Twin Boosting:
Improved Feature Selection and Prediction

Peter Bühlmann
ETH Zurich

Torsten Hothorn
LMU München

June 2008

Abstract

We propose Twin Boosting which has much better feature selection behavior than boosting, particularly with respect to reducing the number of false positives (falsely selected features). In addition, for cases with a few important effective and many noise features, Twin Boosting also substantially improves the predictive accuracy of boosting. Twin Boosting is as general and generic as boosting. It can be used with general weak learners and in a wide variety of situations, including generalized regression, classification or survival modeling. Furthermore, it is computationally feasible for large problems with potentially many more features than observed samples. Finally, for the special case of orthonormal linear models, we prove equivalence of Twin Boosting to the adaptive Lasso which yields a theoretical basis for some properties of Twin Boosting.

Key Words: Classification, Gradient descent, High-dimensional data, Regression, Regularization
Running Title: Twin Boosting

1 Introduction

Boosting has attracted much attention in the machine learning community (cf. Schapire, 2002; Meir and Rätsch, 2003, and the references therein) as well as in related areas in statistics (Breiman, 1998, 1999; Friedman et al., 2000), mainly because of its excellent performance and computational attractiveness for large datasets. The main breakthrough came with Freund and Schapire's most successful AdaBoost algorithm for binary classification (Freund and Schapire, 1996, 1997).

Twin boosting is a very generic boosting-based method which most often yields better feature or variable selection than boosting while keeping or even increasing the prediction accuracy. Twin boosting proceeds roughly as follows. A first round of "classical" boosting is done (the first twin); then, in a second round, another boosting process is run (the second twin) which is forced to resemble the one from the first round. Thus, the two rounds are similar, like twins, and hence the name Twin Boosting.

There have been some attempts to make boosting or also related Lasso-methods (Tibshirani, 1996) more powerful, in particular in terms of feature or variable selection but also for prediction. The potential for improvement is mainly given for cases with many

ineffective and a few effective covariates. To deal with many ineffective features, a strong regularization is employed in boosting or in related Lasso-methods, creating a large estimation bias. Proposals to effectively avoid such large biases include Sparse Boosting (Bühlmann and Yu, 2006), conjugate direction boosting (Lutz and Bühlmann, 2006), Lasso with relaxation (Meinshausen, 2007) or the adaptive Lasso (Zou, 2006). Our approach has some similarity to the latter as it encompasses the adaptive Lasso in the very special case of an orthonormal linear model. An aspect of success of some of these methods is their greater flexibility than what is possible with a single regularization parameter in boosting (the number of iterations) or in related Lasso-methods (the penalty parameter). Our new Twin Boosting involves two tuning parameters which can be chosen sequentially rather than simultaneously optimizing over a two-dimensional space of regularization parameters.

All these recently proposed methods mentioned above, aiming to improve over boosting or Lasso, are far less general and generic than our Twin Boosting approach. The latter can be easily used with any real-valued weak learner which is a (possibly crude) estimator of the conditional mean function. In particular, Twin Boosting can be easily used with trees and hence, it can be applied to data of mixed types with continuous, ordinal and categorical features. Secondly, Twin Boosting can be used for a rich class of loss functions, including squared and absolute error for regression, logistic or exponential loss for classification or Poisson-loss for count data. Therefore, Twin Boosting is essentially as general and generic as boosting.

2 Boosting algorithms

Boosting is used in supervised learning from data $(X_i, Y_i), \ldots, (X_n, Y_n)$, where $X_i \in \mathcal{X}$ is a p-dimensional predictor variable and $Y_i \in \mathcal{Y}$ is a univariate response variable. The space of predictor variables $\mathcal{X}$ is often a subset of $\mathbb{R}^p$ and the space of response variables often a subspace of $\mathbb{R}$, e.g. $\mathcal{Y} = \{-, 1+1\}$ for binary classification or $\mathcal{Y} = \mathbb{R}$ for Gaussian regression.

A boosting algorithm needs the specification of a loss function and a weak learner. Regarding the former, consider a loss function

$$\rho : \mathcal{Y} \times \mathcal{X} \to \mathbb{R},$$

(or a subset of $\mathbb{R}$; and we exclude here the case where the range of $\rho(\cdot, \cdot)$ is multivariate). The loss function is assumed to be differentiable (almost everywhere) and typically convex with respect to the second argument. Examples include squared error loss $\rho(y, f) = |y - f|^2/2$ with $y \in \mathbb{R}$ for regression or the logistic loss $\rho(y, f) = \log_2(1 + \exp(-2yf))$ with $y \in \{-1, +1\}$ for binary classification. The weak learner is in our setting a real-valued function estimator:

$$(X_1, U_1), \ldots, (X_n, U_n) \quad \stackrel{\text{weak learner}}{\longrightarrow} \quad \hat{g}(\cdot),$$

where X_i is the p-dimensional predictor variable and $U_i \in \mathbb{R}$ a pseudo-response variable. The notion of a pseudo-response variable will become clear in the description of the generic

boosting algorithm below. An example of a weak learner is a regression tree yielding a regression function estimate $\hat{g}(\cdot)$.

It is instructive to look at the population minimizer of the loss function:

$$f^*(\cdot) = \text{argmin}_{f(\cdot)} \mathbb{E}[\rho(Y, f(X))], \tag{1}$$

It is itself a function and it is the target of the boosting algorithm. For example, the squared error loss $\rho(y, f) = |y - f|^2/2$ yields the well-known population minimizer $f^*(x) = \mathbb{E}[Y|X = x]$ and boosting algorithms using the squared error loss are estimators of the regression function $f^*(\cdot)$.

2.1 The generic boosting algorithm

Boosting is based on the empirical risk $n^{-1} \sum_{i=1}^n \rho(Y_i, f(X_i))$ and pursuing iterative steepest descent in function space as described below for estimating the unknown function $f^*(\cdot)$. This very general and useful view of boosting has been pioneered by Breiman (1998, 1999) and further developed by Friedman et al. (2000), Rätsch et al. (2001) and Friedman (2001).

Variable or feature selection is pursued in this paper without any statistical significance testing. The selected features is the set of variables which enter explicitly (via the boosting algorithm) in the final function estimate $\hat{f}(\cdot)$.

Generic boosting algorithm

1. Initialize $\hat{f}^{[0]}$: typical values are $\hat{f}^{[0]} \equiv \overline{Y} = n^{-1} \sum_{i=1}^n Y_i$ or $\hat{f}^{[0]} \equiv 0$. Set $m = 0$.

2. Increase m by 1. Compute 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 negative gradient vector $U_1, \ldots, U_n$ by the weak learner

$$(X_1, U_1), \ldots, (X_n, U_n) \quad \overset{\text{weak learner}}{\longrightarrow} \quad \hat{g}^{[m]}(\cdot),$$

   Thus, $\hat{g}^{[m]}(\cdot)$ can be viewed as an approximation of the negative gradient vector.

