Submitted to Statistical Science

BOOSTING ALGORITHMS: REGULARIZATION, PREDICTION AND MODEL FITTING

By Peter Bühlmann and Torsten Hothorn

ETH Zürich and Universität Erlangen-Nürnberg

We present a statistical perspective on boosting. Special emphasis is given to estimating potentially complex parametric or nonparametric models, including generalized linear and additive models as well as regression models for survival analysis. Concepts of degrees of freedom and corresponding Akaike or Bayesian information criteria, particularly useful for regularization and variable selection in high-dimensional covariate spaces, are discussed as well.

The practical aspects of boosting procedures for fitting statistical models are illustrated by means of the dedicated open-source software package **mboost**. This package implements functions which can be used for model fitting, prediction and variable selection. It is flexible, allowing for the implementation of new boosting algorithms optimizing user-specified loss functions.

1. Introduction. Freund and Schapire's AdaBoost algorithm for classification [29–31] has attracted much attention in the machine learning community [cf. 76, and the references therein] as well as in related areas in statistics [15, 16, 33]. Various versions of the AdaBoost algorithm have proven to be very competitive in terms of prediction accuracy in a variety of applications. Boosting methods have been originally proposed as ensemble methods, see Section 1.1, which rely on the principle of generating multiple predictions and majority voting (averaging) among the individual classifiers.

Later, Breiman [15, 16] made a path-breaking observation that the AdaBoost algorithm can be viewed as a gradient descent algorithm in function space, inspired by numerical optimization and statistical estimation. Moreover, Friedman et al. [33] laid out further important foundations which linked AdaBoost and other boosting algorithms to the framework of statistical estimation and additive basis expansion. Also Mason et al. [62] and Rätsch et al. [70] developed related ideas which were mainly acknowledged in the machine learning community. In Hastie et al. [42], additional views on boosting are given: in particular, they were first in pointing out the relation between boosting and ℓ^1-penalized estimation. The insights of Friedman et al. [33] opened new perspectives, namely to use boosting methods in many other

Keywords and phrases: Generalized linear models, Generalized additive models, Gradient boosting, Survival analysis, Variable selection, Software

contexts than classification. We mention here boosting methods for regression (including generalized regression) [22, 32, 71], for density estimation [73], for survival analysis [45, 71] or for multivariate analysis [33, 59]. In quite a few of these proposals, boosting is not only a black-box prediction tool but also an estimation method for models with a specific structure such as linearity or additivity [18, 22, 45]. Boosting can then be seen as an interesting regularization scheme for estimating a model. This statistical perspective will drive the focus of our exposition of boosting.

We present here some coherent explanations and illustrations of concepts about boosting, some derivations which are novel, and we aim to increase understanding of the methods and selected known results. Besides giving an overview on theoretical concepts of boosting as an algorithm for fitting statistical models, we look at the methodology from a practical point of view as well. The dedicated add-on package **mboost** ["model-based boosting", 43] to the R system for statistical computing [69] implements computational tools which enable the data analyst to compute on the theoretical concepts explained in this paper as close as possible. The illustrations presented throughout the paper focus on three regression problems with continuous, binary and censored response variables, some of them having a large number of covariates. For each example, we only present the most important steps of the analysis. The complete analysis is contained in a *vignette* as part of the **mboost** package (see Appendix A) so that every result shown in this paper is reproducible.

Unless stated differently, we assume that the data are realizations of random variables

$$(X_1, Y_1), \ldots, (X_n, Y_n)$$

from a stationary process with p-dimensional predictor variables X_i and one-dimensional response variables Y_i; for the case of multivariate responses, some references are given in Section 9.1. In particular, the setting above includes independent, identically distributed (*i.i.d.*) observations. The generalization to stationary processes is fairly straightforward: the methods and algorithms are the same as in the *i.i.d.* framework, but the mathematical theory requires more elaborate techniques. Essentially, one needs to ensure that some (uniform) laws of large numbers still hold, e.g., assuming stationary, mixing sequences: some rigorous results are given in [59] and [57].

1.1. *Ensemble schemes: multiple prediction and aggregation.* Ensemble schemes construct multiple function estimates or predictions from re-weighted data and use a linear (or sometimes convex) combination thereof for producing the final, aggregated estimator or prediction.

First, we specify a *base procedure* which constructs a function estimate $\hat{g}(\cdot)$ with values in $\mathbb{R}$, based on some data $(X_1, Y_1), \ldots, (X_n, Y_n)$:

$$(X_1, Y_1), \ldots, (X_n, Y_n) \quad \overset{\text{base procedure}}{\longrightarrow} \quad \hat{g}(\cdot).$$

For example, a very popular base procedure is a regression tree.

Then, generating an ensemble from the base procedures, i.e., an ensemble of function estimates or predictions, works generally as follows:

$$\begin{array}{lll} \text{re-weighted data 1} & \overset{\text{base procedure}}{\longrightarrow} & \hat{g}^{[1]}(\cdot) \\ \text{re-weighted data 2} & \overset{\text{base procedure}}{\longrightarrow} & \hat{g}^{[2]}(\cdot) \\ \ldots & \ldots & \\ \ldots & \ldots & \\ \text{re-weighted data } M & \overset{\text{base procedure}}{\longrightarrow} & \hat{g}^{[M]}(\cdot) \end{array}$$

$$\text{aggregation: } \hat{f}_A(\cdot) = \sum_{m=1}^{M} \alpha_m \hat{g}^{[m]}(\cdot).$$

What is termed here with "re-weighted data" means that we assign individual data weights to every of the n sample points. We have also implicitly assumed that the base procedure allows to do some weighted fitting, i.e., estimation is based on a weighted sample. Throughout the paper (except in Section 1.2), we assume that a base procedure estimate $\hat{g}(\cdot)$ is real-valued (i.e., a regression procedure) making it more adequate for the "statistical perspective" on boosting, in particular for the generic FGD algorithm in Section 2.1.

The above description of an ensemble scheme is too general to be of any direct use. The specification of the data re-weighting mechanism as well as the form of the linear combination coefficients $\{\alpha_m\}_{m=1}^{M}$ are crucial, and various choices characterize different ensemble schemes. Most boosting methods are special kinds of *sequential* ensemble schemes, where the data weights in iteration m depend on the results from the previous iteration $m-1$ only (*memoryless* with respect to iterations $m-2, m-3, \ldots$). Examples of other ensemble schemes include bagging [14] or random forests [1, 17].

1.2. *AdaBoost.* The AdaBoost algorithm for binary classification [31] is the most well known boosting algorithm. The base procedure is a classifier with values in $\{0, 1\}$ (slightly different from a real-valued function estimator as assumed above), e.g., a classification tree.

AdaBoost algorithm

1. Initialize some weights for individual sample points: $w_i^{[0]} = 1/n$ for $i = 1, \ldots, n$. Set $m = 0$.
2. Increase m by 1. Fit the base procedure to the weighted data, i.e., do a weighted fitting using the weights $w_i^{[m-1]}$, yielding the classifier $\hat{g}^{[m]}(\cdot)$.
3. Compute the weighted in-sample misclassification rate

$$\text{err}^{[m]} = \sum_{i=1}^{n} w_i^{[m-1]} I\left(Y_i \neq \hat{g}^{[m]}(X_i)\right) / \sum_{i=1}^{n} w_i^{[m-1]},$$
$$\alpha^{[m]} = \log\left(\frac{1 - \text{err}^{[m]}}{\text{err}^{[m]}}\right),$$

and up-date the weights

$$\tilde{w}_i = w_i^{[m-1]} \exp\left(\alpha^{[m]} I\left(Y_i \neq \hat{g}^{[m]}(X_i)\right)\right),$$
$$w_i^{[m]} = \tilde{w}_i / \sum_{j=1}^{n} \tilde{w}_j.$$

4. Iterate steps 2 and 3 until $m = m_{\text{stop}}$ and build the aggregated classifier by weighted majority voting:

$$\hat{f}_{\text{AdaBoost}}(x) = \underset{y \in \{0,1\}}{\text{argmin}} \sum_{m=1}^{m_{\text{stop}}} \alpha^{[m]} I(\hat{g}^{[m]}(x) = y).$$

By using the terminology m_{stop} (instead of M as in the general description of ensemble schemes), we emphasize here and later that the iteration process should be stopped to avoid overfitting. It is a tuning parameter of AdaBoost which may be selected using some cross-validation scheme.

1.3. *Slow overfitting behavior.* It has been debated until about the year of 2000 whether the AdaBoost algorithm is immune to overfitting when running more iterations, i.e., stopping wouldn't be necessary. It is clear nowadays that AdaBoost and also other boosting algorithms are overfitting eventually, and early stopping (using a value of m_{stop} before convergence of the implementing loss function takes place) is necessary [7, 51, 64]. We emphasize that this is not in contradiction to the experimental results by Breiman [15] where the test set misclassification error still decreases after the training misclassification error is zero (because the training error of the implementing loss function, given in (3.3), is not zero before numerical convergence).

Nevertheless, the AdaBoost algorithm is quite resistant to overfitting (slow overfitting behavior) when increasing the number of iterations m_{stop}.

This has been observed empirically, although some cases with clear overfitting do occur for some datasets [64]. A stream of work has been devoted to develop VC-type bounds for the generalization (out-of-sample) error to explain why boosting is overfitting very slowly only. Schapire et al. [77] prove a remarkable bound for the generalization misclassification error for classifiers in the convex hull of a base procedure. This bound for the misclassification error has been improved by Koltchinskii and Panchenko [53], deriving also a generalization bound for AdaBoost which depends on the number of boosting iterations.

It has been argued [33, rejoinder], [21] that the overfitting resistance (slow overfitting behavior) is much stronger for the misclassification error than many other loss functions such as the (out-of-sample) negative log-likelihood (e.g., squared error in Gaussian regression). Thus, boosting's resistance of overfitting is coupled with a general fact that overfitting is less an issue for classification (i.e., the 0-1 loss function). Furthermore, it is proved in [6] that the misclassification risk can be bounded by the risk of the implementing loss function: it demonstrates from a different perspective that the 0-1 loss can exhibit quite a different behavior than the implementing loss.

Finally, Section 5.1 develops the variance and bias for boosting to fit a one-dimensional curve. Figure 5.1 illustrates the difference between the boosting and the smoothing spline approach, and the eigen-analysis of the boosting approach (see Formula (5.2)) yields the following: boosting's variance increases with exponentially small increments while its squared bias decreases exponentially fast as the number of iterations grow. This also explains why overfitting kicks in very slowly in boosting.

1.4. *Historical remarks.* The idea of boosting as an ensemble method for improving the predictive performance of a base procedure seems to have its root in machine learning. Kearns and Valiant [52] proved that if individual classifiers perform at least slightly better than guessing at random, their predictions can be combined and averaged yielding much better predictions. Later, Schapire [75] proposed a boosting algorithm with provable polynomial run-time to construct such a better ensemble of classifiers. The AdaBoost algorithm [29–31] is considered as a first path-breaking step towards practically feasible boosting algorithms.

The results from Breiman [15, 16], showing that boosting can be interpreted as a functional gradient descent algorithm, uncover older roots of boosting. In the context of regression, there is an immediate connection to the Gauss-Southwell algorithm [79] for solving a linear system of equations (see Section 4.1) and to Tukey's [83] method of "twicing" (see Section 5.1).

2. Functional gradient descent. Breiman [15, 16] showed that the AdaBoost algorithm can be represented as a steepest descent algorithm in function space which we call functional gradient descent (FGD). Friedman et al. [33] and Friedman [32] then developed a more general, beautiful framework which yields a direct interpretation of boosting as a method for function estimation. Consider the problem of estimating a real-valued function

$$f^*(\cdot) = \operatorname*{argmin}_{f(\cdot)} \mathbb{E}[\rho(Y, f(X))], \tag{2.1}$$

where $\rho(\cdot,\cdot)$ is a loss function which is typically assumed to be differentiable and convex with respect to the second argument. For example, the squared error loss $\rho(y,f) = |y-f|^2$ yields the well-known population minimizer $f^*(x) = \mathbb{E}[Y|X=x]$.

2.1. *The generic FGD or boosting algorithm.* In the sequel, FGD and boosting are used as equivalent terminology for the same method or algorithm.

Estimation of $f^*(\cdot)$ in (2.1) with boosting can be done by considering the empirical risk $n^{-1}\sum_{i=1}^n \rho(Y_i, f(X_i))$ and pursuing iterative steepest descent in function space as follows. The following algorithm has been given in a key work by Friedman [32].

Generic FGD algorithm

1. Initialize $\hat{f}^{[0]}(\cdot)$ with an offset value. Common choices are
$$\hat{f}^{[0]}(\cdot) \equiv \operatorname*{argmin}_c n^{-1}\sum_{i=1}^n \rho(Y_i, c)$$
or $\hat{f}^{[0]}(\cdot) \equiv 0$. Set $m = 0$.
2. Increase m by 1. Compute the negative gradient $-\frac{\partial}{\partial f}\rho(Y,f)$ and evaluate at $\hat{f}^{[m-1]}(X_i)$:
$$U_i = -\frac{\partial}{\partial f}\rho(Y,f)|_{f=\hat{f}^{[m-1]}(X_i)},\ i = 1,\ldots,n.$$
3. Fit the negative gradient vector $U_1,\ldots,U_n$ to $X_1,\ldots,X_n$ by the real-valued base procedure (e.g., regression)
$$(X_i, U_i)_{i=1}^n \overset{\text{base procedure}}{\longrightarrow} \hat{g}^{[m]}(\cdot).$$
Thus, $\hat{g}^{[m]}(\cdot)$ can be viewed as an approximation of the negative gradient vector.

4. Up-date $\hat{f}^{[m]}(\cdot) = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}^{[m]}(\cdot)$, where $0 < \nu \leq 1$ is a step-length factor (see below), i.e., proceed along an estimate of the negative gradient vector.
5. Iterate steps 2 to 4 until $m = m_{\text{stop}}$ for some stopping iteration m_{stop}.

The stopping iteration, which is the main tuning parameter, can be determined via cross-validation or some information criterion, see Section 5.4. The choice of the step-length factor ν in step 4 is of minor importance, as long as it is "small" such as $\nu = 0.1$. A smaller value of ν typically requires a larger number of boosting iterations and thus more computing time, while the predictive accuracy has been empirically found to be potentially better and almost never worse when choosing ν "sufficiently small" (e.g., $\nu = 0.1$) [32]. Friedman [32] suggests to use an additional line search between steps 3 and 4 (in case of other loss functions $\rho(\cdot, \cdot)$ than squared error): it yields a slightly different algorithm but the additional line search seems unnecessary for achieving a good estimator $\hat{f}^{[m_{\text{stop}}]}$. The statement is based on empirical evidence and some mathematical reasoning as described at the beginning of Section 7.

2.1.1. *Alternative formulation in function space.* In steps 2 and 3 of the generic FGD algorithm, we associated with $U_1, \ldots, U_n$ a negative gradient vector. A reason for this can be seen from the following formulation in function space which is similar to the exposition in Mason et al. [62] and to the discussion in Ridgeway [72].

Consider the empirical risk functional $C(f) = n^{-1} \sum_{i=1}^n \rho(Y_i, f(X_i))$ and the usual inner product $\langle f, g \rangle = n^{-1} \sum_{i=1}^n f(X_i) g(X_i)$. We can then calculate the negative Gâteaux derivative $dC(\cdot)$ of the functional $C(\cdot)$,

$$-dC(f)(x) = -\frac{\partial}{\partial \alpha} C(f + \alpha \delta_x)|_{\alpha=0}, \quad f : \mathbb{R}^p \to \mathbb{R}, \ x \in \mathbb{R}^p,$$

where δ_x denotes the delta- (or indicator-) function at $x \in \mathbb{R}^p$. In particular, when evaluating the derivative $-dC$ at $\hat{f}^{[m-1]}$ and X_i, we get

$$-dC(\hat{f}^{[m-1]})(X_i) = n^{-1} U_i,$$

with $U_1, ..., U_n$ exactly as in steps 2 and 3 of the generic FGD algorithm. Thus, the negative gradient vector $U_1, \ldots, U_n$ can be interpreted as a functional (Gâteaux) derivative evaluated at the data points.

We point out that the algorithm in Mason et al. [62] is different from the generic FGD method above: while the latter is fitting the negative gradient vector by the base procedure, typically using (nonparametric) least squares, Mason et al. [62] fit the base procedure by maximizing $-\langle U, \hat{g} \rangle =$

$n^{-1}\sum_{i=1}^n U_i\hat{g}(X_i)$. Of course, for certain base procedures, the two algorithms coincide. For example, if $\hat{g}(\cdot)$ is the componentwise linear least squares base procedure described in (4.1), it holds that $n^{-1}\sum_{i=1}^n(U_i - \hat{g}(X_i))^2 = C - \langle U, \hat{g}\rangle$, where $C = n^{-1}\sum_{i=1}^n U_i^2$ is a constant.

