001	0.01	0.1	0.5	1e−05	0.001	0.01	0.1	0.5		
Model 1 (u)	m.	1.011	0.923	0.926	0.927	0.930	0.924	0.927	0.926	0.929	**0.920**	1.021	**0.922**
	SD	0.082	0.075	0.076	0.076	0.079	0.076	0.077	0.074	0.074	0.081	0.084	0.078
	$T^\star$	10,000	7924	981	99	11	2178	247	73	18	7		
Model 2 (u)	m.	1.883	0.642	**0.621**	0.621	0.650	0.632	**0.621**	0.621	0.638	0.794	0.677	0.756
	SD	0.202	0.073	0.075	0.074	0.079	0.069	0.072	0.072	0.073	0.090	0.077	0.086
	$T^\star$	10,000	9 989	2 206	214	26	2488	288	91	26	14		
Model 3 (u)	m.	2.983	0.318	**0.037**	0.040	0.119	0.308	0.080	**0.078**	0.125	0.337	0.948	0.587
	SD	0.221	0.039	0.007	0.008	0.015	0.042	0.019	0.017	0.020	0.060	0.067	0.068
	$T^\star$	10,000	10,000	7936	956	97	2500	627	187	49	29		
Model 1 (c)	m.	8.316	4.483	**4.047**	4.051	4.220	4.529	4.141	**4.133**	4.252	5.354	8.549	4.163
	SD	1.143	0.668	0.557	0.559	0.573	0.669	0.564	0.566	0.575	0.694	1.154	0.623
	$T^\star$	10,000	10,000	3 413	330	47	2500	387	120	32	12		
Model 2 (c)	m.	6.558	2.424	**1.936**	1.938	2.093	2.442	2.083	**2.057**	2.145	2.777	4.988	2.082
	SD	1.958	1.120	1.093	1.095	1.118	1.117	1.087	1.087	1.062	1.103	1.580	0.824
	$T^\star$	9900	10,000	4632	458	70	2 499	411	132	35	16		
Model 3 (c)	m.	37.034	6.323	**4.454**	4.480	5.879	6.382	5.274	**5.163**	5.781	8.187	23.898	6.198
	SD	8.617	3.883	3.703	3.708	3.948	3.936	3.824	3.761	3.827	4.020	5.746	3.421
	$T^\star$	10,000	10,000	4296	415	54	2 491	361	113	31	23		

Table 3 continued

		GB					AGB					Lasso	RF
	ν	1e-05	0.001	0.01	0.1	0.5	1e−05	0.001	0.01	0.1	0.5		
Crimes	m.	0.049	0.019	**0.019**	0.019	0.021	0.021	0.021	0.021	0.021	0.024	**0.019**	**0.019**
	SD	0.004	0.001	0.001	0.002	0.002	0.002	0.001	0.001	0.002	0.002	0.001	0.001
	$T^\star$	10,000	9960	2172	214	86	2240	296	91	26	16		
Wine	m.	0.632	0.417	**0.412**	0.412	0.419	0.421	0.421	0.421	0.424	0.459	0.426	**0.365**
	SD	0.044	0.032	0.032	0.032	0.032	0.034	0.033	0.032	0.032	0.034	0.001	0.001
	$T^\star$	10,000	9999	3727	366	79	2433	393	154	36	11		

For each data set, the two best performances are in bold

Table 4 Mean (m.) and standard deviation (SD) of the misclassification test errors for the classification problems. Also shown for the boosting algorithms is the mean over all replications of the optimal number of components ($T^\star$). Results are averaged over 100 independent replications for simulated examples and over 20 independent permutations of the sample for real-life data sets