4. Up-date $\hat{f}^{[m]} = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}^{[m]}(\cdot)$, where $0 < \nu \leq 1$ is a step-length, i.e. proceed along an estimate of the negative gradient vector.

5. Iterate steps 2 - 4 until $m = m_{stop}$ for some stopping iteration m_{stop}.

The stopping iteration, which is the main tuning parameter, can be estimated via cross-validation. The choice of the step-size 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 found to be potentially better for ν being "sufficiently small", e.g. $\nu = 0.1$ (Friedman, 2001). Friedman (2001) suggests to use an additional line search between steps 3 and 4 (in case of different loss functions $\rho(\cdot, \cdot)$ than squared error): it yields a slightly different algorithm but the additional line search seems unnecessary to pursue for achieving a good estimator $\hat{f}^{[m_{stop}]}(\cdot)$.

2.2 L_2Boosting

L_2Boosting is the generic boosting algorithm when using the squared error loss $\rho(y, f) = |y - f|^2/2$. It has been proposed by Friedman (2001) and further developed and analyzed in Bühlmann and Yu (2003).

L_2Boosting algorithm

1. Initialize $\hat{f}^{[0]}$: the default value is $\hat{f}^{[0]} \equiv \overline{Y} = n^{-1}\sum_{i=1}^n Y_i$. 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 $U_1, \ldots, U_n$ by the weak learner
$$(X_1, U_1), \ldots, (X_n, U_n) \quad \stackrel{\text{weak learner}}{\longrightarrow} \quad \hat{g}^{[m]}(\cdot).$$
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.
5. Iterate steps 2 to 4 until $m = m_{stop}$ for some stopping iteration m_{stop}.

The stopping iteration m_{stop} is the main tuning parameter which can be selected using cross-validation.

Other loss functions, e.g. the logistic loss for classification, and corresponding boosting algorithms are described in Section 5. For reasons of clarity, we will first describe all the ideas and motivation for Twin Boosting for the case of the squared error loss. We will then show in Section 5 that the concepts easily generalize to general loss functions.

3 Twin L_2Boosting for linear models

For expository simplicity, we first describe Twin Boosting for linear models:

$$Y_i = \sum_{j=1}^p \beta_j X_i^{(j)} + \varepsilon_i, \tag{2}$$

where $\varepsilon_1, \ldots, \varepsilon_n$ are independent, identically distributed (i.i.d.), independent from $X_1, \ldots, X_n$, with $\mathbb{E}[\varepsilon_i] = 0$. We sometimes write in short:

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon$$

with $\mathbf{Y}_{n\times 1} = (Y_1, \ldots, Y_n)^T$, $\varepsilon_{n\times 1} = (\varepsilon_1, \ldots, \varepsilon_n)^T$ and $\mathbf{X}_{n\times p} = [X_1, \ldots, X_n]^T$.

Fitting of the linear model in (2) with boosting or Twin Boosting is based on the squared error loss (L_2Boosting) and the componentwise linear least squares weak learner, as described in Section 3.1 below. Already L_2Boosting itself has been proven to be very useful for fitting linear models with potentially many more covariates than samples (Friedman, 2001; Bühlmann, 2006; Bühlmann and Yu, 2006) and we will argue here in which circumstances Twin Boosting is even better.

In the sequel, we will extensively use the following notation:

$$\langle u, v\rangle = \sum_{i=1}^{n} u_i v_i \text{ for some vectors } u, v \in \mathbb{R}^n,$$
$$\|u\|^2 = \langle u, u\rangle = \sum_{i=1}^{n} u_i^2 \text{ for some vector } u \in \mathbb{R}^n.$$

Moreover, denote by $\mathbf{X}^{(j)}$ the jth $n \times 1$ column vector of $\mathbf{X}$.

3.1 Componentwise linear least squares as weak learner

Consider the following weak learner based on data $(X_1, U_1), \ldots, (X_n, U_n)$:

$$\hat{g}(x) = \hat{\gamma}_{\hat{\mathcal{S}}} x^{(\hat{\mathcal{S}})},$$
$$\hat{\gamma}_j = \langle \mathbf{U}, \mathbf{X}^{(j)}\rangle / \|\mathbf{X}^{(j)}\|^2,\ \hat{\mathcal{S}} = \operatorname*{arg\,min}_{1\le j\le p} \sum_{i=1}^{n} (U_i - \hat{\gamma}^{(j)} X_i^{(j)})^2. \tag{3}$$

It selects and fits the best predictor variable in a simple linear model in the sense of ordinary least squares fitting. For computational implementation as well as for the construction our Twin Boosting, it is useful to represent the selected predictor variable as:

$$\hat{\mathcal{S}} = \operatorname*{arg\,max}_{1\le j\le p} |\langle \mathbf{U}, \mathbf{X}^{(j)}\rangle|^2 / \|\mathbf{X}^{(j)}\|^2. \tag{4}$$

The derivation of (4) is straightforward.

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 $\hat{f}^{[m]}(\cdot)$ linearly as described in Section 2.2. For componentwise linear least squares, we can simply up-date the parameter vector of a linear model.

L_2Boosting with componentwise linear least squares

1. Initialize $\hat{\beta}^{[0]}$. Set $m = 0$.
2. Increase m by 1. Compute the residuals $U_i = Y_i - \overline{Y} - (\mathbf{X}\hat{\beta}^{[m-1]})_i$ for $i = 1, \ldots, n$.
3. Compute $\hat{\mathcal{S}}_m$ as in (4) and $\hat{\gamma}_{\hat{\mathcal{S}}_m}$ as in (3).
4. Up-date
$$\hat{\beta}^{[m]} = \hat{\beta}^{[m-1]} + \nu \cdot \hat{\gamma}_{\hat{\mathcal{S}}_m}.$$
The notation should be read that only the $\hat{\mathcal{S}}_m$th component of the coefficient estimate is up-dated.
5. Iterate steps 2 to 4 until $m = m_{stop}$ for some stopping iteration m_{stop}.

The selected variables are corresponding to the indices j for which $\hat{\beta}^{[m_1]} \neq 0$.

3.2 Twin L_2Boosting with componentwise linear least squares

Twin Boosting for linear models uses the squared error loss and componentwise linear least squares as weak learner. For expository simplicity, we consider standardized data where $n^{-1}\sum_{i=1}^n Y_i = 0$, $n^{-1}\sum_{i=1}^n X_i^{(j)} = 0$ and $\sum_{i=1}^n (X_i^{(j)})^2 = 1$ for all $j = 1, \ldots, p$ (for unstandardized data, the algorithm has to be slightly re-formulated). This can always be achieved by centering with the empirical mean and scaling with the empirical standard deviation.