3. Some loss functions and boosting algorithms. Various boosting algorithms can now be defined by specifying different loss functions $\rho(\cdot,\cdot)$. The **mboost** package provides infrastructure for defining loss functions via *boost_family* objects, as exemplified below.

3.1. *Binary classification.* For binary classification, the response variable is $Y \in \{0,1\}$ with $\mathbb{P}[Y=1] = p$. Often, it is notationally more convenient to encode the response by $\tilde{Y} = 2Y - 1 \in \{-1,+1\}$ (this coding is used in **mboost** as well). We consider the negative binomial log-likelihood as loss function:

$$-\left(y\log(p) + (1-y)\log(1-p)\right).$$

We parameterize $p = \exp(f)/(\exp(f)+\exp(-f))$ so that $f = \log(p/(1-p))/2$ equals half of the log-odds ratio; the factor $1/2$ is a bit unusual but it will enable that the population minimizer of the loss in (3.1) will be the same as for the exponential loss in (3.3) below. Then, the negative log-likelihood is

$$\log(1+\exp(-2\tilde{y}f)).$$

By scaling, we prefer to use the equivalent loss function

$$\rho_{\text{log-lik}}(\tilde{y}, f) = \log_2(1+\exp(-2\tilde{y}f)), \tag{3.1}$$

which then becomes an upper bound of the misclassification error, see Figure 1. In **mboost**, the negative gradient of this loss function is implemented in a pre-fabricated function `Binomial()` returning an object of class *boost_family* which contains the negative gradient *function* as a slot (assuming a binary response variable `y` $\in \{-1,+1\}$).

The population minimizer can be shown to be [33, cf.]

$$f^*_{\text{log-lik}}(x) = \frac{1}{2}\log\left(\frac{p(x)}{1-p(x)}\right), \quad p(x) = \mathbb{P}[Y=1|X=x].$$

The loss function in (3.1) is a function of $\tilde{y}f$, the so-called margin value, where the function f induces the following classifier for Y:

$$\mathcal{C}(x) = \begin{cases} 1 & \text{if } f(x) > 0 \\ 0 & \text{if } f(x) < 0 \\ \text{undetermined} & \text{if } f(x) = 0. \end{cases}$$

Therefore, a misclassification (including the undetermined case) happens if and only if $\tilde{Y} f(X) \leq 0$. Hence, the misclassification loss is

$$\rho_{0\text{-}1}(y, f) = I_{\{\tilde{y} f \leq 0\}}, \tag{3.2}$$

whose population minimizer is equivalent to the Bayes classifier (for $\tilde{Y} \in \{-1, +1\}$)

$$f^*_{0\text{-}1}(x) = \begin{cases} +1 & \text{if } p(x) > 1/2 \\ -1 & \text{if } p(x) \leq 1/2, \end{cases}$$

where $p(x) = \mathbb{P}[Y = 1|X = x]$. Note that the 0-1 loss in (3.2) cannot be used for boosting or FGD: it is non-differentiable and also non-convex as a function of the margin value $\tilde{y} f$. The negative log-likelihood loss in (3.1) can be viewed as a convex upper approximation of the (computationally intractable) non-convex 0-1 loss, see Figure 1. We will describe in Section 3.3 the BinomialBoosting algorithm (similar to LogitBoost [33]) which uses the negative log-likelihood as implementing loss function.

Another upper convex approximation of the 0-1 loss function in (3.2) is the exponential loss

$$\rho_{\exp}(y, f) = \exp(-\tilde{y} f), \tag{3.3}$$

implemented (with notation $\mathtt{y} \in \{-1, +1\}$) in **mboost** as `AdaExp()` family.

The population minimizer can be shown to be the same as for the log-likelihood loss [33, cf.]:

$$f^*_{\exp}(x) = \frac{1}{2} \log \left(\frac{p(x)}{1 - p(x)} \right), \quad p(x) = \mathbb{P}[Y = 1|X = x].$$

Using functional gradient descent with different implementing loss functions yields different boosting algorithms. When using the log-likelihood loss in (3.1), we obtain LogitBoost [33] or BinomialBoosting from Section 3.3; and with the exponential loss in (3.3), we essentially get AdaBoost [30] from Section 1.2.

We interpret the boosting estimate $\hat{f}^{[m]}(\cdot)$ as an estimate of the population minimizer $f^*(\cdot)$. Thus, the output from AdaBoost, Logit- or BinomialBoosting are estimates of half of the log-odds ratio. In particular, we define probability estimates via

$$\hat{p}^{[m]}(x) = \frac{\exp(\hat{f}^{[m]}(x))}{\exp(\hat{f}^{[m]}(x)) + \exp(-\hat{f}^{[m]}(x))}.$$

The reason for constructing these probability estimates is based on the fact that boosting with a suitable stopping iteration is consistent [7, 51]. Some cautionary remarks about this line of argumentation are presented by Mease et al. [64].

Also very popular in machine learning is the hinge function, the standard loss function for support vector machines:

$$\rho_{\mathrm{SVM}}(y, f) = [1 - \tilde{y}f]_+,$$

where $[x]_+ = xI_{\{x>0\}}$ denotes the positive part. It is also an upper convex bound of the misclassification error, see Figure 1. Its population minimizer is

$$f^*_{\mathrm{SVM}}(x) = \mathrm{sign}(p(x) - 1/2)$$

which is the Bayes classifier for $\tilde{Y} \in \{-1, +1\}$. Since $f^*_{\mathrm{SVM}}(\cdot)$ is a classifier and non-invertible function of $p(x)$, there is no direct way to obtain conditional class probability estimates.

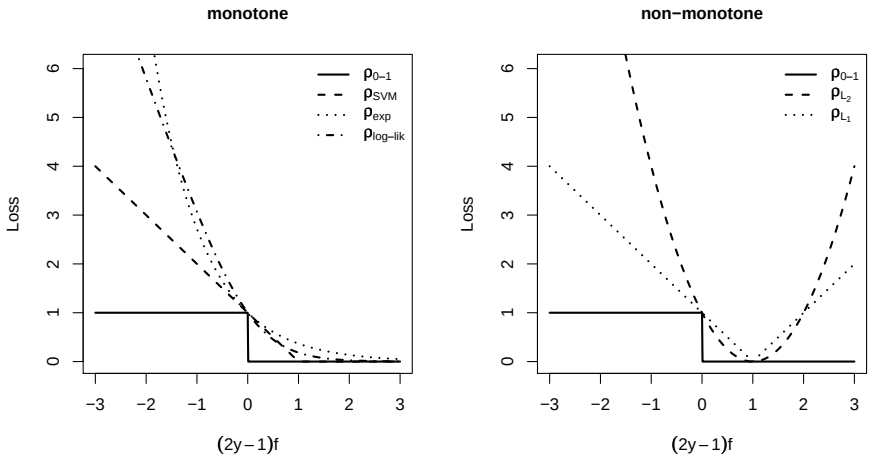

FIG 1. *Losses, as a function of the margin $\tilde{y}f = (2y-1)f$, for binary classification. Left panel with monotone loss functions: 0-1 loss, exponential loss, negative log-likelihood, hinge loss (SVM); right panel with non-monotone loss functions: squared error (L_2) and absolute error (L_1) as in (3.5).*

3.2. *Regression.* For regression with response $Y \in \mathbb{R}$, we use most often the squared error loss (scaled by the factor 1/2 such that the negative

gradient vector equals the residuals, see Section 3.3 below),

$$\rho_{L_2}(y, f) = \frac{1}{2}|y - f|^2 \tag{3.4}$$

with population minimizer

$$f^*_{L_2}(x) = \mathbb{E}[Y|X = x].$$

The corresponding boosting algorithm is L_2Boosting, see Friedman [32] and Bühlmann and Yu [22]. It is described in more detail in Section 3.3. This loss function is available in **mboost** as family `GaussReg()`.

Alternative loss functions which have some robustness properties (with respect to the error distribution, i.e., in "Y-space") include the L_1- and Huber-loss. The former is

$$\rho_{L_1}(y, f) = |y - f|$$

with population minimizer

$$f^*(x) = \text{median}(Y|X = x)$$

and is implemented in **mboost** as `Laplace()`.

Although the L_1-loss is not differentiable at the point $y = f$, we can compute partial derivatives since the single point $y = f$ (usually) has probability zero to be realized by the data. A compromise between the L_1- and L_2-loss is the Huber-loss function from robust statistics:

$$\rho_{\text{Huber}}(y, f) = \begin{cases} |y - f|^2/2, & \text{if } |y - f| \le \delta \\ \delta(|y - f| - \delta/2), & \text{if } |y - f| > \delta \end{cases}$$

which is available in **mboost** as `Huber()`. A strategy for choosing (a changing) δ adaptively has been proposed by Friedman [32]:

$$\delta_m = \text{ median}(\{|Y_i - \hat{f}^{[m-1]}(X_i)|;\ i = 1, \ldots, n\}),$$

where the previous fit $\hat{f}^{[m-1]}(\cdot)$ is used.

3.2.1. *Connections to binary classification.* Motivated from the population point of view, the L_2- or L_1-loss can also be used for binary classification. For $Y \in \{0, 1\}$, the population minimizers are then

$$f^*_{L_2}(x) = \mathbb{E}[Y|X = x] = p(x) = \mathbb{P}[Y = 1|X = x],$$
$$f^*_{L_1}(x) = \text{median}(Y|X = x) = \begin{cases} 1 & \text{if } p(x) > 1/2 \\ 0 & \text{if } p(x) \le 1/2. \end{cases}$$

range spaces	$\rho(y,f)$	$f^*(x)$	algorithm
$y \in \{0,1\}, f \in \mathbb{R}$	$\exp(-(2y-1)f)$	$\frac{1}{2}\log\left(\frac{p(x)}{1-p(x)}\right)$	AdaBoost
$y \in \{0,1\}, f \in \mathbb{R}$	$\log_2(1+e^{-2(2y-1)f})$	$\frac{1}{2}\log\left(\frac{p(x)}{1-p(x)}\right)$	LogitBoost / BinomialBoosting
$y \in \mathbb{R}, f \in \mathbb{R}$	$\frac{1}{2}\lvert y-f\rvert^2$	$\mathbb{E}[Y\lvert X=x]$	L_2Boosting

TABLE 1

Various loss functions $\rho(y,f)$, population minimizers $f^*(x)$ and names of corresponding boosting algorithms; $p(x) = \mathbb{P}[Y=1|X=x]$.

Thus, the population minimizer of the L_1-loss is the Bayes classifier.

Moreover, both L_1- and L_2-loss functions can be parametrized as functions of the margin value $\tilde{y}f$ ($\tilde{y} \in \{-1,+1\}$):

$$\begin{aligned} |\tilde{y}-f| &= |1-\tilde{y}f|, \\ |\tilde{y}-f|^2 &= |1-\tilde{y}f|^2 = (1-2\tilde{y}f+(\tilde{y}f)^2. \end{aligned} \tag{3.5}$$

The L_1- and L_2-loss functions are non-monotone functions of the margin value $\tilde{y}f$, see Figure 1. A negative aspect is that they penalize margin values which are greater than 1: penalizing large margin values can be seen as a way to encourage solutions $\hat{f} \in [-1,1]$ which is the range of the population minimizers $f^*_{L_1}$ and $f^*_{L_2}$ (for $\tilde{Y} \in \{-1,+1\}$) , respectively. However, as discussed below, we prefer to use monotone loss functions.

The L_2-loss for classification (with response variable `y` $\in \{-1,+1\}$) is implemented in `GaussClass()`.

All loss functions mentioned for binary classification (displayed in Figure 1) can be viewed and interpreted from the perspective of proper scoring rules, cf. Buja et al. [24]. We usually prefer the negative log-likelihood loss in (3.1) because: (i) it yields probability estimates; (ii) it is a monotone loss function of the margin value $\tilde{y}f$; (iii) it grows linearly as the margin value $\tilde{y}f$ tends to $-\infty$, unlike the exponential loss in (3.3). The third point reflects a robustness aspect: it is similar to Huber's loss function which also penalizes large values linearly (instead of quadratically as with the L_2-loss).

3.3. *Two important boosting algorithms.* Table 1 summarizes the most popular loss functions and their corresponding boosting algorithms. We now describe the two algorithms appearing in the last two rows of Table 1 in more detail.

3.3.1. *L_2Boosting.* L_2Boosting is the simplest and perhaps most instructive boosting algorithm. It is very useful for regression, in particular when there are very many predictor variables. Applying the general description

of the FGD-algorithm from Section 2.1 to the squared error loss function $\rho_{L_2}(y, f) = |y - f|^2/2$, we obtain the following algorithm.

L_2Boosting algorithm

1. Initialize $\hat{f}^{[0]}(\cdot)$ with an offset value. The default value is $\hat{f}^{[0]}(\cdot) \equiv \overline{Y}$. Set $m = 0$.
2. Increase m by 1. Compute the residuals $U_i = Y_i - \hat{f}^{[m-1]}(X_i)$ for $i = 1, \ldots, n$.
3. Fit the residual vector $U_1, \ldots, U_n$ to $X_1, \ldots, X_n$ by the real-valued base procedure (e.g., regression)

$$(X_i, U_i)_{i=1}^n \overset{\text{base procedure}}{\longrightarrow} \hat{g}^{[m]}(\cdot).$$

4. Up-date $\hat{f}^{[m]}(\cdot) = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}^{[m]}(\cdot)$, where $0 < \nu \leq 1$ is a step-length factor (as in the general FGD-algorithm).
5. Iterate steps 2 to 4 until $m = m_{\text{stop}}$ for some stopping iteration m_{stop}.

The stopping iteration m_{stop} is the main tuning parameter which can be selected using cross-validation or some information criterion as described in Section 5.4.

The derivation from the generic FGD algorithm in Section 2.1 is straightforward. Note that the negative gradient vector becomes the residual vector. Thus, L_2Boosting amounts to refitting residuals multiple times. Tukey [83] recognized this to be useful and proposed "twicing" which is nothing else than L_2Boosting using $m_{\text{stop}} = 2$ (and $\nu = 1$).

3.3.2. *BinomialBoosting: the FGD version of LogitBoost.* We already gave some reasons at the end of Section 3.2.1 why the negative log-likelihood loss function in (3.1) is very useful for binary classification problems. Friedman et al. [33] were first in advocating this, and they proposed LogitBoost which is very similar to the generic FGD algorithm when using the loss from (3.1): the deviation from FGD is the use of Newton's method involving the Hessian matrix (instead of a step-length for the gradient).

For the sake of coherence with the generic functional gradient descent algorithm in Section 2.1, we describe here a version of LogitBoost: to avoid conflicting terminology, we coin it BinomialBoosting.

BinomialBoosting algorithm

Apply the generic FGD algorithm from Section 2.1 using the loss function $\rho_{\text{log-lik}}$ from (3.1). The default offset value is $\hat{f}^{[0]}(\cdot) \equiv \log(\hat{p}/(1 - \hat{p}))/2$, where $\hat{p}$ is the relative frequency of $Y = 1$.

With BinomialBoosting, there is no need that the base procedure is able to do weighted fitting: this constitutes a slight difference to the requirement for LogitBoost [33].

3.4. *Other data structures and models.* Due to the generic nature of boosting or functional gradient descent, we can use the technique in very many other settings. For data with univariate responses and loss functions which are differentiable with respect to the second argument, the boosting algorithm is described in Section 2.1. Survival analysis is an important area of application with censored observations: we describe how one can deal with it in Section 8.

4. Choosing the base procedure. Every boosting algorithm requires the specification of a base procedure. This choice can be driven by the aim of optimizing the predictive capacity only or by considering some structural properties of the boosting estimate in addition. We find the latter usually more interesting as it allows for better interpretation of the resulting model.

We recall that the generic boosting estimator is a sum of base procedure estimates

$$\hat{f}^{[m]}(\cdot) = \nu \sum_{k=1}^{m} \hat{g}^{[k]}(\cdot).$$

Therefore, structural properties of the boosting function estimator are induced by a linear combination of structural characteristics of the base procedure.