		GB					AGB					Lasso	RF
	ν	1e-05	0.001	0.01	0.1	0.5	1e-05	0.001	0.01	0.1	0.5		
Model 4 (u)	m.	0.416	0.229	0.098	0.086	**0.085**	0.248	**0.085**	0.088	0.108	0.217	0.419	0.206
	SD	0.020	0.023	0.018	0.015	0.016	0.023	0.016	0.016	0.017	0.036	0.021	0.025
	$T^\star$	9900	10,000	9998	2619	452	2500	1404	421	97	22		
Model 5 (u)	m.	0.353	0.144	**0.141**	0.141	0.142	0.145	0.142	**0.141**	0.144	0.155	**0.138**	0.151
	SD	0.024	0.016	0.017	0.016	0.018	0.017	0.018	0.017	0.016	0.021	0.018	0.019
	$T^\star$	10,000	10,000	2465	240	41	2500	387	121	34	12		
Model 4 (c)	m.	0.451	0.171	0.086	0.081	**0.079**	0.185	**0.080**	0.081	0.095	0.183	0.453	0.134
	SD	0.027	0.020	0.015	0.014	0.014	0.022	0.014	0.015	0.015	0.03	0.025	0.018
	$T^\star$	10,000	10,000	9996	1781	319	2500	1156	358	88	23		
Model 5 (c)	m.	0.423	0.119	**0.114**	0.114	0.115	0.123	**0.114**	0.116	0.118	0.132	0.118	0.116
	SD	0.037	0.016	0.015	0.016	0.016	0.018	0.016	0.016	0.016	0.020	0.016	0.018
	$T^\star$	10,000	10,000	3694	354	65	2500	493	151	40	14		
Adult	m.	0.249	0.150	0.141	**0.138**	0.138	0.151	**0.140**	0.140	0.143	0.152	0.155	0.186
	SD	0.004	0.004	0.004	0.004	0.004	0.004	0.005	0.005	0.004	0.004	0.004	0.005
	$T^\star$	10,000	10,000	9966	6714	1635	2500	1853	610	143	24		
Advert	m.	0.165	0.062	0.043	0.043	0.043	0.063	0.043	0.043	0.044	0.054	**0.032**	**0.031**
	SD	0.014	0.009	0.012	0.013	0.012	0.008	0.013	0.013	0.011	0.011	0.007	0.009
	$T^\star$	10,000	9999	4716	471	87	2500	568	181	50	18		
Spam	m.	0.396	0.071	**0.061**	0.061	0.065	0.077	0.064	0.065	0.068	0.086	0.095	**0.057**
	SD	0.013	0.009	0.008	0.007	0.007	0.009	0.009	0.007	0.007	0.011	0.072	0.007
	$T^\star$	10,000	10,000	3880	426	84	2500	479	150	40	16		

For each data set, the two best performances are in bold

Table 5 Mean (m.) and standard deviation (SD) of AUC for the classification problems. Also shown for the boosting algorithms is the mean over all replications of the optimal number of components ($T^\star$). Results are averaged over 100 independent replications for simulated examples and over 20 independent permutations of the sample for real-life data sets

		GB					AGB					Lasso	RF
	ν	1e-05	0.001	0.01	0.1	0.5	1e-05	0.001	0.01	0.1	0.5		
Model 4 (u)	m.	0.590	0.885	0.971	0.976	**0.977**	0.869	**0.977**	0.975	0.964	0.862	0.515	0.891
	SD	0.037	0.021	0.008	0.007	0.007	0.023	0.007	0.007	0.010	0.040	0.018	0.021
	$T^\star$	9900	10,000	9998	2 619	452	2 500	1404	421	97	22		
Model 5 (u)	m.	0.772	0.935	0.936	0.936	0.934	0.933	**0.937**	0.936	0.935	0.922	**0.940**	0.922
	SD	0.059	0.013	0.012	0.012	0.013	0.013	0.013	0.012	0.012	0.015	0.011	0.016
	$T^\star$	10,000	10,000	2 465	240	41	2500	387	121	34	12		
Model 4 (c)	m.	0.621	0.927	0.978	**0.981**	0.981	0.916	**0.981**	0.981	0.972	0.898	0.516	0.945
	SD	0.043	0.014	0.006	0.005	0.005	0.016	0.005	0.005	0.008	0.030	0.019	0.012
	$T^\star$	10,000	10,000	9996	1781	319	2500	1 156	358	88	23		
Model 5 (c)	m.	0.753	0.960	0.962	**0.963**	0.961	0.957	**0.962**	0.962	0.960	0.947	0.960	0.955
	SD	0.059	0.009	0.008	0.008	0.008	0.009	0.008	0.008	0.008	0.011	0.007	0.011
	$T^\star$	10,000	10,000	3694	354	65	2500	493	151	40	14		
Adult	m.	0.758	0.905	0.915	**0.920**	0.920	0.902	**0.918**	0.917	0.913	0.901	0.902	0.858
	SD	0.005	0.004	0.004	0.004	0.003	0.004	0.004	0.004	0.003	0.004	0.004	0.008
	$T^\star$	10,000	10,000	9966	6714	1635	2500	1853	610	143	24		
Advert	m.	0.815	0.962	0.974	0.973	0.973	0.956	0.973	**0.975**	0.971	0.950	0.973	**0.983**
	SD	0.059	0.014	0.011	0.012	0.013	0.015	0.014	0.011	0.015	0.022	0.008	0.008
	$T^\star$	10,000	9999	4716	471	87	2500	568	181	50	18		
Spam	m.	0.854	0.975	**0.980**	0.980	0.979	0.973	0.978	0.978	0.977	0.966	0.970	**0.979**
	SD	0.028	0.003	0.003	0.003	0.003	0.004	0.003	0.003	0.003	0.005	0.004	0.003
	$T^\star$	10,000	10,000	3880	426	84	2 500	479	150	40	16		