Twin L_2Boosting with componentwise linear least squares

1. Run a first round of L_2Boosting with componentwise linear least squares, using m_1 iterations. Denote the estimated parameter by $\hat{\beta}_{init}^{[m_1]}$.

2. For the second round, run L_2Boosting with componentwise linear least squares but replace formula (4) by

$$\hat{\mathcal{S}} = \text{argmax}_{1\le j\le p} |\langle \mathbf{U}, \mathbf{X}^{(j)} \rangle|^2 |\hat{\beta}_{init;j}^{[m_1]}|^2. \tag{5}$$

   Note that $\|X^{(j)}\|^2 = 1$ because of standardized data. Use m_2 iterations and denote the Twin L_2Boosting estimator by $\hat{\beta}_{TWB}^{[m_2]}$.

The motivation of the multiplier $|\hat{\beta}_{init;j}^{[m_1]}|^2$ in formula (5) is as follows: if a prediction variable is important, it has a larger multiplier $|\hat{\beta}_{init;j}^{[m_1]}|^2$ and hence, it is more likely to be selected in the criterion (5); and vice-versa. In particular, a non-relevant variable with $\hat{\beta}_{init;j}^{[m_1]} = 0$ will not be selected. We will describe in Proposition 1 that for special cases, the selection with the multiplier in (5) is equivalent to the adaptive Lasso, a method which has shown remarkable success for high-dimensional feature selection (Zou, 2006; Huang et al., 2007; Bühlmann and Meier, 2008). Two tuning parameters m_1 and m_2 are involved. Instead of optimizing (e.g. cross-validating) over both parameters simultaneously, we use the following, computationally much faster sequential approach: first, an estimate $\hat{m}_1$ is obtained from cross-validation for L_2Boosting and then, cross-validation for Twin Boosting with fixed $\hat{m}_1$ in the initial estimator yields an estimate $\hat{m}_2$.

3.3 Connections to the Lasso and the adaptive Lasso

There is an intriguing connection between L_2Boosting with componentwise linear least squares and the Lasso (Tibshirani, 1996). The latter is an ℓ^1-penalty method for regression defined by

$$\hat{\beta}(\lambda) = \arg\min_{\beta} n^{-1} \sum_{i=1}^n (Y_i - \beta_0 - \sum_{j=1}^p \beta_j X_i^{(j)})^2 + \lambda \sum_{j=1}^p |\beta_j|. \tag{6}$$

Efron et al. (2004) 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) produces a set of solutions which is approximately equivalent to the set of

Lasso solutions when varying the regularisation parameter λ in Lasso (see also (6) above). The approximate equivalence is derived by representing FSLR and Lasso as two different modifications of their computationally efficient least angle regression (LARS) algorithm. In special cases where the design matrix satisfies a "positive cone condition", FSLR, Lasso and LARS all coincide (Efron et al., 2004, p.425).

Despite the fact that L_2Boosting and Lasso are not equivalent methods in general, the connection between boosting (as a forward, greedy method) and the Lasso (involving convex optimization) is interesting.

Recently, Zou (2006) has proposed the adaptive Lasso, defined as

$$\hat{\beta}(\lambda) = \arg\min_{\beta} n^{-1} \sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_i^{(j)})^2 + \lambda \sum_{j=1}^{p} \frac{|\beta_j|}{|\beta_{init,j}|}, \tag{7}$$

where β_{init} is an initial estimator. Zou (2006) mainly uses ordinary least squares as initial estimator (for cases with reasonable ratio n/p) and he mentions the Ridge estimator as one among several possible alternatives. In addition, he proposed a more general class of estimators, but the specific form in (7) is useful and often sufficient. The adaptive Lasso has two advantages over the Lasso. It yields consistent variable selection without imposing severe restriction on the design matrix, at least for the case with fixed predictor dimension p (Zou, 2006), whereas the Lasso is inconsistent (typically yields too large models) if the design is roughly speaking "strongly correlated" (Meinshausen and Bühlmann, 2006; Zou, 2006; Zhao and Yu, 2006). Secondly, adaptive Lasso yields better predictions if the true underlying model has many ineffective (noise) predictor variables. Both of these advantages are closely related to the motivation of our Twin Boosting, see Section 1.

In case of an orthonormal linear model, i.e. the model (2) with $\sum_{i=1}^{n} X_i^{(j)} X_i^{(k)} = \delta_{jk}$ (Kronecker $\delta_{jk} = 1$ if $j = k$ and 0 otherwise), explicit connections between boosting and Lasso exist. It has been shown (constructively) that L_2Boosting with componentwise linear least squares approximates the solution from Lasso (as $\nu > 0$ tends to zero) which equals the soft-threshold estimator (Bühlmann and Yu, 2006). For Twin Boosting, the following holds.

Proposition 1 *In an orthonormal linear model, Twin Boosting with componentwise linear least squares approximates the adaptive Lasso with $\beta_{init} = \hat{\beta}_{init}^{[m_1]}$, see (7). In particular, when stopping the second round of Twin Boosting after m_2 iterations, there is a corresponding $\lambda > 0$ such that the following holds:*

$$\hat{\beta}_{TWB;j}^{[m_2]} = \hat{\beta}_{TWB;j}^{[m_2]}(\nu) \to sign(Z_j)(|Z_j| - \frac{\lambda}{|\hat{\beta}_{init,j}^{[m_1]}|})_+ \quad (\nu \to 0),$$

where ν denotes the step-size, $(x)_+ = \max(0, x)$ the positive part and $Z_j = (\mathbf{X}^T\mathbf{Y})_j$.

A proof is given in the Appendix. Proposition 1 illustrates that in the simple case of an orthonormal linear model, Twin Boosting yields in the limiting case with $\nu \to 0$ the adaptive Lasso which equals the adaptive soft-threshold estimator. The connection is interesting but the real power of Twin Boosting is its generic applicability to very general weak learners and loss functions.

4 Twin L_2Boosting with trees and general weak learners

The most popular weak learners for boosting are decision trees. Our proposal for Twin Boosting with trees is simple, easy to implement and effective. We could represent trees in terms of basis functions (with indicator functions of rectangles) and then employ Twin Boosting methodology as for linear models. However, such an approach becomes computationally very impractical due to the huge number of basis functions (particularly for larger trees) and in addition, such an approach would (adaptively) encourage sparseness in the space of basis functions rather than sparseness in the space of predictor variables; the latter is often much more interesting for many datasets and scientific problems.

Twin Boosting with any real-valued (regression-type) weak learner is defined as follows.