The following important examples of base procedures yield useful structures for the boosting estimator $\hat{f}^{[m]}(\cdot)$. The notation is as follows: $\hat{g}(\cdot)$ is an estimate from a base procedure which is based on data $(X_1, U_1), \ldots, (X_n, U_n)$ where $(U_1, \ldots, U_n)$ denotes the current negative gradient. In the sequel, the jth component of a vector c will be denoted by $c^{(j)}$.

4.1. *Componentwise linear least squares for linear models.* Boosting can be very useful for fitting potentially high-dimensional generalized linear models. Consider the base procedure

$$\begin{aligned} &\hat{g}(x) = \hat{\beta}^{(\hat{\mathcal{S}})} x^{(\hat{\mathcal{S}})}, \\ &\hat{\beta}^{(j)} = \sum_{i=1}^{n} X_i^{(j)} U_i / \sum_{i=1}^{n} \left(X_i^{(j)}\right)^2, \ \hat{\mathcal{S}} = \operatorname*{argmin}_{1 \le j \le p} \sum_{i=1}^{n} \left(U_i - \hat{\beta}^{(j)} X_i^{(j)}\right)^2. \end{aligned} \tag{4.1}$$

It selects the best variable in a simple linear model in the sense of ordinary least squares fitting.

When using L_2Boosting with this base procedure, we select in every iteration one predictor variable, not necessarily a different one for each iteration, and we up-date the function linearly:

$$\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \nu \hat{\beta}^{(\hat{\mathcal{S}}_m)} x^{(\hat{\mathcal{S}}_m)},$$

where $\hat{\mathcal{S}}_m$ denotes the index of the selected predictor variable in iteration m. Alternatively, the up-date of the coefficient estimates is

$$\hat{\beta}^{[m]} = \hat{\beta}^{[m-1]} + \nu \cdot \hat{\beta}^{(\hat{\mathcal{S}}_m)}.$$

The notation should be read that only the $\hat{\mathcal{S}}_m$th component of the coefficient estimate $\hat{\beta}^{[m]}$ (in iteration m) has been up-dated. For every iteration m, we obtain a linear model fit. As m tends to infinity, $\hat{f}^{[m]}(\cdot)$ converges to a least squares solution which is unique if the design matrix has full rank $p \leq n$. The method is also known as matching pursuit in signal processing [60], weak greedy algorithm in computational mathematics [81], and it is a Gauss-Southwell algorithm [79] for solving a linear system of equations. We will discuss more properties of L_2Boosting with componentwise linear least squares in Section 5.2.

When using BinomialBoosting with componentwise linear least squares from (4.1), we obtain a fit, including variable selection, of a linear logistic regression model.

As will be discussed in more detail in Section 5.2, boosting typically shrinks the (logistic) regression coefficients towards zero. Usually, we do not want to shrink the intercept term. In addition, we advocate to use boosting on mean centered predictor variables $\tilde{X}_i^{(j)} = X_i^{(j)} - \bar{X^{(j)}}$. In case of a linear model, when centering also the response $\tilde{Y}_i = Y_i - \bar{Y}$, this becomes

$$\tilde{Y}_i = \sum_{j=1}^{p} \beta^{(j)} \tilde{X}_i^{(j)} + \text{ noise}_i$$

which forces the regression surface through the center $(\tilde{x}^{(1)}, \ldots, \tilde{x}^{(p)}, \tilde{y}) = (0, 0, \ldots, 0)$ as with ordinary least squares. Note that it is not necessary to center the response variables when using the default offset value $\hat{f}^{[0]} = \overline{Y}$ in L_2Boosting (for BinomialBoosting, we would center the predictor variables only but never the response, and we would use $\hat{f}^{[0]} \equiv \operatorname{argmin}_c n^{-1} \sum_{i=1}^{n} \rho(Y_i, c)$).

Illustration: Prediction of total body fat. Garcia et al. [34] report on the development of predictive regression equations for body fat content by means of $p = 9$ common anthropometric measurements which were obtained for

$n = 71$ healthy German women. In addition, the women's body composition was measured by Dual Energy X-Ray Absorptiometry (DXA). This reference method is very accurate in measuring body fat but finds little applicability in practical environments, mainly because of high costs and the methodological efforts needed. Therefore, a simple regression equation for predicting DXA measurements of body fat is of special interest for the practitioner. Backward-elimination was applied to select important variables from the available anthropometrical measurements and Garcia et al. [34] report a final linear model utilizing hip circumference, knee breadth and a compound covariate which is defined as the sum of log chin skinfold, log triceps skinfold and log subscapular skinfold:

```
R> bf_lm <- lm(DEXfat ~ hipcirc + kneebreadth + anthro3a,
          data = bodyfat)
R> coef(bf_lm)
```

```
(Intercept)     hipcirc kneebreadth    anthro3a
  -75.23478     0.51153     1.90199     8.90964
```

A simple and easy to communicate regression formula, such as a linear combination of only a few covariates, is of special interest in this application: we employ the `glmboost` function from package **mboost** to fit a linear regression model by means of L_2Boosting with componentwise linear least squares. By default, the function `glmboost` fits a linear model (with initial $m_{\text{stop}} = 100$ and shrinkage parameter $\nu = 0.1$) by minimizing squared error (argument `family = GaussReg()` is the default):

```
R> bf_glm <- glmboost(DEXfat ~ ., data = bodyfat,
          control = boost_control(center = TRUE))
```

Note that, by default, the mean of the response variable is used as an offset in the first step of the boosting algorithm. We center the covariates prior to model fitting in addition. As mentioned above, the special form of the base learner, i.e., componentwise linear least squares, allows for a reformulation of the boosting fit in terms of a linear combination of the covariates which can be assessed via

```
R> coef(bf_glm)
```

```
 (Intercept)          age    waistcirc      hipcirc
    0.000000     0.013602     0.189716     0.351626
elbowbreadth  kneebreadth     anthro3a     anthro3b
   -0.384140     1.736589     3.326860     3.656524
    anthro3c      anthro4
    0.595363     0.000000
attr(,"offset")
```

```
[1] 30.783
```

We notice that most covariates have been used for fitting and thus no extensive variable selection was performed in the above model. Thus, we need to investigate how many boosting iterations are appropriate. Resampling methods such as cross-validation or the bootstrap can be used to estimate the out-of-sample error for a varying number of boosting iterations. The out-of-bootstrap mean squared error for 100 bootstrap samples is depicted in the upper part of Figure 2. The plot leads to the impression that approximately $m_{\text{stop}} = 44$ would be a sufficient number of boosting iterations. In Section 5.4, a corrected version of the Akaike information criterion (AIC) is proposed for determining the optimal number of boosting iterations. This criterion attains its minimum for

```
R> mstop(aic <- AIC(bf_glm))
```

```
[1] 45
```

boosting iterations, see the bottom part of Figure 2 in addition. The coefficients of the boosted linear model with $m_{\text{stop}} = 45$ boosting iterations are

```
R> coef(bf_glm[mstop(aic)])
```

```
 (Intercept)          age    waistcirc      hipcirc
   0.0000000    0.0023271    0.1893046    0.3488781
elbowbreadth  kneebreadth     anthro3a     anthro3b
   0.0000000    1.5217686    3.3268603    3.6051548
    anthro3c       anthro4
   0.5043133    0.0000000
attr(,"offset")
[1] 30.783
```

and thus 7 covariates have been selected for the final model (intercept equal to zero occurs here for mean centered response and predictors and hence, $n^{-1}\sum_{i=1}^{n} Y_i = 30.783$ is the intercept in the uncentered model). Note that the variables `hipcirc`, `kneebreadth` and `anthro3a`, which we have used for fitting a linear model at the beginning of this paragraph, have been selected by the boosting algorithm as well.

4.2. *Componentwise smoothing spline for additive models.* Additive and generalized additive models, introduced by Hastie and Tibshirani [40] (see also [41]), have become path-breaking and very popular for adding more flexibility to the linear structure in generalized linear models. Such flexibility can also be added in boosting (whose framework is especially useful for high-dimensional problems).

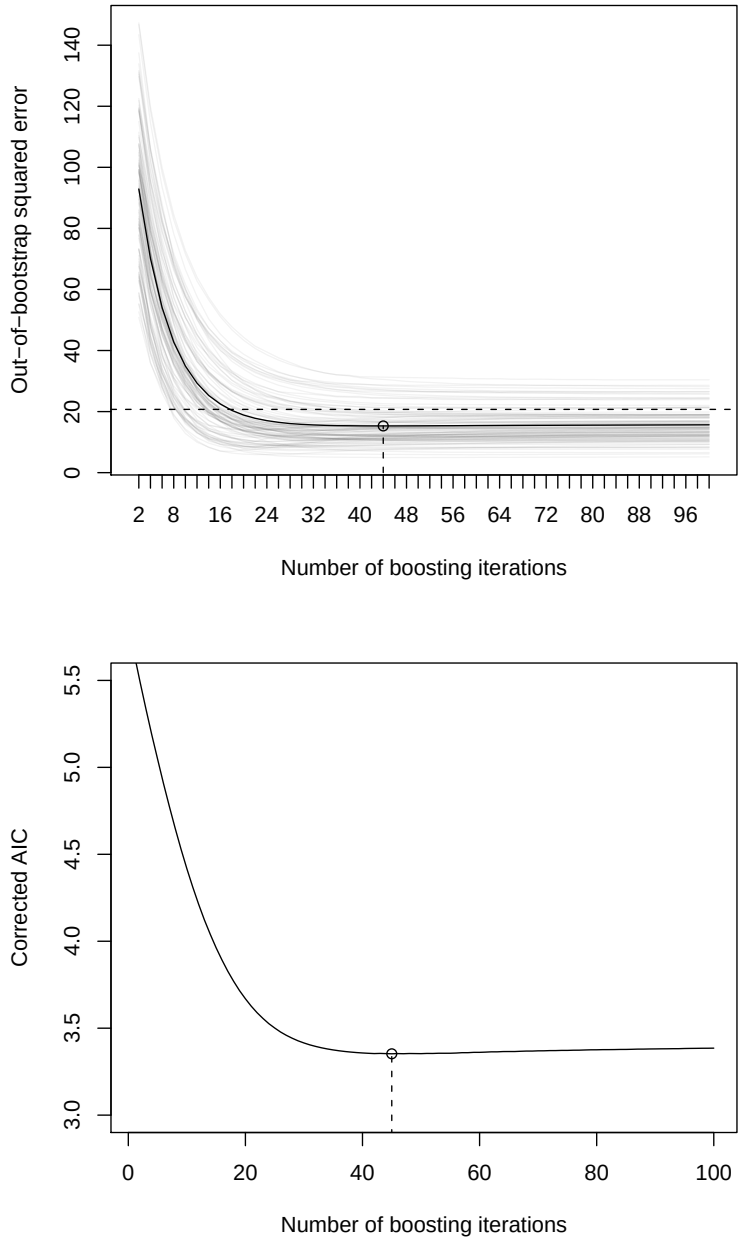

FIG 2. *bodyfat data: Out-of-bootstrap squared error for varying number of boosting iterations* m_{stop} *(top). The dashed horizontal line depicts the average out-of-bootstrap error of the linear model for the pre-selected variables* `hipcirc`, `kneebreadth` *and* `anthro3a` *fitted via ordinary least squares. The lower part show the corrected AIC criterion.*

We can choose use a nonparametric base procedure for function estimation. Suppose that

$$\begin{aligned}&\hat{f}^{(j)}(\cdot) \text{ is a least squares smoothing spline estimate based on}\\ &U_1,\ldots,U_n \text{ against } X_1^{(j)},\ldots,X_n^{(j)} \text{ with fixed degrees of freedom df.}\end{aligned} \tag{4.2}$$

That is,

$$\hat{f}^{(j)}(\cdot) = \operatorname*{argmin}_{f(\cdot)} \sum_{i=1}^{n} \left(U_i - f\left(X_i^{(j)}\right)\right)^2 + \lambda \int (f''(x))^2 dx, \tag{4.3}$$

where $\lambda > 0$ is a tuning parameter such that the trace of the corresponding hat matrix equals df. For further details, we refer to Green and Silverman [36].

The base procedure is then

$$\begin{aligned}&\hat{g}(x) = \hat{f}^{(\hat{\mathcal{S}})}(x^{(\hat{\mathcal{S}})}),\\ &\hat{f}^{(j)}(\cdot) \text{ as above and } \hat{\mathcal{S}} = \operatorname*{argmin}_{1\le j\le p} \sum_{i=1}^{n} \left(U_i - \hat{f}^{(j)}(X_i^{(j)})\right)^2,\end{aligned}$$

where the degrees of freedom df is the same for all $\hat{f}^{(j)}(\cdot)$.

L_2Boosting with componentwise smoothing splines yields an additive model, including variable selection, i.e., a fit which is additive in the predictor variables. This can be seen immediately since L_2Boosting proceeds additively for up-dating the function $\hat{f}^{[m]}(\cdot)$, see Section 3.3. We can normalize to obtain the following additive model estimator:

$$\begin{aligned}&\hat{f}^{[m]}(x) = \hat{\mu} + \sum_{j=1}^{p} \hat{f}^{[m],(j)}\left(x^{(j)}\right),\\ &n^{-1}\sum_{i=1}^{n} \hat{f}^{[m],(j)}\left(X_i^{(j)}\right) = 0 \text{ for all } j = 1,\ldots,p.\end{aligned}$$

As with the componentwise linear least squares base procedure, we can use componentwise smoothing splines also in BinomialBoosting, yielding an additive logistic regression fit.

The degrees of freedom in the smoothing spline base procedure should be chosen "small" such as df $= 4$. This yields low variance but typically large bias of the base procedure. The bias can then be reduced by additional boosting iterations. This choice of low variance but high bias has been analyzed in Bühlmann and Yu [22], see also Section 4.4.

Componentwise smoothing splines can be generalized to pairwise smoothing splines which searches for and fits the best pairs of predictor variables such that a smooth of $U_1, \ldots, U_n$ against this pair of predictors reduces the residual sum of squares most. With L_2Boosting, this yields a nonparametric model fit with first order interaction terms. The procedure has been empirically demonstrated to be often much better than fitting with MARS [23].

Illustration: Prediction of total body fat (cont.). Being more flexible than the linear model which we fitted to the `bodyfat` data in Section 4.1, we estimate an additive model using the `gamboost` function from **mboost** (first with pre-specified $m_{\text{stop}} = 100$ boosting iterations, $\nu = 0.1$ and squared error loss):

```
R> bf_gam <- gamboost(DEXfat ~ ., data = bodyfat)
```

The degrees of freedom for the smoothing splines, which are utilized as base learners here, can be defined by the `dfbase` argument, defaulting to 4.

We can estimate the number of boosting iterations m_{stop} using the corrected AIC criterion described in Section 5.4 via

```
R> mstop(aic <- AIC(bf_gam))
```

```
[1] 46
```

Similar to the linear regression model, the partial contributions of the covariates can be extracted from the boosting fit. For the most important variables, the partial fits are given in Figure 3 showing some slight non-linearity, mainly for `kneebreadth`.

4.3. *Trees.* In the machine learning community, regression trees are the most popular base procedures. They have the advantage to be invariant under monotone transformations of predictor variables, i.e., we do not need to search for good data transformations. Moreover, regression trees handle covariates measured at different scales (continuous, ordinal or nominal variables) in a unified, and potentially unbiased [47], way.

When using stumps, i.e., a tree with two terminal nodes only, the boosting estimate will be an additive model in the original predictor variables, because every stump-estimate is a function of a single predictor variable only. Similarly, boosting trees with (at most) d terminal nodes results in a nonparametric model having at most interactions of order $d - 2$. Therefore, if we want to constrain the degree of interactions, we can easily do this by constraining the (maximal) number of nodes in the base procedure.

