Accelerated Gradient Boosting

G. Biau
Sorbonne Université, CNRS, LPSM
Paris, France
gerard.biau@upmc.fr

B. Cadre
Univ Rennes, CNRS, IRMAR
Rennes, France
benoit.cadre@ens-rennes.fr

L. Rouvière
Univ Rennes, CNRS, IRMAR
Rennes, France
laurent.rouviere@univ-rennes2.fr

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 much 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.

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 for example in the introduction of Chen and Guestrin (2016). To get to the point, 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.

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 smooth convex function $f(x)$ over $\mathbb{R}^d$, then Nesterov's descent may take the following form (Beck and Teboulle, 2009): starting with $x_0 = y_0$, inductively define

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

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 $(x_t)_t$ recovers the minimum of f at a rate of order $1/t^2$, in contrast to vanilla gradient descent methods, which have the same computational complexity but can only achieve a rate in $O(1/t)$. 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 `Accelerated Gradient Boosting`), 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 Figure 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`.

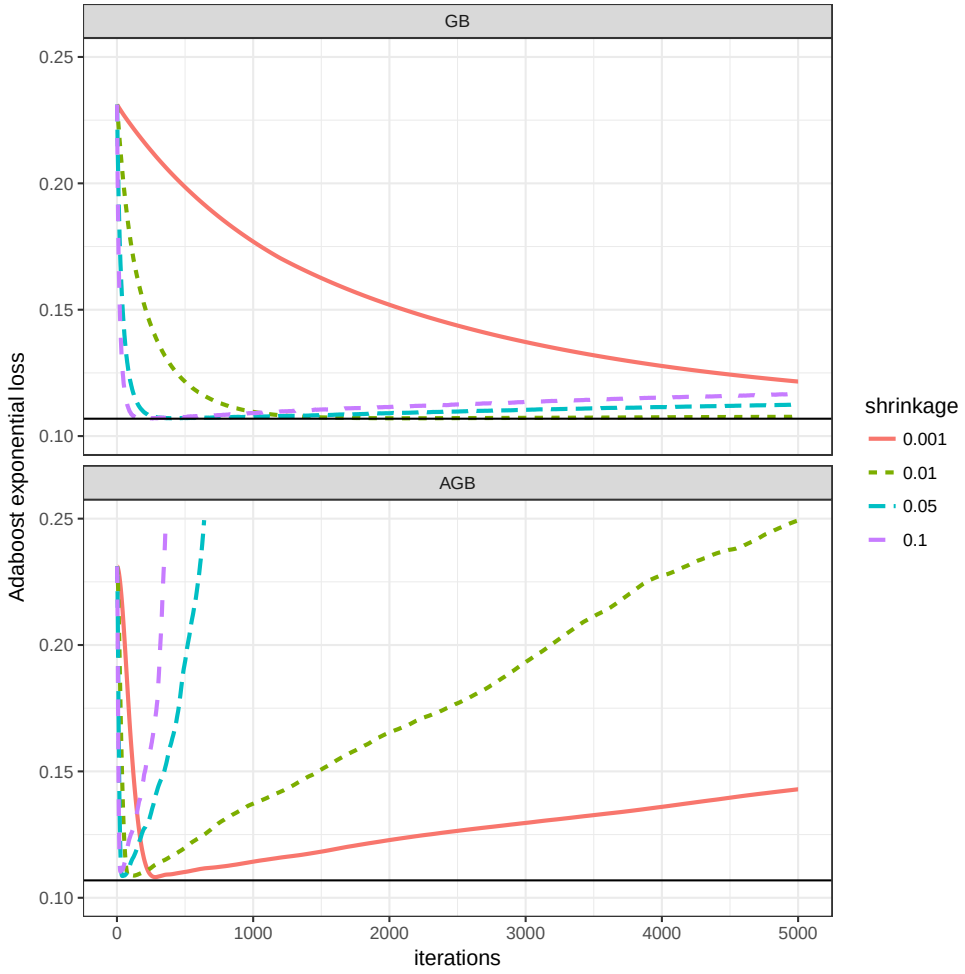

Figure 1: Adaboost exponential loss (estimated on a test data set) by number of iterations for standard gradient boosting (top) and `AGB` (bottom). The data are generated according to `Model 5` with $n = 5\,000$ observations (see page 8).