Twin L_2Boosting with general weak learner

1. Run a first round of L_2Boosting and denote by $\hat{\mathbf{f}}_{init}^{[m_1]} = (\hat{f}_{init}^{[m_1]}(X_1), \ldots, \hat{f}_{init}^{[m_1]}(X_n))$ the fitted function at the data points and by $\hat{\mathcal{V}}^{[m_1]} \subseteq \{1, \ldots, p\}$ the subset of indices corresponding to selected predictor variables, both based on m_1 boosting iterations. (If the weak learner is not doing any variable selection, then $\hat{\mathcal{V}}^{[m_1]} = \{1, \ldots, p\}$ is the full set).

2. For the second round, initialize $\hat{f}^{[0]}$: the default value is $\hat{f}^{[0]} \equiv \overline{Y} = n^{-1}\sum_{i=1}^n Y_i$. Set $m = 0$.

3. Increase m by 1. Compute the residuals $U_i = Y_i - \hat{f}^{[m-1]}(X_i)$ for $i = 1, \ldots, n$.

4. For every subset $\mathcal{W} \subseteq \hat{\mathcal{V}}^{[m_1]}$, [1] fit the residual vector $U_1, \ldots, U_n$ to $X_1^{\mathcal{W}}, \ldots, X_n^{\mathcal{W}}$ with the weak learner; here $X^{\mathcal{W}}$ denotes $\{X^{(j)};\ j \in \mathcal{W}\}$. Denote this fitted function by $\hat{h}_{\mathcal{W}}(\cdot)$ and by $\hat{\mathbf{h}}_{\mathcal{W}} = (\hat{h}_{\mathcal{W}}(X_1), \ldots, \hat{h}_{\mathcal{W}}(X_n))$. Then, choose the best $\mathcal{W}$ according to:

$$\widehat{\mathcal{W}} = \mathrm{argmax}_{\mathcal{W}} C_{\mathcal{W}}^2 (2\langle \mathbf{U}, \hat{\mathbf{h}}_{\mathcal{W}} \rangle - \|\hat{\mathbf{h}}_{\mathcal{W}}\|^2),$$
$$C_{\mathcal{W}} = \langle \hat{\mathbf{f}}_{init}^{[m_1]} - \overline{\hat{f}_{init}^{[m_1]}}, \hat{\mathbf{h}}_{\mathcal{W}} \rangle / \|\hat{\mathbf{h}}_{\mathcal{W}}\|,\ \overline{\hat{f}_{init}^{[m_1]}} = n^{-1}\sum_{i=1}^n \hat{f}_{init}^{[m_1]}(X_i). \qquad (8)$$

   Denote by $\hat{g}^{[m]}(\cdot) = \hat{h}_{\widehat{\mathcal{W}}}(\cdot)$.

5. Up-date $\hat{f}^{[m]} = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}^{[m]}(\cdot)$, where $0 < \nu \leq 1$ is a step-length factor.

6. Iterate steps 3 to 5 until $m = m_2$ for some stopping iteration m_2.

The stopping iteration m_2 is the main tuning parameter (for a given boosting estimator of the first round) which can be selected using cross-validation.

From a computational point of view, it seems awkward to consider all subsets $\mathcal{W} \subseteq \hat{\mathcal{V}}^{[m_1]}$ in step 4. However, if the weak learner does variable selection, selecting at most d predictor variables (e.g. a tree with at most $d+1$ terminal nodes), we only have to consider in step 4 all subsets $\mathcal{W}$ having cardinality $|\mathcal{W}| = d$. For example, when using stumps, $d = 1$ and step 4 becomes:

[1] See also below the modification with random feature subsets.

step 4 for stumps. For every $j \in \hat{\mathcal{V}}^{[m_1]}$, fit the residual vector with a stump and denote it by $\hat{h}_j(\cdot)$. The remaining part of step 4 is then as above.

The selected variables from Twin Boosting arise from the set of variables which occur in $\widehat{\mathcal{W}}$ from step 4 for at least one iteration. For example with trees as weak learner, the selected variables arise from variable selection of the tree-type weak learner during the Twin Boosting iterations.

Using trees as weak learner, $d = 1$ (stumps) or $d = 2$ is often a good choice (see also the discussion below in Section 4.1). More generally, for a tree with $d + 1$ terminal nodes (d splits), the computation in step 4 is of the order $O(dnp|\hat{\mathcal{V}}^{[m_1]}|^d)$. It should be noted here that usually $|\hat{\mathcal{V}}^{[m_1]}| \leq \min(n, p)$, often even $\ll \min(n, p)$.

In case of large trees as weak learner, we propose the following:

Twin L_2Boosting with random feature subsets

step 4 with random feature subsets. Generate B independent random feature subsets $\mathcal{W} \subseteq \hat{\mathcal{V}}^{[m_1]}$ with cardinality $|\mathcal{W}| = s_{\mathcal{W}}$, i.e. the elements of $\mathcal{W}$ are sampled i.i.d. $\sim$ Uniform($\hat{\mathcal{V}}^{[m_1]}$). Then proceed as in step 4 above.

The value $s_{\mathcal{W}}$ is typically chosen to be of the magnitude of the depth of the tree learner and we use $B = 500$ as default value for the number of random subsets. We also emphasize that, unlike in Random Forests (Breiman, 2001), there is no averaging operation involved over the weak learners (e.g. trees) resulting from the fits with the random feature subsets: here, the best (random) weak learner is selected according to the criterion in (8).

We give now a motivation for the construction in step 4. Consider Twin L_2Boosting for linear models, as described in Section 3.2. There, it happens automatically that the second round of Twin Boosting considers only the set of predictor variables $\hat{V}^{[m_1]}$ which has been chosen in the first round. Next, we consider the formula (8). The residual sum of squares is

$$\|\mathbf{U} - \hat{\mathbf{h}}_{\mathcal{W}}\|^2 = \|\mathbf{U}\|^2 - 2\langle \mathbf{U}, \hat{\mathbf{h}}_{\mathcal{W}} \rangle + \|\hat{\mathbf{h}}_{\mathcal{W}}\|^2 = const. - (2\langle \mathbf{U}, \hat{\mathbf{h}}_{\mathcal{W}} \rangle - \|\hat{\mathbf{h}}_{\mathcal{W}}\|^2).$$

L_2Boosting would proceed by choosing the best $\mathcal{W}$ maximizing

$$G(\mathcal{W}) = 2\langle \mathbf{U}, \hat{\mathbf{h}}_{\mathcal{W}} \rangle - \|\hat{\mathbf{h}}_{\mathcal{W}}\|^2. \tag{9}$$

For Twin Boosting, we want to multiply weights $C_{\mathcal{W}}$ into the criterion $G(\mathcal{W})$. For the form of these weights, it is instructive to consider Twin L_2Boosting for linear models with normed predictor variables having $\|\mathbf{X}^{(j)}\|^2 = 1$: there, $\mathcal{W}$ is an element of $\{1, \ldots, p\}$ and $\hat{h}_j(X_i) = \langle \mathbf{U}, \mathbf{X}^{(j)} \rangle X_i^{(j)}$. Using this, we easily obtain for $\mathcal{W} = j$,

$$G(\mathcal{W}) = G(j) = |\langle \mathbf{U}, \mathbf{X}^{(j)} \rangle|^2. \tag{10}$$