Illustration: Prediction of total body fat (cont.). Both the **gbm** package [74] and the **mboost** package are helpful when decision trees are to be used as

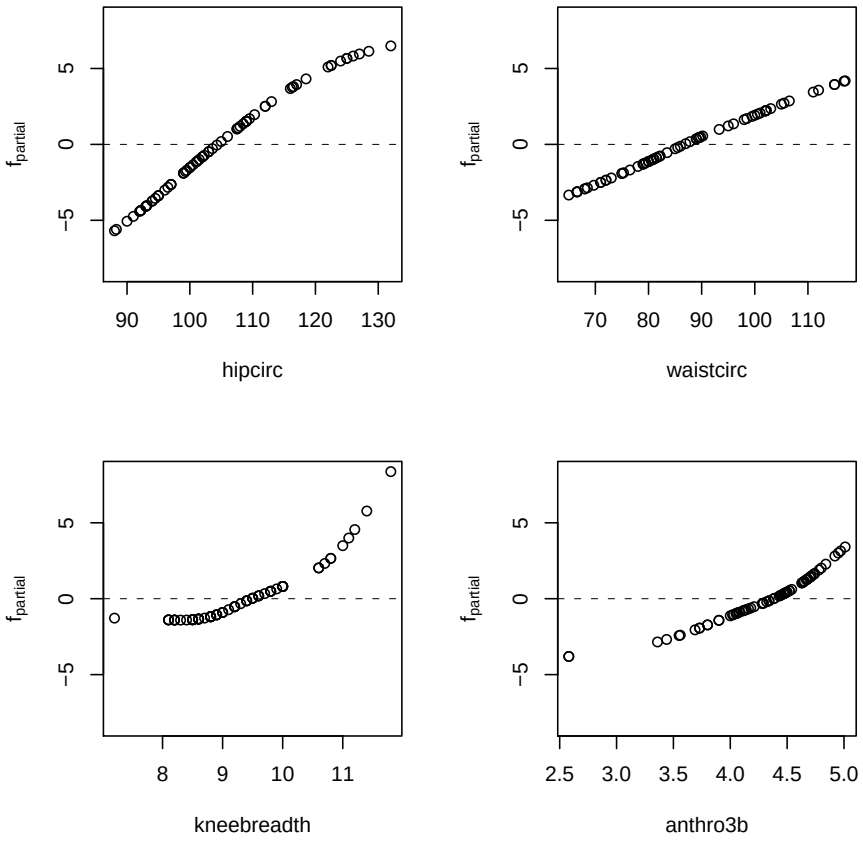

FIG 3. *bodyfat data: Partial contributions of four covariates in an additive model (without centering of estimated functions to mean zero).*

base learners. In **mboost**, the function `blackboost` implements boosting for fitting such classical *black-box* models:

```
R> bf_black <- blackboost(DEXfat ~ ., data = bodyfat,
         control = boost_control(mstop = 500))
```

Conditional inference trees [47] as available from the **party** package [46] are utilized as base learners. Here, the function `boost_control` defines the number of boosting iterations `mstop`.

Alternatively, we can use the function `gbm` from the **gbm** package which yields roughly the same fit as can be seen from Figure 4.

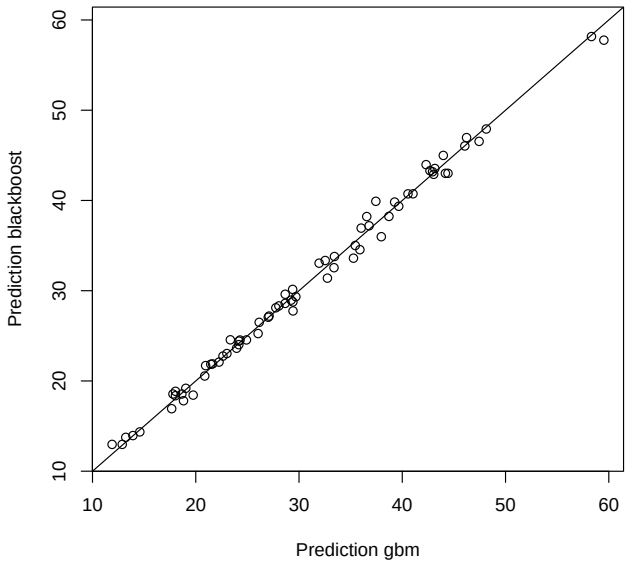

FIG 4. *bodyfat data: Fitted values of both the* **gbm** *and* **mboost** *implementations of L_2Boosting with different regression trees as base learners.*

4.4. *The low-variance principle.* We have seen above that the structural properties of a boosting estimate are determined by the choice of a base procedure. In our opinion, the structure specification should come first. After having made a choice, the question becomes how "complex" the base procedure should be. For example, how should we choose the degrees of freedom for the componentwise smoothing spline in (4.2)? A general answer is: choose the base procedure (having the desired structure) with low variance at the price of larger estimation bias. For the componentwise smoothing splines, this would imply a low number of degrees of freedom, e.g., df $= 4$.

We give some reasons for the low-variance principle in Section 5.1 (Replica 1). Moreover, it has been demonstrated in Friedman [32] that a small step-size factor ν can be often beneficial and almost never yields substantially worse predictive performance of boosting estimates. Note that a small step-size factor can be seen as a shrinkage of the base procedure by the factor ν, implying low variance but potentially large estimation bias.

5. L_2Boosting. L_2Boosting is functional gradient descent using the squared error loss which amounts to repeated fitting of ordinary residuals,

as described already in Section 3.3.1. Here, we aim at increasing the understanding of the simple L_2Boosting algorithm. We first start with a toy problem of curve estimation, and we will then illustrate concepts and results which are especially useful for high-dimensional data. These can serve as heuristics for boosting algorithms with other convex loss functions for problems in e.g., classification or survival analysis.

5.1. *Nonparametric curve estimation: from basics to asymptotic optimality.* Consider the toy problem of estimating a regression function $\mathbb{E}[Y|X = x]$ with one-dimensional predictor $X \in \mathbb{R}$ and a continuous response $Y \in \mathbb{R}$.

Consider the case with a linear base procedure having a hat matrix $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}^n$, mapping the response variables $\mathbf{Y} = (Y_1, \ldots, Y_n)^\top$ to their fitted values $(\hat{f}(X_1), \ldots, \hat{f}(X_n))^\top$. Examples include nonparametric kernel smoothers or smoothing splines. It is easy to show that the hat matrix of the L_2Boosting fit (for simplicity, with $\hat{f}^{[0]} \equiv 0$ and $\nu = 1$) in iteration m equals:

$$\mathcal{B}_m = \mathcal{B}_{m-1} + \mathcal{H}(I - \mathcal{B}_{m-1}) = I - (I - \mathcal{H})^m. \tag{5.1}$$

Formula (5.1) allows for several insights. First, if the base procedure satisfies $\|I - \mathcal{H}\| < 1$ for a suitable norm, i.e., has a "learning capacity" such that the residual vector is shorter than the input-response vector, we see that $\mathcal{B}_m$ converges to the identity I as $m \to \infty$, and $\mathcal{B}_m\mathbf{Y}$ converges to the fully saturated model $\mathbf{Y}$, interpolating the response variables exactly. Thus, we see here explicitly that we have to stop early with the boosting iterations in order to prevent over-fitting.

When specializing to the case of a smoothing spline base procedure (cf. Formula (4.3)), it is useful to invoke some eigen-analysis. The spectral representation is

$$\mathcal{H} = UDU^\top, \quad U^\top U = UU^\top = I, \quad D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n),$$

where $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$ denote the (ordered) eigenvalues of $\mathcal{H}$. It then follows with (5.1) that

$$\begin{aligned} &\mathcal{B}_m = UD_mU^\top, \\ &D_m = \mathrm{diag}(d_{1,m}, \ldots, d_{n,m}), \quad d_{i,m} = 1 - (1 - \lambda_i)^m. \end{aligned}$$

It is well known that a smoothing spline satisfies:

$$\lambda_1 = \lambda_2 = 1, \quad 0 < \lambda_i < 1 \ (i = 3, \ldots, n).$$

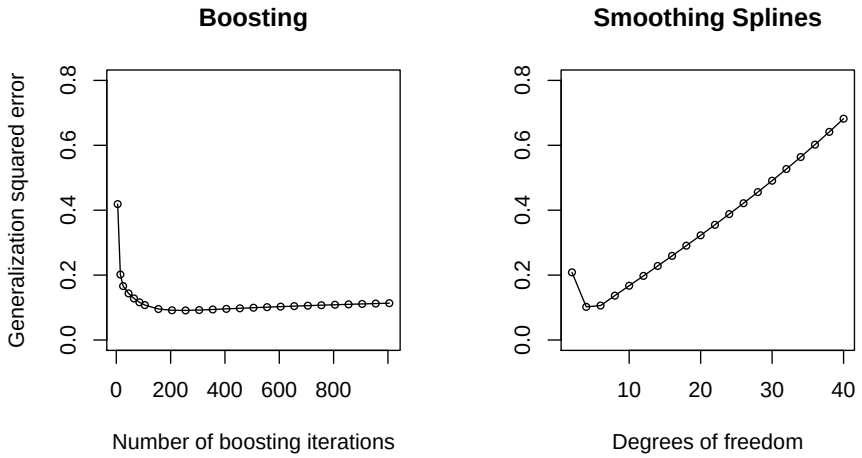

FIG 5. *Mean squared prediction error* $\mathbb{E}[(f(X) - \hat{f}(X))^2]$ *for the regression model* $Y_i = 0.8X_i + \sin(6X_i) + \varepsilon_i$ $(i = 1, \ldots, n = 100)$, *with* $\varepsilon \sim \mathcal{N}(0, 2), X_i \sim \mathcal{U}(-1/2, 1/2)$, *averaged over* 100 *simulation runs. Left:* L_2*Boosting with smoothing spline base procedure (having fixed degrees of freedom* $df = 4$*) and using* $\nu = 0.1$*, for varying number of boosting iterations. Right: single smoothing spline with varying degrees of freedom.*

Therefore, the eigenvalues of the boosting hat operator (matrix) in iteration m satisfy:

$$d_{1,m} \equiv d_{2,m} \equiv 1 \text{ for all m}, \tag{5.2}$$

$$0 < d_{i,m} = 1 - (1 - \lambda_i)^m < 1 \ (i = 3, \ldots, n), \ d_{i,m} \to 1 \ (m \to \infty). \tag{5.3}$$

When comparing the spectrum, i.e., the set of eigenvalues, of a smoothing spline with its boosted version, we see the following. For both cases, the largest two eigenvalues are equal to 1. Moreover, all other eigenvalues can be changed by either varying the degrees of freedom $\mathrm{df} = \sum_{i=1}^n \lambda_i$ in a single smoothing spline, or by varying the boosting iteration m with some fixed (low-variance) smoothing spline base procedure having fixed (low) values λ_i. In Figure 5 we demonstrate the difference between the two approaches for changing "complexity" of the estimated curve fit by means of a toy example first shown in [22]. Both methods have about the same minimum mean squared error but L_2Boosting overfits much more slowly than a single smoothing spline.

By careful inspection of the eigen-analysis for this simple case of boosting a smoothing spline, Bühlmann and Yu [22] proved an asymptotic minimax rate result:

REPLICA 1. *[22] When stopping the boosting iterations appropriately, i.e., $m_{stop} = m_n = O(n^{4/(2\xi+1)})$, $m_n \to \infty$ $(n \to \infty)$ with $\xi \geq 2$ as below, L_2Boosting with cubic smoothing splines having* fixed *degrees of freedom achieves the minimax convergence rate over Sobolev function classes of smoothness degree $\xi \geq 2$, as $n \to \infty$.*

Two items are interesting. First, minimax rates are achieved by using a base procedure with fixed degrees of freedom which means low variance from an asymptotic perspective. Secondly, L_2Boosting with cubic smoothing splines has the capability to *adapt* to higher order smoothness of the true underlying function: thus, with the stopping iteration as the one and only tuning parameter, we can nevertheless adapt to any higher-order degree of smoothness (without the need of choosing a higher order spline base procedure).

Recently, asymptotic convergence and minimax rate results have been established for early-stopped boosting in more general settings [10, 90].

5.1.1. *L_2Boosting using kernel estimators.* As we have pointed out in Replica 1, L_2Boosting of smoothing splines can achieve faster mean squared error convergence rates than the classical $O(n^{-4/5})$, assuming that the true underlying function is sufficiently smooth. We illustrate here a related phenomenon with kernel estimators.

We consider fixed, univariate design points $x_i = i/n$ $(i = 1, \ldots, n)$ and the Nadaraya-Watson kernel estimator for the nonparametric regression function $\mathbb{E}[Y|X = x]$:

$$\hat{g}(x; h) = (nh)^{-1} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) Y_i = n^{-1} \sum_{i=1}^{n} K_h(x - x_i) Y_i,$$

where $h > 0$ is the bandwidth, $K(\cdot)$ a kernel in the form of a probability density which is symmetric around zero and $K_h(x) = h^{-1}K(x/h)$. It is straightforward to derive the form of L_2Boosting using $m = 2$ iterations (with $\hat{f}^{[0]} \equiv 0$ and $\nu = 1$), i.e., twicing [83], with the Nadaraya-Watson kernel estimator:

$$\hat{f}^{[2]}(x) = (nh)^{-1} \sum_{i=1}^{n} K_h^{tw}(x - x_i) Y_i, \;\; K_h^{tw}(u) = 2K_h(u) - K_h * K_h(u),$$

$$\text{where } K_h * K_h(u) = n^{-1} \sum_{r=1}^{n} K_h(u - x_r) K_h(x_r).$$

For fixed design points $x_i = i/n$, the kernel $K_h^{tw}(\cdot)$ is asymptotically equivalent to a higher-order kernel (which can take negative values) yielding a

squared bias term of order $O(h^8)$, assuming that the true regression function is four times continuously differentiable. Thus, twicing or L_2Boosting with $m = 2$ iterations amounts to be a Nadaraya-Watson kernel estimator with a higher-order kernel. This explains from another angle why boosting is able to improve the mean squared error rate of the base procedure. More details including also non-equispaced designs are given in DiMarzio and Taylor [27].

5.2. *L_2Boosting for high-dimensional linear models.* Consider a potentially high-dimensional linear model

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta^{(j)} X_i^{(j)} + \varepsilon_i, \ i = 1, \ldots, n, \tag{5.4}$$

where $\varepsilon_1, \ldots, \varepsilon_n$ are i.i.d. with $\mathbb{E}[\varepsilon_i] = 0$ and independent from all X_i's. We allow for the number of predictors p to be much larger than the sample size n. The model encompasses the representation of a noisy signal by an expansion with an over-complete dictionary of functions $\{g^{(j)}(\cdot) : j = 1, \ldots, p\}$; e.g., for surface modeling with design points in $Z_i \in \mathbb{R}^2$,

$$Y_i = f(Z_i) + \varepsilon_i, \quad f(z) = \sum_j \beta^{(j)} g^{(j)}(z) \ (z \in \mathbb{R}^2).$$

Fitting the model (5.4) can be done using L_2Boosting with the componentwise linear least squares base procedure from Section 4.1 which fits in every iteration the best predictor variable reducing the residual sum of squares most. This method has the following basic properties:

1. As the number m of boosting iterations increases, the L_2Boosting estimate $\hat{f}^{[m]}(\cdot)$ converges to a least squares solution. This solution is unique if the design matrix has full rank $p \leq n$.
2. When stopping early which is usually needed to avoid over-fitting, the L_2Boosting method often does variable selection.
3. The coefficient estimates $\hat{\beta}^{[m]}$ are (typically) shrunken versions of a least squares estimate $\hat{\beta}_{\mathrm{OLS}}$, related to the Lasso as described in Section 5.2.1.

Illustration: Breast cancer subtypes. Variable selection is especially important in high-dimensional situations. As an example, we study a binary classification problem involving $p = 7129$ gene expression levels in $n = 49$ breast cancer tumor samples [data taken from 89]. For each sample, a binary response variable describes the lymph node status (25 negative and 24 positive).

The data are stored in form of an *exprSet* object `westbc` [see 35] and we first extract the matrix of expression levels and the response variable:

```
R> x <- t(exprs(westbc))
R> y <- pData(westbc)$nodal.y
```

We aim at using L_2Boosting for classification, see Section 3.2.1, with classical AIC based on the binomial log-likelihood. Thus, we first transform the factor `y` to a numeric variable with 0/1 coding:

```
R> yfit <- as.numeric(y) - 1
```

The general framework implemented in **mboost** allows us to specify the negative gradient corresponding to the implementing loss function (the `ngradient` argument), here the squared error loss, and a different evaluating loss function (the `loss` argument), here the negative binomial log-likelihood, with the `Family` function as follows:

```
R> rho <- function(y, f, w = 1) {
         p <- pmax(pmin(1 - 1e-05, f), 1e-05)
         -y * log(p) - (1 - y) * log(1 - p)
     }
R> ngradient <- function(y, f, w = 1) y - f
R> offset <- function(y, w) weighted.mean(y, w)
R> L2fm <- Family(ngradient = ngradient, loss = rho,
         offset = offset)
```

The resulting object can now be passed to the `glmboost` function for boosting with componentwise linear least squares (here initial $m_{\text{stop}} = 200$ iterations are used):

```
R> ctrl <- boost_control(mstop = 200, center = TRUE)
R> west_glm <- glmboost(x, yfit, family = L2fm, control = ctrl)
```

Fitting such a linear model to $p = 7129$ covariates for $n = 49$ observations takes about 2.4 seconds on a medium scale desktop computer (Intel Pentium 4, 2.8GHz). Thus, this form of estimation and variable selection is computationally very efficient. As a comparison, computing all Lasso solutions, using the **lars** [28, 39] in R (with `use.Gram=FALSE`), takes about 6 seconds.