The paper is organized as follows. In Section 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. 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, all distributed as an independent generic pair (X,Y) 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. 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 $F_n = \sum_{j=0}^{J} \alpha_j f_j$, where $(\alpha_0,\ldots,\alpha_J) \in \mathbb{R}^{J+1}$ and each component f_j 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 differentiable in its first argument, and 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

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

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 $\psi(x,y) = \ln_2(1+e^{-yx})$ (logit loss).

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). Thus, 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 Figure 2.

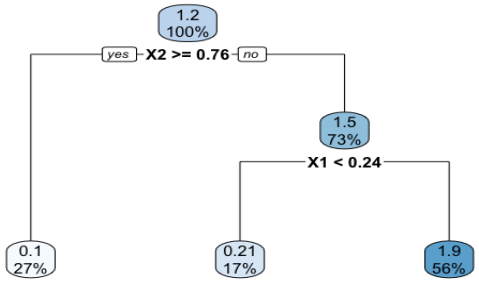

Figure 2: A regression tree in dimension $d = 2$ with $k = 3$ leaves.

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

Suppose 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 $C_n(F_t + w f_{t+1})$ 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 $C_n(F_t + w f_{t+1})$ most rapidly decreases. Observe that, for all $F \in \mathrm{lin}(\mathscr{F})$, $\nabla C_n(F)(X_i) = \partial_x \psi(F(X_i), Y_i)$, where the symbol ∂_x means partial derivative with respect to the first component. Then the knee-jerk reaction is to take $f_{t+1}(\cdot) = -\nabla C_n(F_t)(\cdot)$, the opposite of the gradient of C_n at F_t (this is a function over $\mathbb{R}^d$), and do something like

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

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

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

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. This is the general principle of Friedman's gradient boosting (Friedman, 2001), which after T iterations outputs an

additive expansion of the form $F_T = \sum_{t=0}^{T} \alpha_t f_t$. The operational algorithm includes several regularization techniques to reduce the eventual overfitting. Some of these features are incorporated in our accelerated version, which we now describe.

2.2 The AGB algorithm

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

`AGB` algorithm

1: **Require** $T \geq 1$ (number of iterations), $k \geq 1$ (number of terminal nodes in the trees), $0 < \nu < 1$ (shrinkage parameter).

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

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

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`).

Data set	n	d	**Output** Y
Adult	30 162	14	binary
Advert.	2 359	1 431	binary
Crime	1 993	102	continuous
Spam	4 601	57	binary
Wine	1 559	11	continuous

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

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 $\psi(x,y) = \ln_2(1+e^{-yx})$. 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

$$\frac{1}{\sharp\mathscr{D}_{\text{val}}} \sum_{i \in \mathscr{D}_{\text{val}}} \psi(F_T(X_i), Y_i). \tag{3}$$

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 vs 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` (bottom), as a function of the number of iterations.