Twin Boosting for linear models does nothing else than multiplying $G(j)$ by the weights $\beta^2_{init,j}$. Therefore, we want to multiply $G(\mathcal{W})$ in (9) with the square of a suitable regression coefficient. Our $C_{\mathcal{W}}$ in formula (8) is the standardized regression coefficient when regressing the fitted function from the first round of Twin Boosting $\hat{\mathbf{f}}^{[m_1]}$ against the candidate

estimate $\hat{\mathbf{h}}_{\mathcal{W}}$; the standardization is a multiplication by $\|\hat{\mathbf{h}}_{\mathcal{W}}\|$. The standardization is useful as it implicitly measures the regression coefficient on a scale where $\hat{\mathbf{h}}_{\mathcal{W}}$ would have been standardized to $\|\hat{\mathbf{h}}_{\mathcal{W}}\| \equiv 1$ for all $\mathcal{W}$. We end our motivation of step 4 with the following result.

Proposition 2 *Consider the general Twin L_2Boosting algorithm and choose the componentwise linear least squares weak learner. Then, this algorithm coincides exactly with Twin L_2Boosting for linear models, as described in Section 3.2, provided that the underlying regression model is orthonormal as specified in (2) with $\sum_{i=1}^{n} X_i^{(j)} X_i^{(k)} = \delta_{jk}$.*

A proof is given in the Appendix.

4.1 Stumps and larger trees as weak learners

The choice of the weak learner is usually driven by optimizing the prediction performance. In addition, some structural properties can be another useful criterion as well.

The generic boosting and Twin Boosting estimator is a linear combination of weak learners

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

Therefore, structural properties of the boosting function estimator are given by linear combination of structural characteristics of the weak leaner.

Trees are among the most popular base procedures in machine learning. 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.

When using stumps, i.e., a tree with two terminal nodes, the boosting and Twin 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) $d+1$ terminal nodes results in a nonparametric model having at most interactions of order $d-1$: e.g. for $d = 2$, we would pick up interaction terms between pairs of predictor variables. Thus, if we want to constrain the degree of interactions, we can easily do this by constraining the (maximal) number of nodes in the tree learner. For many real datasets, it seems that low-order interaction (or even additive) models are sufficiently rich for good prediction and interpretation. For example, the naive Bayes classifier or linear discriminant analysis, based on an additive or linear decision function respectively, works surprisingly well in many applications (Jamain and Hand, 2005; Hand, 2006). Also boosting with stumps, yielding an additive model, has proven to be successful in many areas, e.g. winning the performance prediction challenge of the IEEE World Congress on Computational Intelligence 2006 (Lutz, 2006). Thus, we often get good performance with trees having 2 or 3 terminal nodes ($d = 1$ or 2, respectively). With such small values of d, Twin Boosting is computationally fast, as discussed after the description of the Twin Boosting algorithm in Section 4.

5 Other loss functions and generic Twin Boosting

For other loss functions than squared error (i.e. other boosting algorithms than L_2Boosting) we can use the general functional gradient descent approach as described in Section 2.1.

Interesting examples of loss functions include the following. For binary classification with $y \in \{-1, +1\}$, the logistic loss is

$$\rho_{logit}(y, f) = \log_2(1 + \exp(-2yf)), \tag{11}$$

and the exponential loss is

$$\rho_{exp}(y, f) = \exp(-yf). \tag{12}$$

Boosting with the logistic loss or exponential loss function is essentially LogitBoost (Friedman et al., 2000) (also called BinomialBoosting) or AdaBoost (Freund and Schapire, 1996), respectively. For both loss function, the population minimizer is

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

For cases where $Y \in \{0, 1, 2, \ldots\}$, the Poisson log-likelihood is often appropriate:

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

In survival analysis, we can derive the loss function from the partial likelihood in the Cox model (Cox, 1975).

5.1 Generic Twin Boosting with general weak learners

If the loss function $\rho(\cdot, \cdot)$ is differentiable (almost everywhere) with respect to the second argument, the generic boosting algorithm from Section 2.1 can be used.

Generic Twin Boosting is exactly as Twin L_2Boosting from Section 4, except that in step 3, instead of using residuals U_i we will use

$$U_i = -\frac{\partial}{\partial f}\rho(Y, f)|_{f=\hat{f}_{m-1}(X_i)}, \ i = 1, \ldots, n,$$

exactly as in the generic boosting algorithm from Section 2.1. In the special case of the componentwise linear least squares learner, we would modify the residual vector $\mathbf{U}$ in step 2 of the algorithm in Section 3.2. From an implementation point of view, Twin Boosting with general loss functions is as simple as Twin L_2Boosting.

6 Empirical results

We report here some results on Twin Boosting for regression and classification and we compare them with boosting. We will demonstrate that Twin Boosting has a clear advantage over boosting if the truth has many ineffective predictor variables. Given the success of boosting algorithms in many application areas, Twin Boosting exhibits a substantial potential for further improvements over boosting. All of our results are displayed in Figures, giving a better summary how the methods behave as a function of boosting iterations.

6.1 Regression

The response variables Y_i are real-valued and the goal is estimation of the function $\mathbb{E}[Y|X = x]$ or prediction of new observations Y.

6.1.1 Simulated data

Consider the linear model:

$$\begin{aligned} &\text{model in (2) with} \quad && p = 500,\ \beta_1 = 5,\ \beta_j = 0\ (j = 2, \dots p), \\ & && X_i \sim \mathcal{N}_p(0, I) \text{ and } \varepsilon_i \sim \mathcal{N}(0, 1); \qquad (13) \\ &\text{model in (2) with} \quad && p = 500,\ \beta_1 = \ldots = \beta_5 = 1.175,\ \beta_j = 0\ (j = 6, \dots p), \\ & && X_i \sim \mathcal{N}_p(0, \Sigma), \Sigma_{ij} = 0.8^{|i-j|},\ \text{and } \varepsilon_i \sim \mathcal{N}(0, 1). \qquad (14) \end{aligned}$$

Both models (13) and (14) have the same signal to noise ratio $\mathbb{E}[|f(X)|^2]/\mathbb{E}[|\varepsilon|^2]$, where $f(x) = \sum_{j=1}^{p} \beta_j x^{(j)}$. Sample size is chosen as $n = 50$ and the number of independent simulation runs is 100.