For each data set, the two best performances are in bold

4 Conclusion and discussion

In this paper, we have proposed an algorithm named Accelerated Gradient Boosting (AGB). It is based on Friedman's gradient tree boosting algorithm (Friedman 2001), and incorporates the Nesterov's accelerated gradient descent technique (Nesterov 1983) to the gradient step. Extensive numerical experiments were conducted that reach the following conclusion: AGB achieves a similar level of predictive error as gradient boosting, but uses far less components in the output model and is less sensitive to the shrinkage parameter. Our results are best summarized by Fig. 6, which offers the most compelling evidence about the benefits of AGB: the plots on the left show statistically significant performance similarities between AGB and regular gradient tree boosting, while the right clearly show a substantive drop in the number of iterations required (resulting in sparser models). The code base is made freely available at https://github.com/lrouviere/AGB.

The present article is based on empirical considerations and cannot, on its own, explain the reasons for the good performance of the AGB algorithm. This would require a thorough analysis of the theoretical properties of the combination gradient boosting + Nesterov's acceleration, taking the point of view of functional optimization. Such an analysis is difficult and goes far beyond the scope of our work. In fact, even for regular gradient boosting, few theoretical results are known and much work remains to be done to clarify the mathematical forces driving the algorithm. Many articles regard boosting with a statistical eye and study the somewhat idealized problem of empirical risk minimization with a convex loss (e.g., Blanchard et al. 2003; Lugosi and Vayatis 2004). These papers essentially concentrate on the statistical properties of the approach (that is, consistency and rates of convergence as the sample size grows) and often ignore the underlying optimization aspects. Other articles, such as Bühlmann and Yu (2003), Zhang and Yu (2005), Bartlett and Traskin (2007) take advantage of the iterative principle of boosting, but mainly focus on regularization via early stopping (that is, stopping the boosting iterations at some point), without paying too much attention to the optimization side. Notable exceptions are the pioneering notes of Breiman (1997, 1998, 1999, 2000, 2004), together with the paper by Mason et al. (2000), who envision gradient boosting as an infinite-dimensional numerical optimization problem and pave the way for more abstract investigations. More recently, Biau and Cadre (2017) analyze two versions of gradient boosting and prove their convergence as the number of iterations tends to infinity. Nevertheless, despite all these research efforts, there is to date no sound theory of gradient boosting.

On the other hand, Nesterov's accelerated descent is provably faster than gradient descent when the gradient used is accurate (see Bubeck 2015, Chapter 3). However, when the gradient is not accurate (e.g., in a stochastic setting), then Nesterov's descent is prone to accumulating error and diverging. This type of situation is analyzed in Devolder et al. (2014), who prove that the superiority of fast gradient methods over the classical ones is no longer absolute when an inexact oracle is used. Therefore, the benefits of Nesterov's technique may be lost, or reduced, in some inexact gradient settings. This is of course the case in our boosting problem, since the gradient direction is highly inexact due to the least squares approximation (5). It is thus theoretically not immediately clear when and how Nesterov's descent can really help gradient boosting. Therefore, beyond our empirical findings, it is essential to tackle the problem from a mathematical point of view. With this respect, we note that Jain et al. (2018) address the issues of instability and error accumulation of fast gradient methods for the special case of stochastic approximation for the least squares regression problem. They show in particular that acceleration can be made robust to statistical errors by introducing an

accelerated stochastic gradient method that provably achieves the minimax optimal statistical risk faster than stochastic gradient descent.

Acknowledgements We greatly thank two referees for valuable comments and insightful suggestions, which led to a substantial improvement of the paper.

References

Bartlett, P. L., & Traskin, M. (2007). AdaBoost is consistent. *Journal of Machine Learning Research*, *8*, 2347–2368.

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, *2*, 183–202.