The question how to choose m_{stop} can be addressed by the classical AIC criterion as follows

```
R> aic <- AIC(west_glm, method = "classical")
R> mstop(aic)
```

```
[1] 100
```

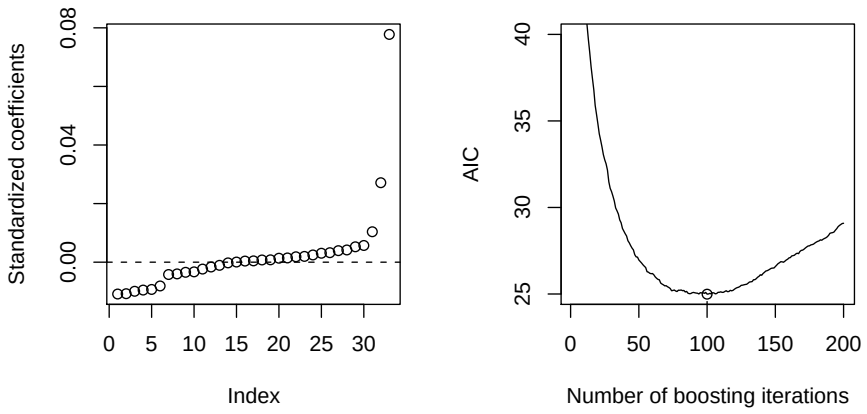

FIG 6. *westbc data: Standardized regression coefficients* $\hat{\beta}^{(j)}\sqrt{\widehat{Var}(X^{(j)})}$ *(left panel) for* $m_{stop} = 100$ *determined from the classical AIC criterion shown in the right panel.*

where the AIC is computed as -2(log-likelihood) + 2(degrees of freedom) = 2 (evaluating loss) + 2(degrees of freedom), see Formula (5.8). The notion of degrees of freedom is discussed in Section 5.3.

Figure 6 shows the AIC curve depending on the number of boosting iterations. When we stop after $m_{\text{stop}} = 100$ boosting iterations, we obtain 33 genes with non-zero regression coefficients whose standardized values $\hat{\beta}^{(j)}\sqrt{\widehat{\text{Var}}(X^{(j)})}$ are depicted in the left panel of Figure 6.

Of course, we could also use BinomialBoosting for analyzing the data: the computational CPU time would be of the same order of magnitude, i.e. 1 - 3 seconds.

5.2.1. *Connections to the Lasso.* Hastie et al. [42] first pointed out an intriguing connection between L_2Boosting with componentwise linear least squares and the Lasso [82] which is the following ℓ^1-penalty method:

$$(5.5)\hat{\beta}(\lambda) = \operatorname*{argmin}_{\beta} n^{-1}\sum_{i=1}^{n}\left(Y_i - \beta_0 - \sum_{j=1}^{p}\beta^{(j)}X_i^{(j)}\right)^2 + \lambda\sum_{j=1}^{p}|\beta^{(j)}|.$$

Efron et al. [28] made the connection rigorous and explicit: they consider a version of L_2Boosting, called forward stagewise linear regression (FSLR), and they show that FSLR with infinitesimally small step-sizes (i.e., the value

ν in step 4 of the L_2Boosting algorithm in Section 3.3.1) produces a set of solutions which is approximately equivalent to the set of Lasso solutions when varying the regularization parameter λ in Lasso (see (5.5) above). The approximate equivalence is derived by representing FSLR and Lasso as two different modifications of the computationally efficient least angle regression (LARS) algorithm from Efron et al. [28] (see also [68] for generalized linear models). The latter is very similar to the algorithm proposed earlier by Osborne et al. [67]. In special cases where the design matrix satisfies a "positive cone condition", FSLR, Lasso and LARS all coincide [28, p.425]. For more general situations, when adding some backward steps to boosting, such modified L_2Boosting coincides with the Lasso (Zhao and Yu [92]).

Despite the fact that L_2Boosting and Lasso are not equivalent methods in general, it may be useful to interpret boosting as being "related" to ℓ^1-penalty based methods.

5.2.2. *Asymptotic consistency in high dimensions.* We review here a result establishing asymptotic consistency for very high-dimensional but sparse linear models as in (5.4). To capture the notion of high-dimensionality, we equip the model with a dimensionality $p = p_n$ which is allowed to grow with sample size n; moreover, the coefficients $\beta^{(j)} = \beta_n^{(j)}$ are now potentially depending on n and the regression function is denoted by $f_n(\cdot)$.

REPLICA 2. *[18] Consider the linear model in (5.4). Assume that* $p_n = O(\exp(n^{1-\xi}))$ *for some* $0 < \xi \le 1$ *(high-dimensionality) and* $\sup_{n\in\mathbb{N}} \sum_{j=1}^{p_n} |\beta_n^{(j)}| < \infty$ *(sparseness of the true regression function w.r.t. the* ℓ^1*-norm); moreover, the variables* $X_i^{(j)}$ *are bounded and* $\mathbb{E}[|\varepsilon_i|^{4/\xi}] < \infty$*. Then: when stopping the boosting iterations appropriately, i.e.,* $m = m_n \to \infty$ $(n \to \infty)$ *sufficiently slowly,* L_2*Boosting with componentwise linear least squares satisfies*

$$\mathbb{E}_{X_{new}}[(\hat{f}_n^{[m_n]}(X_{new}) - f_n(X_{new}))^2] \to 0 \text{ in probability } (n \to \infty),$$

where X_{new} *denotes new predictor variables, independent of and with the same distribution as the* X*-component of the data* (X_i, Y_i) $(i = 1, \ldots, n)$.

The result holds for almost arbitrary designs and no assumptions about collinearity or correlations are required. Replica 2 identifies boosting as a method which is able to consistently estimate a very high-dimensional but sparse linear model; for the Lasso in (5.5), a similar result holds as well [37]. In terms of empirical performance, there seems to be no overall superiority of L_2Boosting over Lasso or vice-versa.

5.2.3. *Transforming predictor variables.* In view of Replica 2, we may enrich the design matrix in model (5.4) with many transformed predictors: if the true regression function can be represented as a sparse linear combination of original or transformed predictors, consistency is still guaranteed. It should be noted though that the inclusion of non-effective variables in the design matrix does degrade the finite-sample performance to a certain extent.

For example, higher order interactions can be specified in generalized AN(C)OVA models and L_2Boosting with componentwise linear least squares can be used to select a small number out of potentially many interaction terms.

As an option for continuously measured covariates, we may utilize a B-spline basis as illustrated in the next paragraph. We emphasize that during the process of L_2Boosting with componentwise linear least squares, individual spline basis functions from various predictor variables are selected and fitted one at a time; in contrast, L_2Boosting with componentwise smoothing splines fits a whole smoothing spline function (for a selected predictor variable) at a time.

Illustration: Prediction of total body fat (cont.). Such transformations and estimation of a corresponding linear model can be done with the `glmboost` function, where the model formula performs the computations of all transformations by means of the `bs` (B-spline basis) function from package **splines**. First, we set up a formula transforming each covariate

```
R> bsfm
```

```
DEXfat ~ bs(age) + bs(waistcirc) + bs(hipcirc) + bs(elbowbreadth) +
    bs(kneebreadth) + bs(anthro3a) + bs(anthro3b) + bs(anthro3c) +
    bs(anthro4)
```

and then fit the complex linear model by using the `glmboost` function with initial $m_{\text{stop}} = 5000$ boosting iterations:

```
R> ctrl <- boost_control(mstop = 5000)
R> bf_bs <- glmboost(bsfm, data = bodyfat, control = ctrl)
R> mstop(aic <- AIC(bf_bs))
```

```
[1] 2891
```

The corrected AIC criterion (see Section 5.4) suggests to stop after $m_{\text{stop}} = 2891$ boosting iterations and the final model selects 21 (transformed) predictor variables. Again, the partial contributions of each of the 9 original covariates can be computed easily and are shown in Figure 7 (for the same variables as in Figure 3). Note that the depicted functional relationship de-

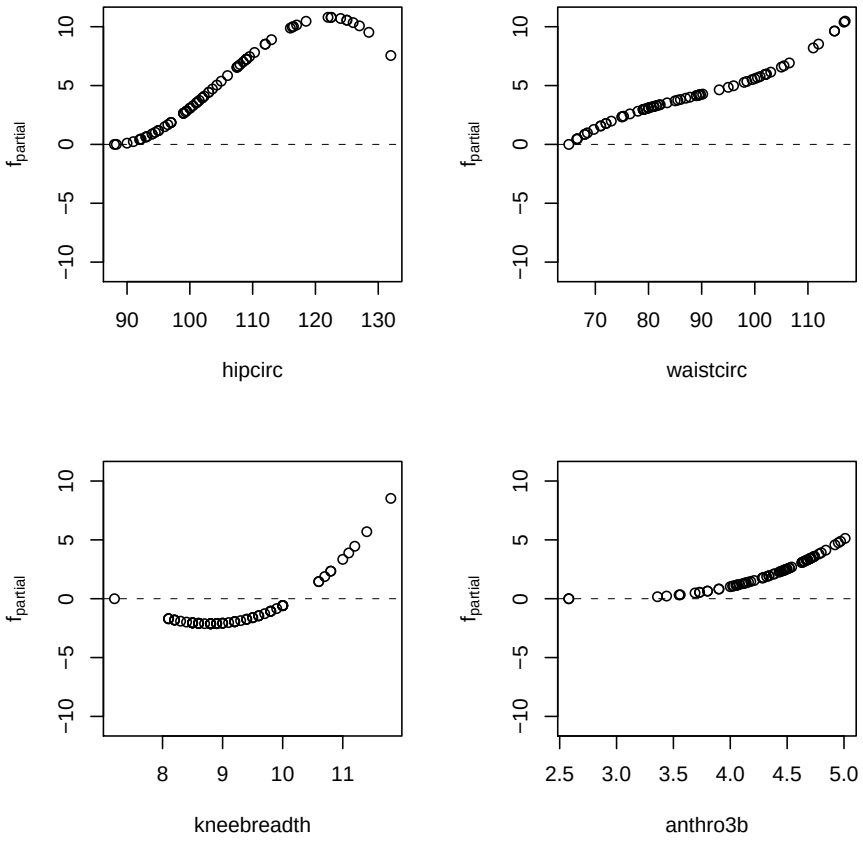

FIG 7. *bodyfat data: Partial fits for a linear model fitted to transformed covariates using B-splines (without centering of estimated functions to mean zero).*

rived from the model fitted above (Figure 7) is qualitatively the same as the one derived from the additive model (Figure 3).

5.3. *Degrees of freedom for L_2Boosting.* A notion of degrees of freedom will be useful for estimating the stopping iteration of boosting (Section 5.4).

5.3.1. *Componentwise linear least squares.* We consider L_2Boosting with componentwise linear least squares. Denote by

$$\mathcal{H}^{(j)} = \mathbf{X}^{(j)}(\mathbf{X}^{(j)})^\top / \|\mathbf{X}^{(j)}\|^2, \ j = 1, \ldots, p,$$

the $n \times n$ hat matrix for the linear least squares fitting operator using the jth predictor variable $\mathbf{X}^{(j)} = (X_1^{(j)}, \ldots, X_n^{(j)})^\top$ only; $\|x\|^2 = x^\top x$ denotes the Euclidean norm for a vector $x \in \mathbb{R}^n$. The hat matrix of the componentwise linear least squares base procedure (see (4.1)) is then

$$\mathcal{H}^{(\hat{\mathcal{S}})} : \ (U_1, \ldots, U_n) \mapsto \hat{U}_1, \ldots, \hat{U}_n,$$

where $\hat{\mathcal{S}}$ is as in (4.1). Similarly to (5.1), we then obtain the hat matrix of L_2Boosting in iteration m:

$$\begin{aligned} \mathcal{B}_m &= \mathcal{B}_{m-1} + \nu \cdot \mathcal{H}^{(\hat{\mathcal{S}}_m)}(I - \mathcal{B}_{m-1}) \\ &= I - (I - \nu \mathcal{H}^{(\hat{\mathcal{S}}_m)})(I - \nu \mathcal{H}^{(\hat{\mathcal{S}}_{m-1})}) \cdots (I - \nu \mathcal{H}^{(\hat{\mathcal{S}}_1)}), \end{aligned} \tag{5.6}$$

where $\hat{\mathcal{S}}_r \in \{1, \ldots, p\}$ denotes the component which is selected in the componentwise least squares base procedure in the rth boosting iteration. We emphasize that $\mathcal{B}_m$ is depending on the response variable Y via the selected components $\hat{\mathcal{S}}_r, \ r = 1, \ldots m$. Due to this dependence on Y, $\mathcal{B}_m$ should be viewed as an approximate hat matrix only. Neglecting the selection effect of $\hat{\mathcal{S}}_r$ $(r = 1, \ldots m)$, we define the degrees of freedom of the boosting fit in iteration m as

$$\mathrm{df}(m) = \mathrm{trace}(\mathcal{B}_m).$$

Even with $\nu = 1$, $\mathrm{df}(m)$ is very different from counting the number of variables which have been selected until iteration m.

Having some notion of degrees of freedom at hand, we can estimate the error variance $\sigma_\varepsilon^2 = \mathbb{E}[\varepsilon_i^2]$ in the linear model (5.4) by

$$\hat{\sigma}_\varepsilon^2 = \frac{1}{n - \mathrm{df}(m_{\mathrm{stop}})} \sum_{i=1}^n \left(Y_i - \hat{f}^{[m_{\mathrm{stop}}]}(X_i) \right)^2.$$

We can represent

$$\mathcal{B}_m = \sum_{j=1}^p \mathcal{B}_m^{(j)}, \tag{5.7}$$

where $\mathcal{B}_m^{(j)}$ is the (approximate) hat matrix which yields the fitted values for the jth predictor, i.e., $\mathcal{B}_m^{(j)} \mathbf{Y} = \mathbf{X}^{(j)} \hat{\beta}_j^{[m]}$. Note that the $\mathcal{B}_m^{(j)}$'s can be easily computed in an iterative way by up-dating as follows:

$$\begin{aligned} &\mathcal{B}_m^{(\hat{\mathcal{S}}_m)} = \mathcal{B}_{m-1}^{(\hat{\mathcal{S}}_m)} + \nu \cdot \mathcal{H}^{(\hat{\mathcal{S}}_m)}(I - \mathcal{B}_{m-1}), \\ &\mathcal{B}_m^{(j)} = \mathcal{B}_{m-1}^{(j)} \text{ for all } j \neq \hat{\mathcal{S}}_m. \end{aligned}$$

Thus, we have a decomposition of the total degrees of freedom into p terms:

$$\mathrm{df}(m) = \sum_{j=1}^{p} \mathrm{df}^{(j)}(m),$$
$$\mathrm{df}^{(j)}(m) = \mathrm{trace}(\mathcal{B}_m^{(j)}).$$

The individual degrees of freedom $\mathrm{df}^{(j)}(m)$ are a useful measure to quantify the "complexity" of the coefficient estimate $\hat{\beta}_j^{[m]}$.

5.4. *Internal stopping criteria for L_2Boosting.* Having some degrees of freedom at hand, we can now use internal information criteria for estimating a good stopping iteration, without pursuing some sort of cross-validation.

We can use the corrected AIC [49]:

$$\mathrm{AIC}_c(m) = \log(\hat{\sigma}^2) + \frac{1 + \mathrm{df}(m)/n}{(1 - \mathrm{df}(m) + 2)/n},$$
$$\hat{\sigma}^2 = n^{-1} \sum_{i=1}^{n} (Y_i - (\mathcal{B}_m \mathbf{Y})_i)^2.$$

In **mboost**, the corrected AIC criterion can be computed via `AIC(x, method = "corrected")` (with `x` being an object returned by `glmboost` or `gamboost` called with `family = GaussReg()`). Alternatively, we may employ the gMDL criterion (Hansen and Yu [38]):

$$\mathrm{gMDL}(m) = \log(S) + \frac{\mathrm{df}(m)}{n} \log(F),$$
$$S = \frac{n\hat{\sigma}^2}{n - \mathrm{df}(m)}, \quad F = \frac{\sum_{i=1}^{n} Y_i^2 - n\hat{\sigma}^2}{\mathrm{df}(m) S}.$$

The gMDL criterion bridges the AIC and BIC in a data-driven way: it is an attempt to adaptively select the better among the two.