We first use L_2Boosting and Twin L_2Boosting for linear models, using the componentwise linear least squares weak learner. The step-length factor is chosen as $\nu = 0.1$ and the number of boosting iterations in the first round of Twin Boosting is chosen as $m_1 = 50$ which is a reasonable value according to the performance of L_2Boosting. Figure 1 displays the mean squared error (MSE) $\mathbb{E}[((\hat{\beta} - \beta)^T X_{new})^2] = \mathbb{E}[(\hat{\beta}_j - \beta_j)^T \Sigma (\hat{\beta}_j - \beta_j)]$, with $\Sigma = \mathrm{Cov}(X)$, (i.e. generalization error) and the number of selected and incorrectly selected predictor variables (false positives), as a function of boosting iterations. Figures 1 and 2 illustrate very clearly that Twin Boosting is substantially better than boosting in terms of variable selection. For the very sparse case in (13), Figure 1 also indicates relevant improvements in terms of prediction. In Table 1, we report some exact numbers.

Figure 1 about here.

Figure 2 about here.

Next, we consider L_2Boosting and Twin L_2Boosting with stumps. The results are displayed in Figures 3 and 4; for Twin Boosting, we used $m_1 = 50$ iterations in the first round.

Figure 3 about here.

Figure 4 about here.

Although boosting (and Twin Boosting) with stumps yields an additive model fit, the weak learner is "mis-specified" (as is often the case in practice). This explains why the mean squared error is much larger than with componentwise linear least squares. Also here, L_2Boosting selects way too many predictor variables while Twin Boosting is very effective and substantially reduces the number of selected variables. Table 1 reports some numerical values. The results shown here are quite representative for many other simulation settings.

model, method	MSE	no. variables	no. incorrect variables
(13), L_2Boost comp. LS	0.22 (0.015)	5.97 (0.124)	4.97 (0.124)
(13), Twin L_2Boost comp. LS	0.05 (0.005)	1.01 (0.010)	0.01 (0.010)
(14), L_2Boost comp. LS	0.40 (0.020)	12.41 (0.165)	7.41 (0.165)
(14), Twin L_2Boost comp. LS	0.35 (0.018)	7.39 (0.167)	2.40 (0.166)
(13), L_2Boost stumps	2.91 (0.087)	10.67 (0.233)	9.67 (0.237)
(13), Twin L_2Boost stumps	2.25 (0.070)	4.52 (0.173)	3.52 (0.173)
(14), L_2Boost stumps	3.45 (0.080)	31.98 (0.352)	26.98 (0.352)
(14), Twin L_2Boost stumps	2.84 (0.069)	11.12 (0.231)	6.13 (0.213)

Table 1: Performances of boosting with componentwise linear least squares (comp. LS) or stumps as weak learners, for models (13) and (14), at stopping iteration which minimizes mean squared error. Mean squared error (MSE), number of selected variables/features and number of incorrectly selected variables/features (i.e. false positives). Standard errors are given in parentheses.

6.1.2 Real data

We consider two real data sets: *Ozone* concentration and *Motif* regression from molecular biology. Both data sets are available from

`ftp://ftp.stat.math.ethz.ch/Research-Reports/Other-Manuscripts/buhlmann/ozone.dat`
`ftp://ftp.stat.math.ethz.ch/Research-Reports/Other-Manuscripts/buhlmann/motif.dat`

respectively.

The *Ozone* data is about daily ozone concentration in the Los Angeles basin as a function of $p = 8$ meteorological predictor variables. Sample size is $n = 330$. From a prediction point of view, the componentwise linear least squares weak learner is inferior than stumps. Thus, Figure 5 reports only for boosting and Twin Boosting (with $m_1 = 100$ iterations in the first round) with stumps.

Figure 5 about here.

In addition, we look at a synthetically enlarged problem. We add 500 additional, ineffective noise predictor variables $X_{add} \sim \mathcal{N}_{500}(0, I)$. The problem has then dimension $p = 508$ with at most 8 effective predictors. This will enable us to see whether and how many from the obviously ineffective variables will be selected; we do not know whether all of the 8 original predictor variables are effective or not. We refer to an obviously incorrectly selected predictor variable if it is one of the 500 synthetically added predictors. Figure 6 reports the results (with $m_1 = 100$ iterations in the first round of Twin Boosting). Twin L_2Boosting has slightly better prediction performance than L_2Boosting and is much better with respect to obviously incorrectly selected variables.

Figure 6 about here.

The *Motif* regression data models gene expression as a function of MDSCAN motif scores (Conlon et al., 2003, p. 3343: Spellman et al. data, 15th time point). This data is representative for many gene expression - motif scores data-sets, all of them being very

noisy. Our data has $p = 4312$ motif scores (predictor variables) and sample size (number of genes) is $n = 4443$. Figure 7 displays the results for the componentwise linear least squares weak learner.

Figure 7 about here.

Although L_2Boosting is performing as well as Twin L_2Boosting from a prediction point of view, the sparsity of Twin L_2Boosting in terms of selected variables, and hence with a lower number of false positives, is crucial in this application. When using about 600-800 boosting iterations, L_2Boosting selects 144-178 predictors while Twin Boosting uses 41-53 variables only. Biological validation of about 50 potential motifs (cis-regulatory elements) is much more realistic than for 3 times as many candidates. L_2Boosting and Twin L_2Boosting with trees did not improve prediction performance while it selected more predictor variables than what is reported above for componentwise linear least squares.

6.2 Classification

We consider some binary classification problems and use exclusively the logistic loss in (11) for boosting, i.e. Binomial- or LogitBoosting. The classifier is given by $sign(\hat{f}(x))$ where $\hat{f}(\cdot)$ is the estimated function from boosting or Twin Boosting, respectively. This rule is equivalent to classify to the label with larger (conditional) class-probability.

6.2.1 Simulated data

We modify model (13) as follows:

$$\begin{aligned} &p = 500,\ \beta_1 = 2,\ \beta_j = 0\ (j = 2, \dots p), \\ &X_i \sim \mathcal{N}_p(0, I),\ \log(\pi_i/(1-\pi_i)) = \sum_{j=1}^{p} \beta_j X_i^{(j)},\ Y_i \sim \text{Bernoulli}(\pi_i). \end{aligned} \tag{15}$$

Sample size is again chosen as $n = 50$ and the number of independent simulation runs is 100. We reduced the size of the coefficient β_1 in comparison to model (13) to decrease the signal to noise ratio in the problem.

Figure 8 reports the results for Binomial/LogitBoosting and its Twin Boosting version (with $m_1 = 10$ iterations in the first round of Twin Boosting) with componentwise linear least squares (which yields a logistic linear model). The results are qualitatively comparable to the case of regression in Figure 1, demonstrating a clear advantage of Twin Boosting.

Figure 8 about here.