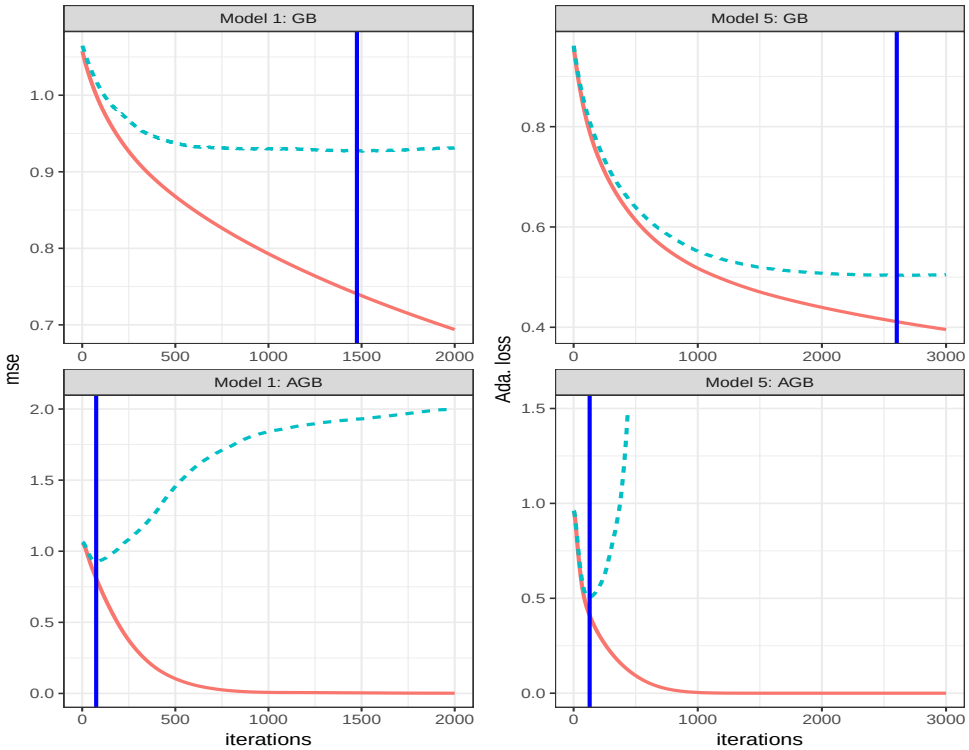

Figure 3: Training (solid lines) and validation (dashed lines) errors for Model 1 and Model 5. Shrinkage parameter ν is fixed to 0.01.

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 blue line shows the optimal number of iterations $T^\star$, selected by minimizing (3). 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 Figures 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.,

$$\frac{1}{\sharp\mathscr{D}_{\text{test}}} \sum_{i \in \mathscr{D}_{\text{test}}} \psi(F_{T^\star}(X_i), Y_i), \tag{4}$$

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 much 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 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. Of course, this benefit is striking when we are faced with large-scale data, i.e., when iterations have a computational price.

3.3 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 are selected by minimizing (3). 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 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 were 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 2 shows the test errors for the regression problems, while Tables 3 and 4 display misclassification errors and AUC for classification tasks. All results are averaged over 100