When using L_2Boosting for binary classification (see also the end of Section 3.2 and the illustration in Section 5.2), we prefer to work with the binomial log-likelihood in AIC,

$$\mathrm{AIC}(m) = -2 \sum_{i=1}^{n} Y_i \log\left((\mathcal{B}_m \mathbf{Y})_i\right) + (1 - Y_i) \log\left(1 - (\mathcal{B}_m \mathbf{Y})_i\right) + 2\mathrm{df}(m), \tag{5.8}$$

or for $\mathrm{BIC}(m)$ with the penalty term $\log(n)\mathrm{df}(m)$. (If $(\mathcal{B}_m \mathbf{Y})_i \notin [0, 1]$, we truncate by $\max(\min((\mathcal{B}_m \mathbf{Y})_i, 1-\delta), \delta)$ for some small $\delta > 0$. e.g., $\delta = 10^{-5}$).

6. Boosting for variable selection. We address here the question whether boosting is a good variable selection scheme. For problems with many predictor variables, boosting is computationally much more efficient than classical all subset selection schemes. The mathematical properties of boosting for variable selection are still open questions, e.g., whether it leads to a consistent model selection method.

6.1. *L_2Boosting.* When borrowing from the analogy of L_2Boosting with the Lasso (see Section 5.2.1), the following is relevant. Consider a linear model as in (5.4), allowing for $p \gg n$ but being sparse. Then, there is a necessary and almost sufficient condition such that for some suitable penalty parameter λ in (5.5), the Lasso finds the true underlying sub-model (the predictor variables with corresponding regression coefficients $\neq 0$) with probability tending quickly to 1 as $n \to \infty$ [65]. It is important to note the role of the sufficient and necessary condition of the Lasso for model selection: Zhao and Yu [93] call it the "irrepresentable condition" which has (mainly) implications on the "degree of collinearity" of the design (predictor variables), and they give examples where it holds and where it fails to be true. A further complication is the fact that when tuning the Lasso for prediction optimality, i.e., choosing the penalty parameter λ in (5.5) such that the mean squared error is minimal, the probability for estimating the true sub-model converges to a number which is less than one or even zero if the problem is high-dimensional [65]. In fact, the prediction optimal tuned Lasso selects asymptotically too large models.

The bias of the Lasso mainly causes the difficulties mentioned above. We often would like to construct estimators which are less biased. It is instructive to look at regression with orthonormal design, i.e., the model (5.4) with $\sum_{i=1}^n X_i^{(j)} X_i^{(k)} = \delta_{jk}$. Then, the Lasso and also L_2Boosting with componentwise linear least squares and using very small ν (in step 4 of L_2Boosting, see Section 3.3.1) yield the soft-threshold estimator [23, 28], see Figure 8. It exhibits the same amount of bias regardless by how much the observation (the variable z in Figure 8) exceeds the threshold. This is in contrast to the hard-threshold estimator and the adaptive Lasso in (6.1) which are much better in terms of bias.

Nevertheless, the (computationally efficient) Lasso seems to be a very useful method for variable filtering: for many cases, the prediction optimal tuned Lasso selects a sub-model which contains the true model with high probability. A nice proposal to correct Lasso's over-estimation behavior is the adaptive Lasso, given by Zou [95]. It is based on re-weighting the penalty

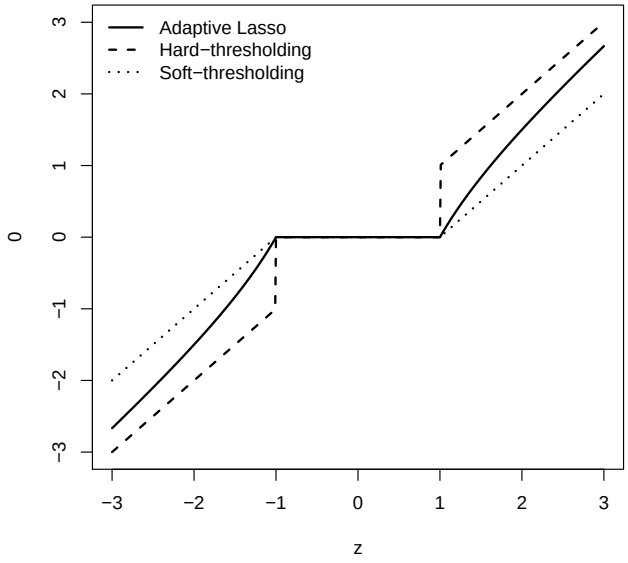

FIG 8. *Hard-threshold (dotted-dashed), soft-threshold (dotted) and adaptive Lasso (solid) estimator in a linear model with orthonormal design. For this design, the adaptive Lasso coincides with the non-negative garrote [13]. The value on the x-abscissa, denoted by z, is a single component of* $\mathbf{X}^\top \mathbf{Y}$.

function. Instead of (5.5), the adaptive Lasso estimator is

$$\hat{\beta}(\lambda) = \operatorname*{argmin}_{\beta} n^{-1} \sum_{i=1}^{n} \left(Y_i - \beta_0 - \sum_{j=1}^{p} \beta^{(j)} X_i^{(j)} \right)^2 + \lambda \sum_{j=1}^{p} \frac{|\beta^{(j)}|}{|\hat{\beta}_{\text{init}}^{(j)}|}, \tag{6.1}$$

where $\hat{\beta}_{\text{init}}$ is an initial estimator, e.g. the Lasso (from a first stage of Lasso estimation). Consistency of the adaptive Lasso for variable selection has been proved for the case with fixed predictor-dimension p [95] and also for the high-dimensional case with $p = p_n \gg n$ [48].

We do not expect that boosting is free from the difficulties which occur when using the Lasso for variable selection. The hope is though, that also boosting would produce an interesting set of sub-models when varying the number of iterations.

6.2. *Twin Boosting.* Twin Boosting [19] is the boosting analogue to the adaptive Lasso. It consists of two stages of boosting: the first stage is as usual, and the second stage is enforced to resemble the first boosting round. For example, if a variable has not been selected in the first round of boosting, it will not be selected in the second: this property also holds for the adaptive Lasso in (6.1), i.e. $\hat{\beta}^{(j)}_{\text{init}} = 0$ enforces $\hat{\beta}^{(j)} = 0$. Moreover, Twin Boosting with componentwise linear least squares is proved to be equivalent to the adaptive Lasso for the case of an orthonormal linear model and it is empirically shown, in general for various base procedures and models, that it has much better variable selection properties than the corresponding boosting algorithm [19]. In special settings, similar results can be obtained with Sparse Boosting [23]: however, Twin Boosting is much more generically applicable.

7. Boosting for exponential family models. For exponential family models with general loss functions, we can use the generic FGD algorithm as described in Section 2.1.

First, we address the issue about omitting a line search between steps 3 and 4 of the generic FGD algorithm. Consider the empirical risk at iteration m,

$$n^{-1}\sum_{i=1}^{n}\rho(Y_i, \hat{f}^{[m]}(X_i)) \approx n^{-1}\sum_{i=1}^{n}\rho(Y_i, \hat{f}^{[m-1]}(X_i)) - \nu n^{-1}\sum_{i=1}^{n} U_i \hat{g}^{[m]}(X_i), \tag{7.1}$$

using a first-order Taylor expansion and the definition of U_i. Consider the case with the componentwise linear least squares base procedure and without loss of generality with standardized predictor variables (i.e., $n^{-1}\sum_{i=1}^{n}\left(X_i^{(j)}\right)^2 = 1$ for all j). Then,

$$\hat{g}^{[m]}(x) = n^{-1}\sum_{i=1}^{n} U_i X_i^{(\hat{\mathcal{S}}_m)} x^{(\hat{\mathcal{S}}_m)},$$

and the expression in (7.1) becomes:

$$n^{-1}\sum_{i=1}^{n}\rho(Y_i, \hat{f}^{[m]}(X_i)) \approx n^{-1}\sum_{i=1}^{n}\rho(Y_i, \hat{f}^{[m-1]}(X_i)) - \nu(n^{-1}\sum_{i=1}^{n} U_i X_i^{(\hat{\mathcal{S}}_m)})^2. \tag{7.2}$$

In case of the squared error loss $\rho_{L_2}(y,f) = |y-f|^2/2$, we obtain the exact identity:

$$\begin{aligned} n^{-1}\sum_{i=1}^n \rho_{L_2}(Y_i, \hat{f}^{[m]}(X_i)) &= n^{-1}\sum_{i=1}^n \rho_{L_2}(Y_i, \hat{f}^{[m-1]}(X_i)) \\ &\quad -\nu(1-\nu/2)(n^{-1}\sum_{i=1}^n U_i X_i^{(\hat{\mathcal{S}}_m)})^2. \end{aligned}$$

Comparing this with Formula (7.2) we see that functional gradient descent with a general loss function and without additional line-search behaves very similar to L_2Boosting (since ν is small) with respect to optimizing the empirical risk; for L_2Boosting, the numerical convergence rate is $n^{-1}\sum_{i=1}^n \rho_{L_2}(Y_i, \hat{f}^{[m]}(X_i)) = O(m^{-1/6})$ $(m \to \infty)$ [81]. This completes our reasoning why the line-search in the general functional gradient descent algorithm can be omitted, of course at the price of doing more iterations but not necessarily more computing time (since the line-search is omitted in every iteration).

7.1. *BinomialBoosting.* For binary classification with $Y \in \{0,1\}$, BinomialBoosting uses the negative binomial log-likelihood from (3.1) as loss function. The algorithm is described in Section 3.3.2. Since the population minimizer is $f^*(x) = \log[p(x)/(1-p(x))]/2$, estimates from BinomialBoosting are on half of the logit-scale: the componentwise linear least squares base procedure yields a logistic linear model fit while using componentwise smoothing splines fits a logistic additive model. Many of the concepts and facts from Section 5 about L_2Boosting become useful heuristics for BinomialBoosting.

One principal difference is the derivation of the boosting hat matrix. Instead of (5.6), a linearization argument leads to the following recursion (assuming $\hat{f}^{[0]}(\cdot) \equiv 0$) for an approximate hat matrix $\mathcal{B}_m$:

$$\begin{aligned} &\mathcal{B}_1 = \nu 4 W^{[0]} \mathcal{H}^{(\hat{\mathcal{S}}_1)}, \\ &\mathcal{B}_m = \mathcal{B}_{m-1} + 4\nu W^{[m-1]} \mathcal{H}^{(\hat{\mathcal{S}}_m)}(I - \mathcal{B}_{m-1}) \quad (m \geq 2), \\ &W^{[m]} = \operatorname{diag}(\hat{p}^{[m]}(X_i)(1-\hat{p}^{[m]}(X_i);\ 1 \leq i \leq n). \end{aligned} \tag{7.3}$$

A derivation is given in Appendix A. Degrees of freedom are then defined as in Section 5.3,

$$\mathrm{df}(m) = \operatorname{trace}(\mathcal{B}_m),$$

and they can be used for information criteria, e.g.

$$\text{AIC}(m) = -2\sum_{i=1}^{n}[Y_i \log(\hat{p}^{[m]}(X_i)) + (1 - Y_i)\log(1 - \hat{p}^{[m]}(X_i))] + 2\text{df}(m),$$

or for BIC(m) with the penalty term $\log(n)\text{df}(m)$. In **mboost**, this AIC criterion can be computed via `AIC(x, method = "classical")` (with `x` being an object returned by `glmboost` or `gamboost` called with `family = Binomial()`).

Illustration: Wisconsin prognostic breast cancer. Prediction models for recurrence events in breast cancer patients based on covariates which have been computed from a digitized image of a fine needle aspirate of breast tissue (those measurements describe characteristics of the cell nuclei present in the image) have been studied by Street et al. [80] [the data is part of the UCI repository 11].

We first analyze this data as a binary prediction problem (recurrence vs. non-recurrence) and later in Section 8 by means of survival models. We are faced with many covariates ($p = 32$) for a limited number of observations without missing values ($n = 194$), and variable selection is an issue. We can choose a classical logistic regression model via AIC in a stepwise algorithm as follows

```
R> cc <- complete.cases(wpbc)
R> wpbc2 <- wpbc[cc, colnames(wpbc) != "time"]
R> wpbc_step <- step(glm(status ~ ., data = wpbc2,
         family = binomial()), trace = 0)
```

The final model consists of 16 parameters with

```
R> logLik(wpbc_step)
```

```
'log Lik.' -80.13 (df=16)
```

```
R> AIC(wpbc_step)
```

```
[1] 192.26
```

and we want to compare this model to a logistic regression model fitted via gradient boosting. We simply select the `Binomial` family (with default offset of $1/2\log(\hat{p}/(1-\hat{p}))$, where $\hat{p}$ is the empirical proportion of recurrences) and we initially use $m_{\text{stop}} = 500$ boosting iterations

```
R> ctrl <- boost_control(mstop = 500, center = TRUE)
R> wpbc_glm <- glmboost(status ~ ., data = wpbc2,
         family = Binomial(), control = ctrl)
```

The classical AIC criterion (-2 log-likelihood + 2 df) suggests to stop after

```
R> aic <- AIC(wpbc_glm, "classical")
R> aic
```

```
[1] 199.54
Optimal number of boosting iterations: 465
Degrees of freedom (for mstop = 465): 9.147
```

boosting iterations. We now restrict the number of boosting iterations to $m_{\text{stop}} = 465$ and then obtain the estimated coefficients via

```
R> wpbc_glm <- wpbc_glm[mstop(aic)]
R> coef(wpbc_glm)[abs(coef(wpbc_glm)) > 0]
```

```
      (Intercept)       mean_radius      mean_texture
      -1.2511e-01       -5.8453e-03       -2.4505e-02
  mean_smoothness     mean_symmetry   mean_fractaldim
       2.8513e+00       -3.9307e+00       -2.8253e+01
       SE_texture      SE_perimeter    SE_compactness
      -8.7553e-02        5.4917e-02        1.1463e+01
     SE_concavity  SE_concavepoints       SE_symmetry
      -6.9238e+00       -2.0454e+01        5.2125e+00
    SE_fractaldim      worst_radius   worst_perimeter
       5.2187e+00        1.3468e-02        1.2108e-03
       worst_area  worst_smoothness worst_compactness
       1.8646e-04        9.9560e+00       -1.9469e-01
            tsize            pnodes
       4.1561e-02        2.4445e-02
```

(because of using the offset-value $\hat{f}^{[0]}$, we have to add the value $\hat{f}^{[0]}$ to the reported intercept estimate above for the logistic regression model).

A generalized additive model adds more flexibility to the regression function but is still interpretable. We fit a logistic additive model to the `wpbc` data as follows:

```
R> wpbc_gam <- gamboost(status ~ ., data = wpbc2,
         family = Binomial())
R> mopt <- mstop(aic <- AIC(wpbc_gam, "classical"))
R> aic
```

```
[1] 196.33
Optimal number of boosting iterations: 84
Degrees of freedom (for mstop = 84): 13.754
```

This model selected 16 out of 32 covariates. The partial contributions of the four most important variables are depicted in Figure 9 indicating a remarkable degree of non-linearity.