6.2.2 Real data

We consider the *Sonar* dataset ($n = 208,\ p = 60$) from the Statlog project, available from `ftp//ftp.stat.math.ethz.ch/Research-Reports/Other-Manuscripts/buhlmann/sonar.dat`, the *Ionosphere* ($n = 351,\ p = 34$) and the monk dataset (*Monk1*) ($n = 432,\ p = 6$) from

the UCI machine learning repository (`http://www.ics.uci.edu/ mlearn/MLSummary.html`), and the three datasets *Arcene*, *Madelon* and *Gisette* from the NIPS 2003 feature selection challenge (Guyon et al., 2006). We note that *Monk1* is a synthetic dataset: however, it is not generated by ourselves. For the *Sonar* and *Ionosphere* dataset, we also consider synthetically enlarged predictor spaces where we add 500 ineffective predictor variables; for the *Monk1* data, we exclusively consider the case with an enlarged feature space. We consider boosting and Twin Boosting with stumps for the *Sonar*, *Ionosphere*, *Arcene*, *Madelon* and *Gisette* data while for *Monk1*, we use larger trees as weak learners and the corresponding Twin Boosting with random feature subsets as described in Section 4. Logistic linear models, fitted by using the componentwise linear least squares weak learner, were not competitive for all six data-sets.

For the *Sonar* data, we use $m_1 = 100$ iterations in the first round of Twin Boosting. The results are displayed in Figure 9. The classification accuracy is about the same for boosting and Twin Boosting while the latter selects about 30% fewer variables (when using reasonable stopping iterations which differ for the two methods).

Figure 9 about here.

We enlarge the number of features by adding 500 additional, ineffective noise predictor variables $X_{add} \sim \mathcal{N}_{500}(0, I)$. Then, the classification problem involves dimension $p = 560$ with at most 60 effective predictors. We refer to an obviously incorrectly selected predictor variable if it belongs to one of the 500 synthetically added features. Results are given in Figure 10.

Figure 10 about here.

The interpretation is similar (even more in favor of Twin Boosting) as for the original *Sonar* data. In addition, Twin Boosting is much better in terms of obviously incorrectly selected variables.

For the *Ionosphere* data, $m_1 = 500$ iterations in the first round of Twin Boosting is a reasonable value. The results are displayed in Figure 11.

Figure 11 about here.

Twin Boosting has marginally better prediction power while being more sparse in the selected variables. When adding 500 additional, ineffective noise predictor variables $X_{add} \sim \mathcal{N}_{500}(0, I)$ the problem has dimension $p = 534$ with at most 34 effective predictors. Results are displayed in Figure 12, based on $m_1 = 200$ iterations in the first round of Twin Boosting; as above for the *Sonar* data, the obviously incorrectly selected variables can be determined.

Figure 12 about here.

Interestingly, the classification performance does not degrade for both boosting and Twin Boosting. The reason is probably due to the increased resistance of overfitting (e.g. when selecting wrong features) when using the misclassification error (Friedman

et al., 2000, p.400-404) and having a situation with low noise (low misclassification error). Regarding the quality of feature selection, however, Twin Boosting is much better than boosting in terms of selecting obviously incorrect predictor variables.

In addition, LogitBoosting and TwinBoosting are benchmarked on three high-dimensional problems, the *Arcene*, *Gisette*, and *Madelon* datasets, see also
`http://www.nipsfsc.ecs.soton.ac.uk/` for a detailed description. Here, our interest is to compare both algorithms in real high-dimensional situations with respect to their feature selection properties for a varying number of initial boosting steps m_1. The data comes with separate learning and validation samples, and we report performance measures (balanced misclassification error and negative binomial log-likelihood) for the validation samples.

Arcene offers a binary response and $p = 10'000$ features, the learning sample consists of $n_{\text{train}} = 100$ observations, additional $n_{\text{valid}} = 100$ observations are available for validation. The log-likelihood (Figure 13, left panel) suggests to stop LogitBoosting after ca. 25 iterations to prevent overfitting. The balanced misclassification error attains it's minimum after ca. 20-30 iterations as well. Four runs of TwinBoosting (with $m_1 \in \{25, 50, 75, 100\}$) have been performed as well. Based on the validation log-likelihood, between 15 and 20 iterations should be enough, the balanced misclassification error for all four values of m_1 is practically equivalent. Note that the number of selected features is smaller compared to LogitBoosting.

Figure 13 about here.

Roughly the same conclusions can be drawn for the *Madelon* problem ($n_{\text{train}} = 2000$, $n_{\text{valid}} = 600$, $p = 500$), see Figure 14). The optimal number of boosting iterations is smaller for TwinBoosting whereas the prediction performance is slightly better for Boosting. However, TwinBoosting leads to a sparser model. It should be noted that the performance of TwinBoosting seems to be rather robust against different choices of m_1. This is even more pronounced for the *Gisette* problem ($n_{\text{train}} = 6000$, $n_{\text{valid}} = 1000$, $p = 5000$), see Figure 15, where the four different models (based on $m_1 = (100, 200, 300, 400)$) are practically not distinguishable. Boosting (with 400 iterations) requires more than twice as many variables entering the model than TwinBoosting to achieve a similar performance.

Figures 14 and 15 about here.

6.2.3 Large trees and random feature subsets in Twin Boosting

For the *Monk1* data, boosting and Twin Boosting with stumps has a cross-validated misclassification error of about 0.25. Boosting with larger trees yields substantial improvements with a misclassification error of 0.03. Thus, for this problem, it is essential to allow for interactions among the predictor variables (which is well known due to the construction of the *Monk1* data).

There are only little differences between boosting and Twin Boosting with larger trees. However, when adding 500 ineffective noise variables $X_{add} \in \{-1, +1\}^{500}$ with independent components and $\mathbb{P}[X_{add}^{(j)} = 1] = 0.5$ for all j (the original 6 predictors are categorical, often binary), the situation is very different. As weak learners, we use trees whose depths are at most 4 (which allows for interactions among 4 predictor variables, at

least). Furthermore, for Twin Boosting, we sample $B = 500$ random feature subsets of size $|\mathcal{W}| = s_{\mathcal{W}} = 4$ (per boosting iteration), see section 4. We use $m_1 = 10$ iterations in the first round of Twin Boosting. In addition, we add an additional iteration to Twin Boosting, termed Triple Boosting: i.e. the Twin Boosting fit (with $m_2 = 50$) is used as initialization and we then proceeded exactly as with Twin Boosting. Figure 16 illustrates the results: Twin Boosting improves upon boosting with respect to prediction and feature selection. Furthermore, Triple Boosting yields additional improvements over Twin Boosting.

Figure 16 about here.

This example with the *Monk1* data (with enlarged feature space) is demonstrating that Twin Boosting with random feature subsets and with larger trees leads to similar qualitative conclusions as for Twin Boosting with stumps. Moreover, we see that further (smaller) gains can be achieved by pursuing Triple Boosting invoking one stage more than Twin Boosting.