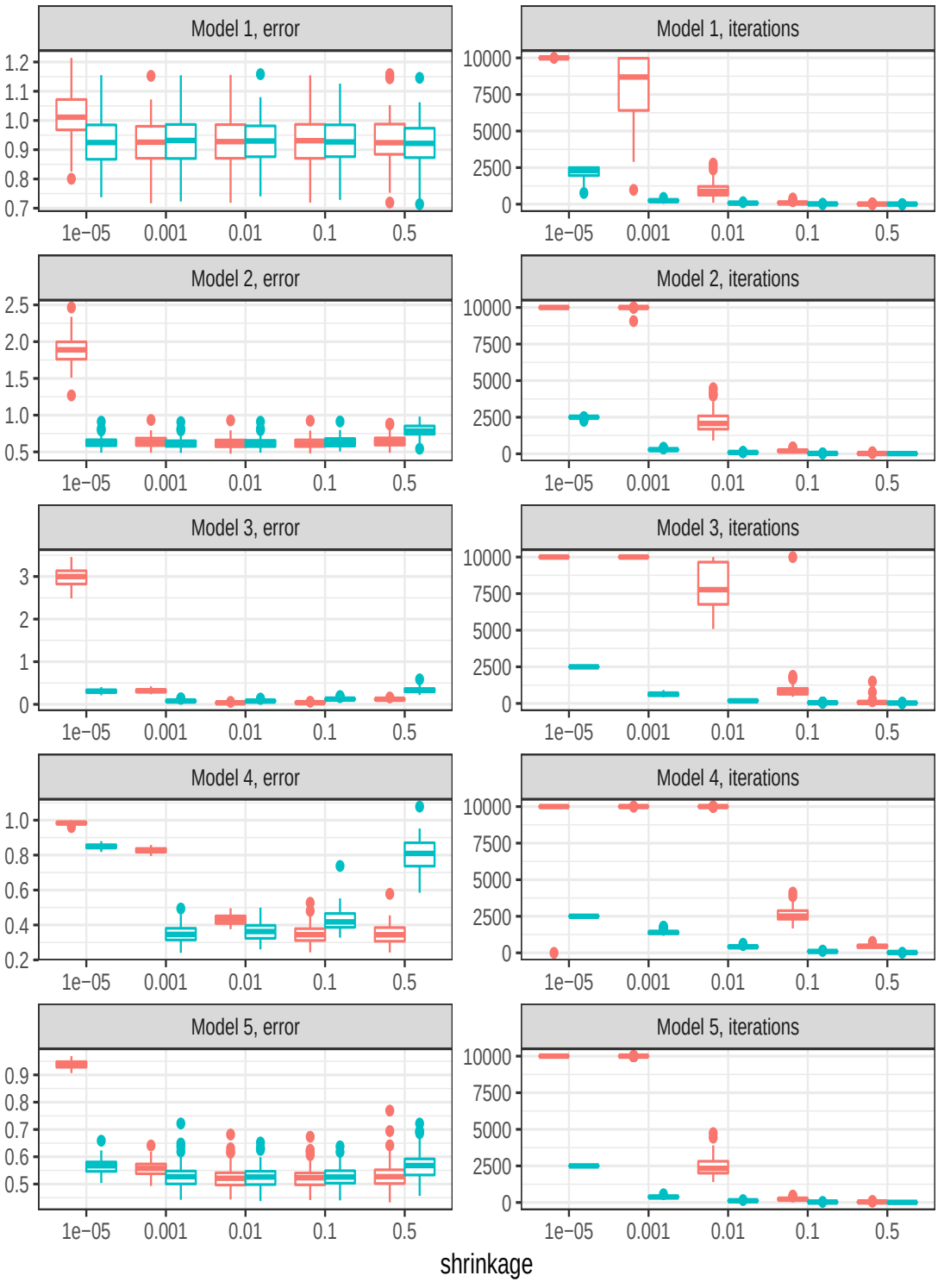

Figure 4: Boxplots of the test error (4) (left) and selected numbers of iterations (right), as a function of the shrinkage parameter ν for standard gradient boosting (red, left) and `AGB` (blue, right). Results are presented for simulated models with uncorrelated design.