Becker, S., Bobin, J., & Candès, E. J. (2011). NESTA: A fast and accurate first-order method for sparse recovery. *SIAM Journal on Imaging Sciences*, *4*, 1–39.

Biau, G., & Cadre, B. (2017). Optimization by gradient boosting. arXiv:1707.05023.

Biau, G., Fischer, A., Guedj, B., & Malley, J. D. (2016). COBRA: A combined regression strategy. *Journal of Multivariate Analysis*, *146*, 18–28.

Bickel, P. J., Ritov, Y., & Zakai, A. (2006). Some theory for generalized boosting algorithms. *Journal of Machine Learning Research*, *7*, 705–732.

Blanchard, G., Lugosi, G., & Vayatis, N. (2003). On the rate of convergence of regularized boosting classifiers. *Journal of Machine Learning Research*, *4*, 861–894.

Breiman, L. (1997). *Arcing the edge*. Technical Report 486, Statistics Department, University of California, Berkeley.

Breiman, L. (1998). Arcing classifiers (with discussion). *The Annals of Statistics*, *26*, 801–824.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, *11*, 1493–1517.

Breiman, L. (2000). *Some infinite theory for predictor ensembles*. Technical Report 577, Statistics Department, University of California, Berkeley.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Breiman, L. (2004). Population theory for boosting ensembles. *The Annals of Statistics*, *32*, 1–11.

Bubeck, S. (2013). *ORF523: Nesterov's accelerated gradient descent*. https://blogs.princeton.edu/imabandit/2013/04/01/acceleratedgradientdescent.

Bubeck, S. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, *8*, 231–357.

Bühlmann, P., & Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting (with discussion). *Statistical Science*, *22*, 477–505.

Bühlmann, P., & Yu, B. (2003). Boosting with the L_2 loss: Regression and classification. *Journal of the American Statistical Association*, *98*, 324–339.

Chen, T. & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794). New York: ACM.

Devolder, O., Glineur, F., & Nesterov, Y. (2014). First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, *146*, 37–75.

Devroye, L., Györfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. New York: Springer.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, *121*, 256–285.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In S. Lorenza (Ed.), *Machine learning: Proceedings of the thirteenth international conference on machine learning* (pp. 148–156). San Francisco: Morgan Kaufmann Publishers.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting (with discussion). *The Annals of Statistics*, *28*, 337–374.

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

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*, 367–378.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). New York: Springer.

Jain, P., Netrapalli, P., Kakade, S. M., Kidambi, R., & Sidford, A. (2018). Accelerating stochastic gradient descent for least squares regression. In S. Bubeck, V. Perchet, & P. Rigollet (Ed.), *Proceedings of the 31st conference on learning theory* (Vol. 75, pp. 545–604). PMLR.

Lugosi, G., & Vayatis, N. (2004). On the Bayes-risk consistency of regularized boosting methods. *The Annals of Statistics*, *32*, 30–55.

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent. In S. A. Solla, T. K. Leen, & K. Müller (Eds.), *Proceedings of the 12th international conference on neural information processing systems* (pp. 512–518). Cambridge, MA: The MIT Press.

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (2000). Functional gradient techniques for combining hypotheses. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers* (pp. 221–246). Cambridge, MA: The MIT Press.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, *27*, 372–376.

Nesterov, Y. (2004). *Introductory lectures on convex optimization: A basic course*. New York: Springer.

Nesterov, Y. (2005). Smooth minimization of non-smooth functions. *Mathematical Programming*, *103*, 127–152.

Nesterov, Y. (2013). Gradient methods for minimizing composite functions. *Mathematical Programming*, *140*, 125–161.

Qu, G., & Li, N. (2016). Accelerated distributed Nesterov gradient descent. In *54th Annual Allerton conference on communication, control, and computing* (pp. 209–216). Red Hook: Curran Associates, Inc.

Ridgeway, G. (2007). *Generalized boosted models: A guide to the gbm package*. http://www.saedsayad.com/docs/gbm2.pdf.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*, 197–227.

Su, W., Boyd, S., & Candès, E. J. (2016). A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, *17*, 1–43.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning, proceedings of machine learning research* (pp. 1139–1147).

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B*, *58*, 267–288.

Tseng, P. (2008). *On accelerated proximal gradient methods for convex-concave optimization*. http://www.mit.edu/~dimitrib/PTseng/papers/apgm.pdf.

Zhang, T., & Yu, B. (2005). Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, *33*, 1538–1579.