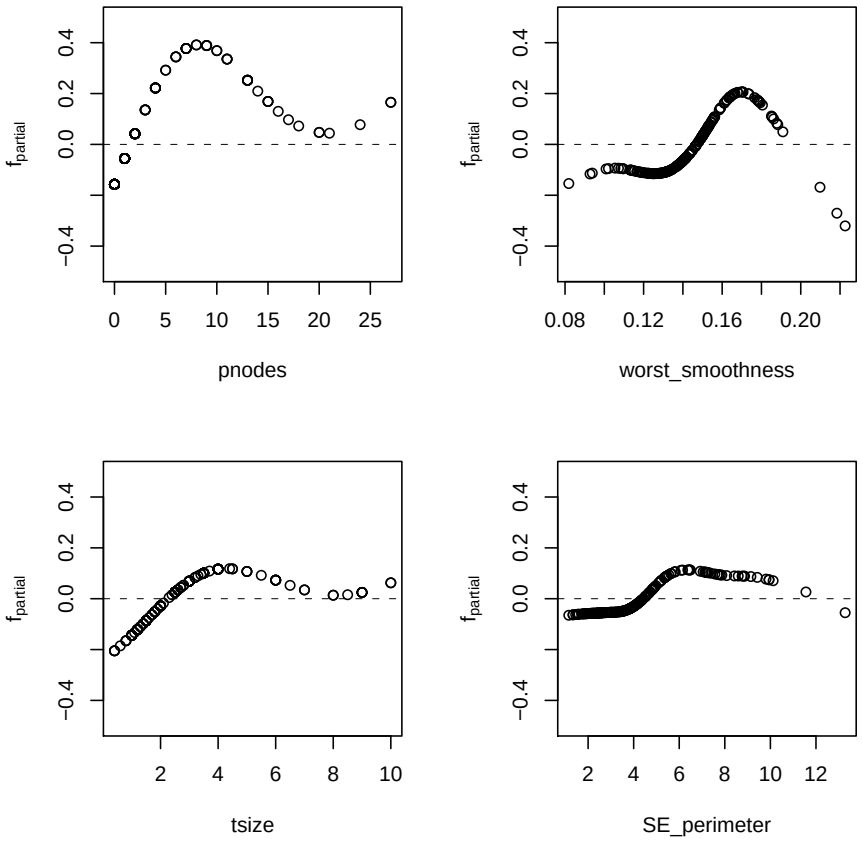

FIG 9. *wpbc data: Partial contributions of four selected covariates in an additive logistic model (without centering of estimated functions to mean zero).*

7.2. *PoissonBoosting.* For count data with $Y \in \{0, 1, 2, \ldots\}$, we can use Poisson regression: we assume that $Y|X = x$ has a Poisson$(\lambda(x))$ distribution and the goal is to estimate the function $f(x) = \log(\lambda(x))$. The negative log-likelihood yields then the loss function

$$\rho(y, f) = -yf + \exp(f), \quad f = \log(\lambda),$$

which can be used in the functional gradient descent algorithm in Section 2.1, and it is implemented in **mboost** as `Poisson()` family.

Similarly to (7.3), the approximate boosting hat matrix is computed by

the following recursion

$$\begin{aligned}
&\mathcal{B}_1 = \nu W^{[0]} \mathcal{H}^{(\hat{\mathcal{S}}_1)}, \\
&\mathcal{B}_m = \mathcal{B}_{m-1} + \nu W^{[m-1]} \mathcal{H}^{(\hat{\mathcal{S}}_m)}(I - \mathcal{B}_{m-1}) \quad (m \geq 2), \\
&W^{[m]} = \operatorname{diag}(\hat{\lambda}^{[m]}(X_i);\ 1 \leq i \leq n).
\end{aligned} \tag{7.4}$$

7.3. *Initialization of boosting.* We have briefly described in Sections 2.1 and 4.1 the issue of choosing an initial value $\hat{f}^{[0]}(\cdot)$ for boosting. This can be quite important for applications where we would like to estimate some parts of a model in an unpenalized (non-regularized) fashion and others being subject to regularization.

For example, we may think of a parametric form of $\hat{f}^{[0]}(\cdot)$, estimated by maximum likelihood, and deviations from the parametric model would be built in by pursuing boosting iterations (with a nonparametric weak learner). A concrete example would be: $\hat{f}^{[0]}(\cdot)$ is the maximum likelihood estimate in a generalized linear model and boosting would be done with componentwise smoothing splines to model additive deviations from a generalized linear model. A related strategy has been used in [4] for modeling multivariate volatility in financial time series.

Another example would be a linear model $\mathbf{Y} = \mathbf{X}\beta + \varepsilon$ as in (5.4) where some of the predictor variables, say the first q predictor variables $X^{(1)}, \ldots, X^{(q)}$, enter the estimated linear model in an unpenalized way. We propose to do ordinary least squares regression on $X^{(1)}, \ldots, X^{(q)}$: consider the projection P_q onto the linear span of $X^{(1)}, \ldots, X^{(q)}$ and use L_2Boosting with componentwise linear least squares on the new response $(I - P_q)\mathbf{Y}$ and the new $p - q$-dimensional predictor $(I - P_q)\mathbf{X}$. The final model estimate is then $\sum_{j=1}^{q} \hat{\beta}_{\mathrm{OLS},j} x^{(j)} + \sum_{j=q+1}^{p} \hat{\beta}^{[m_{\mathrm{stop}}]} \tilde{x}^{(j)}$, where the latter part is from L_2Boosting and $\tilde{x}^{(j)}$ is the residual when linearly regressing $x^{(j)}$ to $x^{(1)}, \ldots, x^{(q)}$. A special case which is used in most applications is with $q = 1$ and $X^{(1)} \equiv 1$ encoding for an intercept. Then, $(I - P_1)Y = Y - \overline{Y}$ and $(I - P_1)X^{(j)} = X^{(j)} - n^{-1} \sum_{i=1}^{n} X_i^{(j)}$. This is exactly the proposal at the end of Section 4.1. For generalized linear models, analogous concepts can be used.

8. Survival analysis. The negative gradient of Cox' partial likelihood can be used to fit proportional hazards models to censored response variables with boosting algorithms [71]. Of course, all types of base learners can be utilized; for example, componentwise linear least squares fits a Cox model with a linear predictor.

Alternatively, we can use the weighted least squares framework with weights arising from inverse probability censoring. We sketch this approach in the sequel, details are given in [45]. We assume complete data of the following form: survival times $T_i \in \mathbb{R}^+$ (some of them right-censored) and predictors $X_i \in \mathbb{R}^p,\ i = 1, \ldots, n$. We transform the survival times to the log-scale, but this step is not crucial for what follows: $Y_i = \log(T_i)$. What we observe is

$$O_i = (\tilde{Y}_i, X_i, \Delta_i),\ \tilde{Y}_i = \log(\tilde{T}_i),\ \tilde{T}_i = \min(T_i, C_i),$$

where $\Delta_i = I(T_i \leq C_i)$ is a censoring indicator and C_i the censoring time. Here, we make a restrictive assumption that C_i is conditionally independent of T_i given X_i (and we assume independence among different indices i): this implies that the coarsening at random assumption holds [88].

We consider the squared error loss for the complete data, $\rho(y, f) = |y - f|^2$ (without the irrelevant factor 1/2). For the observed data, the following weighted version turns out to be useful:

$$\rho_{\text{obs}}(o, f) = (\tilde{y} - f)^2 \Delta \frac{1}{G(\tilde{t}|x)},$$
$$G(c|x) = \mathbb{P}[C > c|X = x].$$

Thus, the observed data loss function is weighted by the inverse probability for censoring $\Delta G(\tilde{t}|x)^{-1}$ (the weights are inverse probabilities of censoring; IPC). Under the coarsening at random assumption, it then holds that

$$\mathbb{E}_{Y,X}[(Y - f(X))^2] = \mathbb{E}_O[\rho_{\text{obs}}(O, f(X))],$$

see van der Laan and Robins [88].

The strategy is then to estimate $G(\cdot|x)$, e.g., by the Kaplan-Meier estimator, and do weighted L_2Boosting using the weighted squared error loss:

$$\sum_{i=1}^{n} \Delta_i \frac{1}{\hat{G}(\tilde{T}_i|X_i)} (\tilde{Y}_i - f(X_i))^2,$$

where the weights are of the form $\Delta_i \hat{G}(\tilde{T}_i|X_i)^{-1}$ (the specification of the estimator $\hat{G}(t|x)$ may play a substantial role in the whole procedure). As demonstrated in the previous sections, we can use various base procedures as long as they allow for weighted least squares fitting. Furthermore, the concepts of degrees of freedom and information criteria are analogous to Sections 5.3 and 5.4. Details are given in [45].

Illustration: Wisconsin prognostic breast cancer (cont.). Instead of the binary response variable describing the recurrence status, we make use of the additionally available time information for modeling the time to recurrence, i.e., all observations with non-recurrence are censored. First, we calculate IPC weights

```
R> zensored <- wpbc$status == "R"
R> iw <- IPCweights(Surv(wpbc$time, zensored))
R> wpbc3 <- wpbc[, names(wpbc) != "status"]
```

and fit a weighted linear model by boosting with componentwise linear weighted least squares as base procedure:

```
R> ctrl <- boost_control(mstop = 500, center = TRUE)
R> wpbc_surv <- glmboost(log(time) ~ ., data = wpbc3,
         control = ctrl, weights = iw)
R> mstop(aic <- AIC(wpbc_surv))
```

```
[1] 122
```

```
R> wpbc_surv <- wpbc_surv[mstop(aic)]
```

The following variables have been selected for fitting

```
R> names(coef(wpbc_surv)[abs(coef(wpbc_surv)) > 0])
```

```
 [1] "mean_radius"          "mean_texture"
 [3] "mean_perimeter"       "mean_smoothness"
 [5] "mean_symmetry"        "SE_texture"
 [7] "SE_smoothness"        "SE_concavepoints"
 [9] "SE_symmetry"          "worst_concavepoints"
```

and the fitted values are depicted in Figure 10, showing a reasonable model fit.

Alternatively, a Cox model with linear predictor can be fitted using L_2Boosting by implementing the negative gradient of the partial likelihood (see [71]) via

```
R> ctrl <- boost_control(center = TRUE)
R> glmboost(Surv(wpbc$time, wpbc$status == "N") ~
         ., data = wpbc, family = CoxPH(), control = ctrl)
```

For more examples, such as fitting an additive Cox model using **mboost**, see [44].

9. Other works. We briefly summarize here some other works which have not been mentioned in the earlier sections. A very different exposition than ours is the overview of boosting by Meir and Rätsch [66].

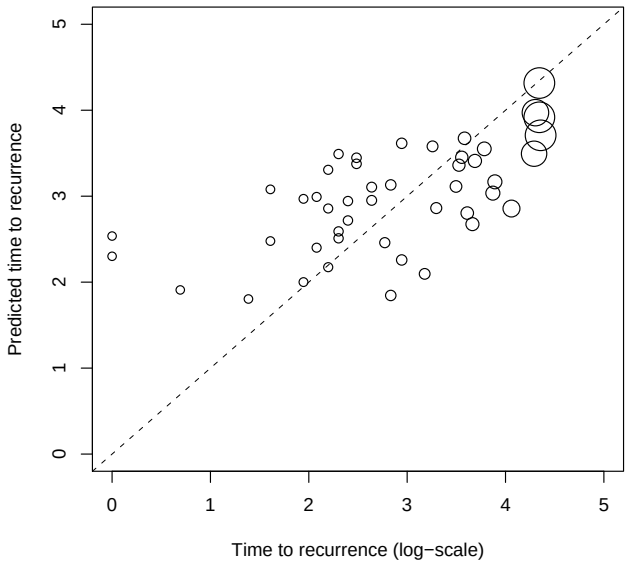

FIG 10. *wpbc data: Fitted values of an IPC-weighted linear model, taking both time to recurrence and censoring information into account. The radius of the circles is proportional to the IPC weight of the corresponding observation, censored observations with IPC weight zero are not plotted.*

9.1. *Methodology and applications .* Boosting methodology has been used for various other statistical models than what we have discussed in the previous sections. Models for multivariate responses are studied in [20, 59]; some multi-class boosting methods are discussed in [33, 94]. Other works deal with boosting approaches for generalized linear and nonparametric models [55, 85, 86], for flexible semiparametric mixed models [87] or for nonparametric models with quality constraints [54, 56]. Boosting methods for estimating propensity scores, a special weighting scheme for modeling observational data, are proposed by [63].

There are numerous applications of boosting methods to real data problems. We mention here classification of tumor types from gene expressions [25, 26], multivariate financial time series [2–4], text classification [78], document routing [50] or survival analysis [8] (different from the approach in Section 8).

9.2. *Asymptotic theory.* The asymptotic analysis of boosting algorithms include consistency and minimax rate results. The first consistency result for AdaBoost has been given by Jiang [51], refinements are given in [7]. Later, Zhang and Yu [91] generalized the results for a functional gradient descent with an additional relaxation scheme, and their theory covers also more general loss functions than the exponential loss in AdaBoost. For L_2Boosting, the first minimax rate result has been established by Bühlmann and Yu [22]. This has been extended to much more general settings by Yao et al. [90] and Bissantz et al. [10].

In the machine learning community, there has been a substantial focus on estimation in the convex hull of function classes (cf. [5, 6, 58]). For example, one may want to estimate a regression or probability function by using

$$\sum_{k=1}^{\infty} \hat{w}_k \hat{g}^{[k]}(\cdot), \ \hat{w}_k \geq 0, \ \sum_{k=1}^{\infty} \hat{w}_k = 1,$$

where the $\hat{g}^{[k]}(\cdot)$'s belong to a function class such as stumps or trees with a fixed number of terminal nodes. The estimator above is a convex combination of individual functions, in contrast to boosting which pursues a linear combination. By scaling, which is necessary in practice and theory (cf. [58]), one can actually look at this as a linear combination of functions whose coefficients satisfy $\sum_k \hat{w}_k = \lambda$. This then represents an ℓ^1-constraint as in Lasso, a relation which we have already seen from another perspective in Section 5.2.1. Consistency of such convex combination or ℓ^1-regularized "boosting" methods have been given by Lugosi and Vayatis [58]. Mannor et al. [61] and Blanchard et al. [12] derived results for rates of convergence of (versions of) convex combination schemes.

APPENDIX A: SOFTWARE

The data analyses presented in this paper are performed using the **mboost** package. The theoretical ingredients of boosting algorithms, such as loss functions and its negative gradients, base learners and internal stopping criteria, find their computational counterparts in the **mboost** package. Its implementation and user-interface reflect our statistical perspective of boosting as a tool for estimation in structured models. For example, and extending the reference implementation of tree-based gradient boosting from the **gbm** package [74], **mboost** allows to fit potentially high-dimensional linear or smooth additive models, and it has methods to compute degrees of freedom which in turn allow for the use of information criteria such as AIC or BIC or for estimation of variance. Moreover, for high-dimensional (generalized)

linear models, our implementation is very fast to fit models even when the dimension of the predictor space is in the ten-thousands.

The `Family` function in **mboost** can be used to create an object of class *boost_family* implementing the negative gradient for general loss functions. Such an object can later be fed into the fitting procedure of a linear or additive model which optimizes the corresponding empirical risk (an example is given in Section 5.2). Therefore, we aren't limited to already implemented boosting algorithms but can easily set up our own boosting procedure by implementing the negative gradient of the loss function of interest.

Both the source version as well as binaries for several operating systems of the **mboost** [43] package are freely available from the Comprehensive R Archive Network (`http://CRAN.R-project.org`). The reader can install our package directly from the R prompt via

```
R> install.packages("mboost", dependencies = TRUE)
R> library("mboost")
```

All analyses presented in this paper are contained in a package vignette. The rendered output of the analyses is available by the R-command

```
R> vignette("mboost_illustrations", package = "mboost")
```

whereas the R code for reproducibility of our analyses can be assessed by

```
R> edit(vignette("mboost_illustrations", package = "mboost"))
```

There are several alternative implementations of boosting techniques available as R add-on packages. The reference implementation for tree-based gradient boosting is **gbm** [74]. Boosting for additive models based on penalized B-splines is implemented in **GAMBoost** [9, 84].

APPENDIX B: DERIVATION OF BOOSTING HAT MATRICES

Derivation of formula (7.3). The negative gradient is

$$-\frac{\partial}{\partial f}\rho(y,f) = 2(y-p), \quad p = \frac{\exp(f)}{\exp(f)+\exp(-f)}.$$

Next, we linearize $\hat{p}^{[m]}$: we denote by $\hat{p}^{[m]} = (\hat{p}^{[m]}(X_1), \ldots, \hat{p}^{[m]}(X_n))^\top$ and analogously for $\hat{f}^{[m]}$. Then,

$$\begin{aligned} \hat{p}^{[m]} &\approx \hat{p}^{[m-1]} + \frac{\partial p}{\partial f}\Big|_{f=\hat{f}^{m-1}} \left(\hat{f}^{[m]} - \hat{f}^{[m-1]}\right) \\ &= \hat{p}^{[m-1]} + 2W^{[m-1]}\nu\mathcal{H}^{(\hat{\mathcal{S}}_m)}2\left(\mathbf{Y} - \hat{p}^{[m-1]}\right), \end{aligned} \tag{B.1}$$

where $W^{[m]} = \mathrm{diag}(\hat{p}(X_i)(1-\hat{p}(X_i)); 1 \le i \le n)$. Since for the hat matrix, $\mathcal{B}_m \mathbf{Y} = \hat{p}^{[m]}$, we obtain from (B.1)

$$\begin{aligned}
\mathcal{B}_1 &\approx \nu 4 W^{[0]} \mathcal{H}^{\hat{\mathcal{S}}_1}, \\
\mathcal{B}_m &\approx \mathcal{B}_{m-1} + \nu 4 W^{[m-1]} \mathcal{H}^{\hat{\mathcal{S}}_m} (I - \mathcal{B}_{m-1}) \quad (m \ge 2),
\end{aligned}$$

which shows that (7.3) is approximately true. □

Derivation of formula (7.4). The arguments are analogous as for the binomial case above. Here, the negative gradient is

$$-\frac{\partial}{\partial f}\rho(y,f) = y - \lambda, \quad \lambda = \exp(f).$$

When linearizing $\hat{\lambda}^{[m]} = (\hat{\lambda}^{[m]}(X_1), \ldots, \hat{\lambda}^{[m]}(X_n))^\top$ we get, analogously to (B.1),

$$\begin{aligned}
\hat{\lambda}^{[m]} &\approx \hat{\lambda}^{[m-1]} + \frac{\partial \lambda}{\partial f}|_{f=\hat{f}^{m-1}} \left(\hat{f}^{[m]} - \hat{f}^{[m-1]}\right) \\
&= \hat{\lambda}^{[m-1]} + W^{[m-1]} \nu \mathcal{H}^{(\hat{\mathcal{S}}_m)} \left(\mathbf{Y} - \hat{\lambda}^{[m-1]}\right),
\end{aligned}$$

where $W^{[m]} = \mathrm{diag}(\hat{\lambda}(X_i)); 1 \le i \le n)$. We then complete the derivation of (7.4) as in the binomial case above. □

ACKNOWLEDGEMENTS

We would like to thank Axel Benner, Florian Leitenstorfer, Roman Lutz and Lukas Meier for discussions and detailed remarks. Moreover, we thank four referees, the editor and the executive editor Ed George for constructive comments. The work of T. Hothorn was supported by Deutsche Forschungsgemeinschaft (DFG) under grant HO 3242/1-3.