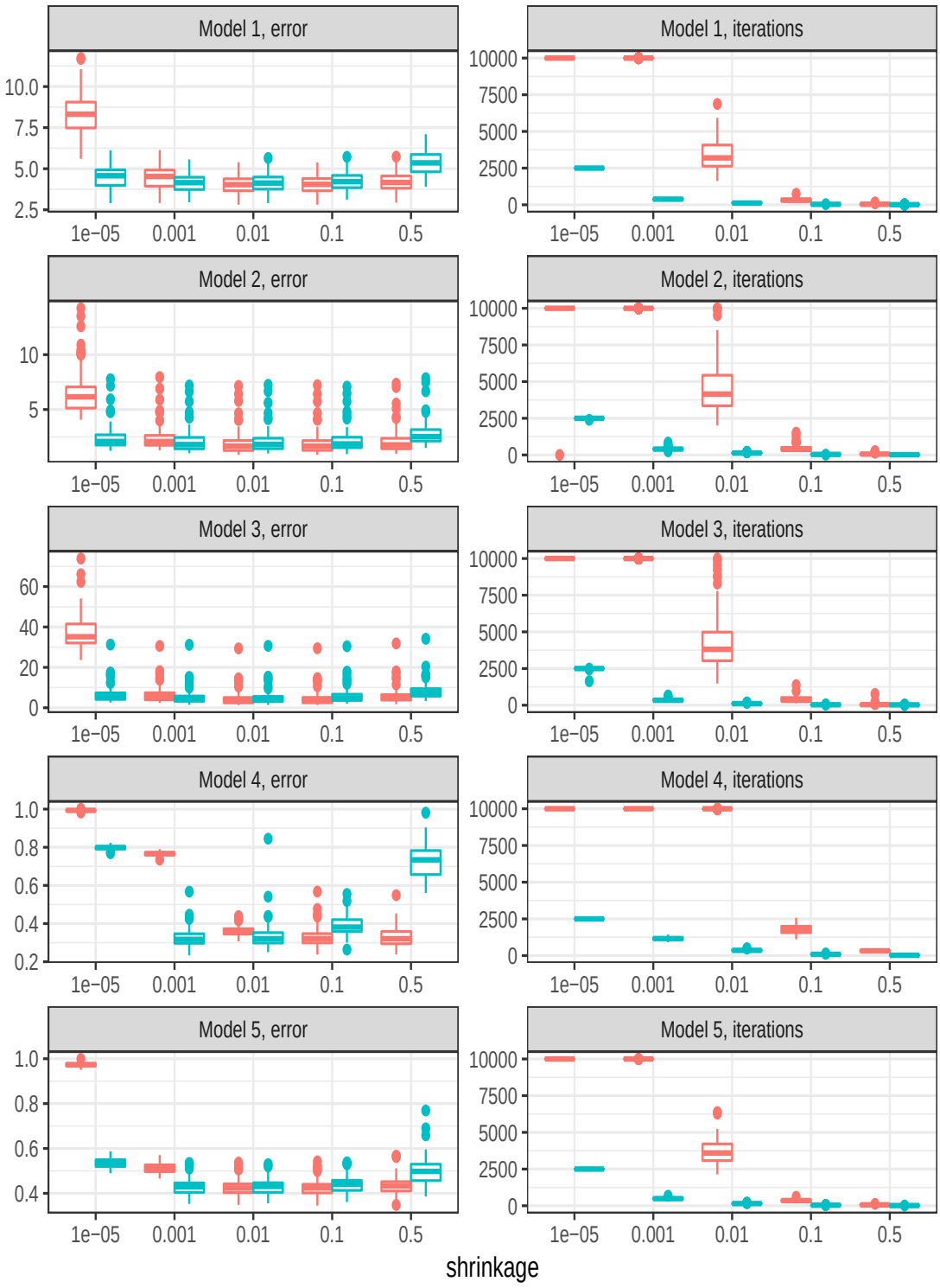

Figure 5: Boxplots of the test error (4) (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and `AGB` (blue, right). Results are presented for simulated models with correlated design.

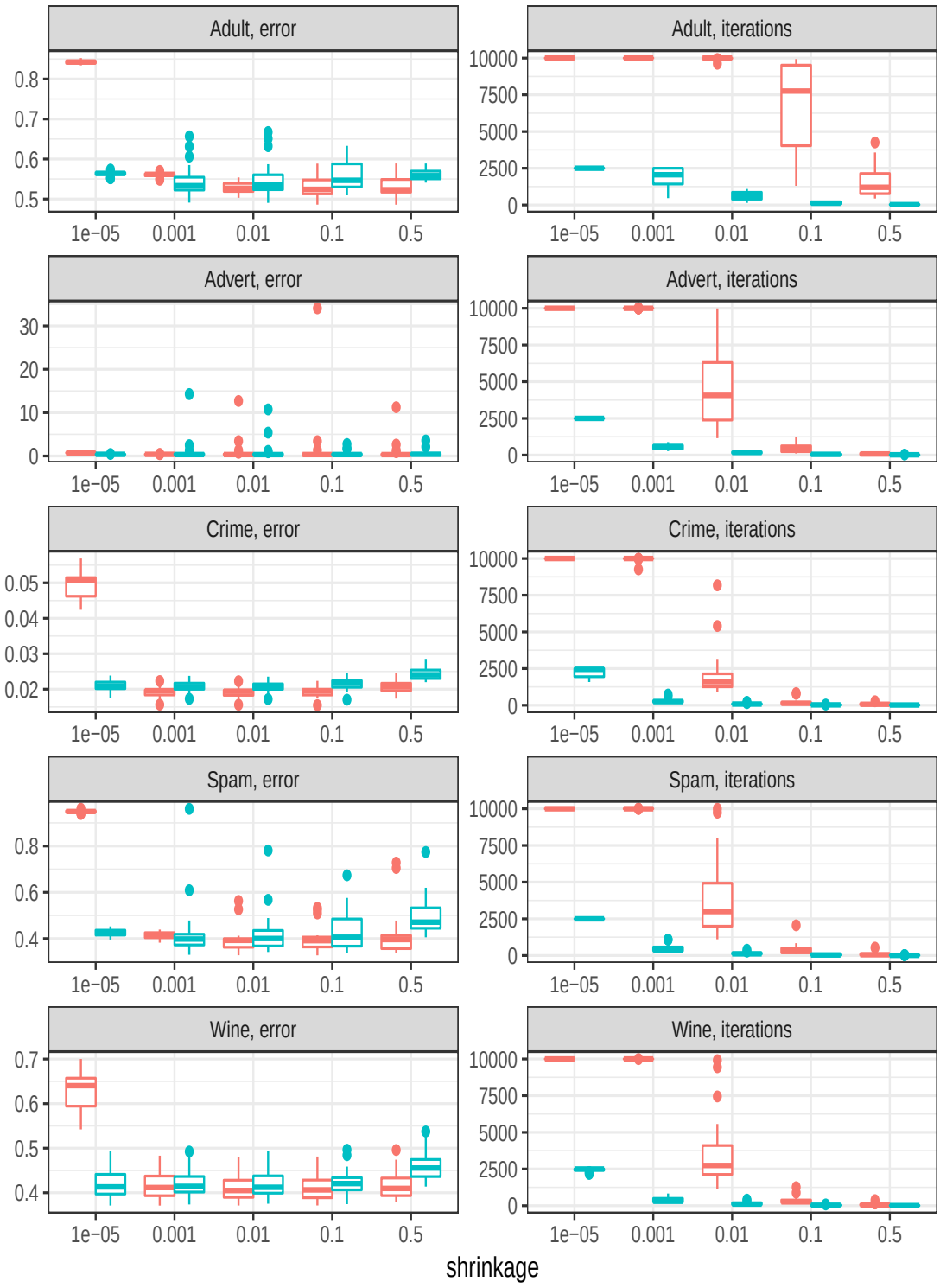

Figure 6: Boxplots of the test error (4) (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and `AGB` (blue, right). Results are presented for real-life data sets.

replications for simulated examples and over 20 permutations of the sample for real-life data set.

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. They are the take-home messages of our paper.

		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	7 924	981	99	11	2 178	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	2 488	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	10000	7 936	956	97	2 500	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	2 500	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
	$T^\star$	9 900	10 000	4 632	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	4 296	415	54	2 491	361	113	31	23		
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	9 960	2 172	214	86	2 240	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	9 999	3727	366	79	2 433	393	154	36	11		

Table 2: 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. For each data set, the two best performances are in bold.

		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
	$T^\star$	9 900	10 000	9 998	2 619	452	2 500	1 404	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	2 465	240	41	2 500	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	9 996	1 781	319	2 500	1 156	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	3 694	354	65	2 500	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	9 966	6 714	1 635	2 500	1 853	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	9 999	4 716	471	87	2 500	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	3 880	426	84	2 500	479	150	40	16		

Table 3: 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. For each data set, the two best performances are in bold.

		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
	$T^\star$	9 900	10 000	9 998	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	2 500	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	9 996	1 781	319	2 500	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	3 694	354	65	2 500	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	9 966	6 714	1 635	2 500	1 853	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	3 880	426	84	2 500	479	150	40	16		

Table 4: 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. For each data set, the two best performances are in bold.

References

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

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

G. Biau and B. Cadre. *Optimization by gradient boosting*. arXiv:1707.05023, 2017.

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

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

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

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

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

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

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

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

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

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

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

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

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, New York, 2016.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

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

Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In S. Lorenza, editor, *Machine Learning: Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann Publishers, San Francisco, 1996.

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

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

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

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

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition*. Springer, New York, 2009.

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

L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In S.A. Solla, T.K. Leen, and K. Müller, editors, *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 512–518. The MIT Press, Cambridge, MA, 1999.

L. Mason, J. Baxter, P. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 221–246. The MIT Press, Cambridge, MA, 2000.

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

Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Science+Business Media, New York, 2004.

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

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

G. Qu and N. Li. Accelerated distributed Nesterov gradient descent. In *54th Annual Allerton Conference on Communication, Control, and Computing*, pages 209–216. Curran Associates, Inc., Red Hook, 2016.

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

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

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

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147. Proceedings of Machine Learning Research, 2013.

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

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

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