REFERENCES

[1] Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation* **9** 1545–1588.

[2] Audrino, F. and Barone-Adesi, G. (2005). Functional gradient descent for financial time series with an application to the measurement of market risk. *Journal of Banking and Finance* **29** 959–977.

[3] Audrino, F. and Barone-Adesi, G. (2005). A multivariate FGD technique to improve VaR computation in equity markets. *Computational Management Science* **2** 87–106.

[4] Audrino, F. and Bühlmann, P. (2003). Volatility estimation with functional gradient descent for very high-dimensional financial time series. *Journal of Computational Finance* **6** 65–89.

[5] Bartlett, P. (2003). Prediction algorithms: complexity, concentration and convexity. In *Proceedings of the 13th IFAC Symposium on System Identification*.
[6] Bartlett, P., Jordan, M. and McAuliffe, J. (2006). Convexity, classification, and risk bounds. *Journal of the American Statistical Association* **101** 138–156.
[7] Bartlett, P. and Traskin, M. (2006). Adaboost is consistent. Tech. rep., University of California.
URL http://www.stat.berkeley.edu/tech-reports/722.pdf
[8] Benner, A. (2002). Application of "aggregated classifiers" in survival time studies. In *Proceedings in Computational Statistics (COMPSTAT)* (W. H. W. and B. Rönz, eds.). Physica-Verlag, Heidelberg.
[9] Binder, H. (2006). *Generalized additive models by likelihood based boosting*. R package version 0.9-3.
URL http://CRAN.R-project.org
[10] Bissantz, N., Hohage, T., Munk, A. and Ruymgaart, F. (2005). Convergence rates of general regularization methods for statistical inverse problems and applications. Tech. rep., Universität Göttingen.
URL http://www.stochastik.math.uni-goettingen.de/preprints/bissantz_hohage_munk_ruymgaart.pdf
[11] Blake, C. L. and Merz, C. J. (1998). UCI repository of machine learning databases.
URL http://www.ics.uci.edu/~mlearn/MLRepository.html
[12] Blanchard, G., Lugosi, G. and Vayatis, N. (2003). On the rate of convergence of regularized boosting classifiers. *Journal of Machine Learning Research* **4** 861–894.
[13] Breiman, L. (1995). Better subset regression using the nonnegative garrote. *Technometrics* **37** 373–384.
[14] Breiman, L. (1996). Bagging predictors. *Machine Learning* **24** 123–140.
[15] Breiman, L. (1998). Arcing classifiers (with discussion). *The Annals of Statistics* **26** 801–849.
[16] Breiman, L. (1999). Prediction games & arcing algorithms. *Neural Computation* **11** 1493–1517.
[17] Breiman, L. (2001). Random forests. *Machine Learning* **45** 5–32.
[18] Bühlmann, P. (2006). Boosting for high-dimensional linear models. *The Annals of Statistics* **34** 559–583.
[19] Bühlmann, P. (2006). Twin boosting: improved feature selection and prediction. Tech. rep., ETH Zürich.
URL ftp://ftp.stat.math.ethz.ch/Research-Reports/Other-Manuscripts/buhlmann/TwinBoosting.pdf
[20] Bühlmann, P. and Lutz, R. (2006). Boosting algorithms: with an application to bootstrapping multivariate time series. In *The Frontiers in Statistics* (J. Fan and H. Koul, eds.). Imperial College Press.
[21] Bühlmann, P. and Yu, B. (2000). Discussion of additive logistic regression: a statistical view (j. friedman, t. hastie and r. tibshirani, auths.). *The Annals of Statistics* **28** 377–386.
[22] Bühlmann, P. and Yu, B. (2003). Boosting with the L_2 loss: Regression and classification. *Journal of the American Statistical Association* **98** 324–339.
[23] Bühlmann, P. and Yu, B. (2006). Sparse boosting. *Journal of Machine Learning Research* **7** 1001–1024.
[24] Buja, A., Stuetzle, W. and Shen, Y. (2005). Loss functions for binary class probability estimation: structure and applications. Tech. rep., University of Washington.
URL http://www.stat.washington.edu/wxs/Learning-papers/paper-proper-scoring.pdf

[25] Dettling, M. (2004). Bagboosting for tumor classification with gene expression data. *Bioinformatics* **20** 3583–3593.
[26] Dettling, M. and Bühlmann, P. (2003). Boosting for tumor classification with gene expression data. *Bioinformatics* **19** 1061–1069.
[27] DiMarzio, M. and Taylor, C. (2005). Multistep kernel regression smoothing by boosting. Tech. rep., University of Leeds.
[28] Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004). Least angle regression (with discussion). *The Annals of Statistics* **32** 407–451.
[29] Freund, Y. and Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*. Lecture Notes in Computer Science, Springer.
[30] Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA.
[31] Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55** 119–139.
[32] Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics* **29** 1189–1232.
[33] Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *The Annals of Statistics* **28** 337–407.
[34] Garcia, A. L., Wagner, K., Hothorn, T., Koebnick, C., Zunft, H. J. and Trippo, U. (2005). Improved prediction of body fat by measuring skinfold thickness, circumferences, and bone breadths. *Obesity Research* **13** 626–634.
[35] Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, M., Iacus, S., Irizarry, R., Leisch, F., Li, C., Mächler, M., Rossini, A. J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J. Y. and Zhang, J. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology* **5** R80.
[36] Green, P. and Silverman, B. (1994). *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*. Chapman & Hall, New York.
[37] Greenshtein, E. and Ritov, Y. (2004). Persistence in high-dimensional predictor selection and the virtue of over-parametrization. *Bernoulli* **10** 971–988.
[38] Hansen, M. and Yu, B. (2001). Model selection and minimum description length principle. *Journal of the American Statistical Association* **96** 746–774.
[39] Hastie, T. and Efron, B. (2004). *lars: Least Angle Regression, Lasso and Forward Stagewise*. R package version 0.9-5.
URL http://CRAN.R-project.org
[40] Hastie, T. and Tibshirani, R. (1986). Generalized additive models (with discussion). *Statistical Science* **1** 297–318.
[41] Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models*. Chapman & Hall, London.
[42] Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning; Data Mining,Inference and Prediction*. Springer, New York.
[43] Hothorn, T. and Bühlmann, P. (2006). *mboost: Model-Based Boosting*. R package version 0.5-0.
URL http://CRAN.R-project.org/
[44] Hothorn, T. and Bühlmann, P. (2006). Model-based boosting in high dimensions.

Bioinformatics **22** 2828–2829.

[45] HOTHORN, T., BÜHLMANN, P., DUDOIT, S., MOLINARO, A. and VAN DER LAAN, M. (2006). Survival ensembles. *Biostatistics* **7** 355–373.

[46] HOTHORN, T., HORNIK, K. and ZEILEIS, A. (2006). *party: A Laboratory for Recursive Part(y)itioning*. R package version 0.9-8, http://CRAN.R-project.org/.

[47] HOTHORN, T., HORNIK, K. and ZEILEIS, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics* **15** 651–674.

[48] HUANG, J., MA, S. and ZHANG, C.-H. (2006). Adaptive Lasso for sparse high-dimensional regression. Tech. rep., University of Iowa.

[49] HURVICH, C., SIMONOFF, J. and TSAI, C.-L. (1998). Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion. *Journal of the Royal Statististical Society, Series B* **60** 271–293.

[50] IYER, R., LEWIS, D., SCHAPIRE, R., SINGER, Y. and SINGHAL, A. (2000). Boosting for document routing. In *Proceedings of CIKM-00, 9th ACM Int. Conf. on Information and Knowledge Management* (A. Agah, J. Callan and E. Rundensteiner, eds.). ACM Press.

[51] JIANG, W. (2004). Process consistency for AdaBoost (with discussion). *The Annals of Statistics* **32** 13–29 (disc. pp. 85–134).

[52] KEARNS, M. and VALIANT, L. (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery* **41** 67–95.

[53] KOLTCHINSKII, V. and PANCHENKO, D. (2002). Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics* **30** 1–50.

[54] LEITENSTORFER, F. and TUTZ, G. (2006). Smoothing with curvature constraints based on boosting techniques. In *Proceedings in Computational Statistics (COMPSTAT)* (A. Rizzi and M. Vichi, eds.). Physica-Verlag, Heidelberg.

[55] LEITENSTORFER, F. and TUTZ, G. (2007). Generalized monotonic regression based on B-splines with an application to air pollution data. *Biostatistics* In press.

[56] LEITENSTORFER, F. and TUTZ, G. (2007). Generalized smooth monotonic regression in additive modelling. *Journal of Computational and Graphical Statistics* In press.

[57] LOZANO, A., KULKARNI, S. and SCHAPIRE, R. (2006). Convergence and consistency of regularized boosting algorithms with stationary β-mixing observations. In *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf and J. Platt, eds.), vol. 18. MIT Press.

[58] LUGOSI, G. and VAYATIS, N. (2004). On the Bayes-risk consistency of regularized boosting methods (with discussion). *The Annals of Statistics* **32** 30–55 (disc. pp. 85–134).

[59] LUTZ, R. and BÜHLMANN, P. (2006). Boosting for high-multivariate responses in high-dimensional linear regression. *Statistica Sinica* **16** 471–494.

[60] MALLAT, S. and ZHANG, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing* **41** 3397–3415.

[61] MANNOR, S., MEIR, R. and ZHANG, T. (2003). Greedy algorithms for classification–consistency, convergence rates, and adaptivity. *Journal of Machine Learning Research* **4** 713–741.

[62] MASON, L., BAXTER, J., BARTLETT, P. and FREAN, M. (2000). Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers* (A. Smola, P. Bartlett, B. B. Schölkopf and D. Schuurmans, eds.). MIT Press, Cambridge.

[63] McCaffrey, D. F., Ridgeway, G. and Morral, A. R. G. (2004). Propensity score estimation with boosted regression for evaluating causal effects in observational studies. *Psychological Methods* **9** 403–425.

[64] Mease, D., Wyner, A. and Buja, A. (2006). Cost-weighted boosting with jittering and over/under-sampling: Jous-boost. Tech. rep., University of Pennsylvania.
URL http://www-stat.wharton.upenn.edu/~buja/PAPERS/paper-jous-boost.pdf

[65] Meinshausen, N. and Bühlmann, P. (2006). High-dimensional graphs and variable selection with the Lasso. *The Annals of Statistics* **34** 1436–1462.

[66] Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning* (S. Mendelson and A. Smola, eds.). Lecture Notes in Computer Science, Springer.

[67] Osborne, M., Presnell, B. and Turlach, B. (2000). A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis* **20** 389–403.

[68] Park, M.-Y. and Hastie, T. (2006). An l1 regularization-path algorithm for generalized linear models. Tech. rep., Stanford University.
URL http://www-stat.stanford.edu/~hastie/Papers/glmpath.jrssb.pdf

[69] R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
URL http://www.R-project.org

[70] Rätsch, G., Onoda, T. and Müller, K. (2001). Soft margins for AdaBoost. *Machine Learning* **42** 287–320.

[71] Ridgeway, G. (1999). The state of boosting. *Computing Science and Statistics* **31** 172–181.

[72] Ridgeway, G. (2000). Discussion of Additive logistic regression: a statistical view of boosting (J. Friedman, T. Hastie, R. Tibshirani, auths.). *The Annals of Statistics* **28** 393–400.

[73] Ridgeway, G. (2002). Looking for lumps: Boosting and bagging for density estimation. *Computational Statistics & Data Analysis* **38** 379–392.

[74] Ridgeway, G. (2006). *gbm: Generalized Boosted Regression Models*. R package version 1.5-7.
URL http://www.i-pensieri.com/gregr/gbm.shtml

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

[76] Schapire, R. (2002). The boosting approach to machine learning: an overview. In *MSRI Workshop on Nonlinear Estimation and Classification* (D. Denison, M. Hansen, C. Holmes, B. Mallick and B. Yu, eds.). Springer.

[77] Schapire, R., Freund, Y., Bartlett, P. and Lee, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics* **26** 1651–1686.

[78] Schapire, R. and Singer, Y. (2000). Boostexter: a boosting-based system for text categorization. *Machine Learning* **39** 135–168.

[79] Southwell, R. (1946). *Relaxation Methods in Theoretical Physics*. Oxford Univeristy Press.

[80] Street, W. N., Mangasarian, O. L., and Wolberg, W. H. (1995). An inductive learning approach to prognostic prediction. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

[81] Temlyakov, V. (2000). Weak greedy algorithms. *Advances in Computational Mathematics* **12** 213–227.

[82] Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B* **58** 267–288.
[83] Tukey, J. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.
[84] Tutz, G. and Binder, H. (2006). Generalized additive modelling with implicit variable selection by likelihood based boosting. *Biometrics* **62** 961–971.
[85] Tutz, G. and Binder, H. (2007). Boosting Ridge regression. *Computational Statistics & Data Analysis* In press.
[86] Tutz, G. and Hechenbichler, K. (2005). Aggregating classifiers with ordinal response structure. *Journal Statistical Computation and Simulation* **75** 391–408.
[87] Tutz, G. and Reithinger, F. (2007). Flexible semiparametric mixed models. *Statistics in Medicine* In press.
[88] van der Laan, M. and Robins, J. (2003). *Unified Methods for Censored Longitudinal Data and Causality*. Springer.
[89] West, M., Blanchette, C., Dressman, H., Huang, E., Ishida, S., Spang, R., Zuzan, H., Olson, J., Marks, J. and Nevins, J. (2001). Predicting the clinical status of human breast cancer by using gene expression profiles. *Proceedings of the National Academy of Sciences (USA)* **98** 11462–11467.
[90] Yao, Y., Rosasco, L. and Caponneto, A. (2005). On early stopping in gradient descent learning. Tech. rep., University of California, Berkeley.
URL http://math.berkeley.edu/~yao/publications/earlystop.pdf
[91] Zhang, T. and Yu, B. (2005). Boosting with early stopping: convergence and consistency. *The Annals of Statistics* **33** 1538–1579.
[92] Zhao, P. and Yu, B. (2005). Boosted Lasso. Tech. rep., University of California, Berkeley.
[93] Zhao, P. and Yu, B. (2006). On model selection consistency of Lasso. *Journal of Machine Learning Research* **7** 2541–2563.
[94] Zhu, J., Rosset, S., Zou, H. and Hastie, T. (2005). Multiclass Adaboost. Tech. rep., Stanford University.
URL http://www-stat.stanford.edu/~hastie/Papers/samme.pdf
[95] Zou, H. (2006). The adaptive Lasso and its oracle properties. *Journal of the American Statistical Association* **101** 1418–1429.

Seminar für Statistik
ETH Zürich
CH-8092 Zürich
Switzerland
e-mail: buhlmann@stat.math.ethz.ch

Institut für Medizininformatik,
Biometrie und Epidemiologie
Friedrich-Alexander-Universität
Erlangen-Nürnberg
Waldstrasse 6, D-91054 Erlangen, Germany
e-mail: torsten.hothorn@rzmail.uni-erlangen.de