7 Conclusions

We proposed Twin Boosting which is as general and generic as boosting. It can be used with general weak learners, for example with trees enabling the applicability for mixed data types with continuous, ordinal and categorical features, and it is suitable in a wide variety of situations, including regression, classification, Poisson regression or survival analysis (using the loss function from the partial likelihood in the Cox model). It is easy to implement and computationally feasible for large problems with potentially very many features (or predictors or covariates) and/or large sample size. Furthermore, it is useful for high-dimensional situations where the number of features is much larger than sample size.

We have empirically shown that Twin Boosting has much better feature or variable selection behavior than boosting. In particular, Twin Boosting leads to sparser solutions which implies a reduction in the number of false positives (fewer falsely selected features): a low number of false positives is sometimes highly desirable, e.g. in computational biology where only a *few* features or variables (e.g. genes) will be biologically validated in follow-up experiments. For cases with a small number of important effective covariates and many noise features, Twin Boosting also improves the predictive accuracy of boosting; for other situations, we never found it worse for prediction than boosting. For the special case of orthonormal linear models, we prove equivalence to the adaptive Lasso (Zou, 2006) which yields a theoretical basis for explaining our general empirical findings for Twin Boosting.

8 Appendix

Proof of Proposition 1.
The proof of Theorem 2 in Bühlmann and Yu (2006) can be adapted. The main modification is needed for formula (22) and its previous 5 lines. We denote in short by $\beta_{init} = \hat{\beta}^{[m_1]}_{init}$ and $\hat{\beta}^{[m]} = \hat{\beta}^{[m]}_{TWB}$.

The residual sum of squares of Twin L_2Boosting at iteration m, denoted by RSS_m, decreases monotonically in m. The difference in residual sum of squares is:

$$RSS_m - RSS_{m+1} = |\langle \mathbf{U}, \mathbf{X}^{(\mathcal{S}_{m+1})} \rangle|^2,$$

where $\mathbf{U}$ denotes the residual vector $\mathbf{Y} - \mathbf{X}\hat{\beta}^{[m]}$ and $\mathcal{S}_{m+1}$ the selected variable in iteration $m+1$. In every step of Twin L_2Boosting, a maximal reduction of the weighted difference in residual sum of squares is used:

$$G_{m+1} = (RSS_m - RSS_{m+1})|\beta_{init,\mathcal{S}_{m+1}}|^2 = |\langle \mathbf{U}, \mathbf{X}^{(\mathcal{S}_{m+1})} \rangle|^2 |\beta_{init,\mathcal{S}_{m+1}}|^2,$$

and the sequence $G_{m+1}, m = 1, 2, \ldots$ is monotonically decreasing (because of the definition of Twin L_2Boosting, the independence of fitting the ith component of β from the jth component ($i \neq j$) and the form of the decay of differences of residual sum of squares). Therefore, every stopping iteration corresponds to a tolerance δ^2 as in formula (22) in Bühlmann and Yu (2006), using here G_{m+1} instead of $RSS_m - RSS_{m+1}$. The remaining part of the proof is exactly as in Bühlmann and Yu (2006): the additional factor $|\beta_{init,i}|^2$ leads to the assertion of Proposition 1. □

Proof of Proposition 2.
By formula (10), we only have to deal with the form of the coefficient $C_{\mathcal{W}} = C_j$ in (8). Denote by $\hat{\gamma}_j = \langle \mathbf{U}, \mathbf{X}^{(j)} \rangle$ the estimated regression coefficient of $\mathbf{U}$ versus $\mathbf{X}^{(j)}$. Then,

$$\begin{aligned} C_j &= \langle \hat{\mathbf{f}}_{init}^{[m_1]}, \hat{\gamma}_j \mathbf{X}^{(j)} \rangle / \|\hat{\gamma}_j \mathbf{X}^{(j)}\| \\ &= \langle \sum_{k=1}^{p} \hat{\beta}_{init,k}^{[m_1]} \mathbf{X}^{(k)}, \hat{\gamma}_j \mathbf{X}^{(j)} \rangle / \|\hat{\gamma}_j \mathbf{X}^{(j)}\| = \hat{\beta}_{init,j}^{[m_1]} \hat{\gamma}_j / |\hat{\gamma}_j|. \end{aligned}$$

Hence,

$$C_j^2 = (\hat{\beta}_{init,j}^{[m_1]})^2,$$

equaling the factor in formula (5). This completes the proof. □

References

Breiman, L. (1998). Arcing classifiers (with discussion). *Annals of Statistics* **26** 801–849.

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

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

Bühlmann, P. (2006). Boosting for high-dimensional linear models. *Annals of Statistics* **34** 559–583.

Bühlmann, P. and Meier, L. (2008). Discussion of "One-step sparse estimates in nonconcave penalized likelihood models (H. Zou and R. Li, auths.). *Annals of Statistics, to appear* .

Bühlmann, P. and Yu, B. (2003). Boosting with the $l_2$loss: regression and classification. *Journal of the American Statistical Association* **98** 324–339.

Bühlmann, P. and Yu, B. (2006). Sparse boosting. *Journal of Machine Learning Research* **7** 1001–1024.

Conlon, E., Liu, X., Lieb, J. and Liu, J. (2003). Integrating regulatory motif discovery and genome-wide expression analysis. *Proceedings of the National Academy of Sciences USA* **100** 3339–3344.

Cox, D. (1975). Partial likelihood. *Biometrika* **62** 269–276.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004). Least angle regression (with discussion). *Annals of Statistics* **32** 407–451.

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.

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.

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

Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *Annals of Statistics* **28** 337–407.

Guyon, I., Gunn, S., Nikravesh, M. and Zadeh, L. (2006). *Feature Extraction, Foundations and Applications, Studies in Fuzziness and Soft Computing*. Springer, Physica-Verlag, Heidelberg.

Hand, D. (2006). Classifier technology and the illusion of progress (with discussion). *Statistical Science* **21** 1–34.

Huang, J., Ma, S. and Zhang, C.-H. (2007). Adaptive Lasso for sparse high-dimensional regression models. *Statistica Sinica, to appear* .

Jamain, A. and Hand, D. (2005). The naive bayes mystery. *Pattern Recognition Letters* **26** 1752–1760.

Lutz, R. (2006). Logitboost with trees applied to the wcci 2006 performance prediction challenge datasets. In *Proceedings of the IJCNN 2006*.

Lutz, R. W. and Bühlmann, P. (2006). Conjugate direction boosting. *Journal Computational and Graphical Statistics* **15** 287–311.

Meinshausen, N. (2007). Relaxed Lasso. *Computational Statististics & Data Analysis* **52** 374–393.

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

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.

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

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.

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

Zhao, P. and Yu, B. (2006). On model selection consistency of Lasso. *Journal of Machine Learning Research* **7** 2541–2563.

Zou, H. (2006). The adaptive Lasso and its oracle properties. *Journal of the American Statistical Association* **101** 1418–1429.

Peter Bühlmann
Seminar für Statistik
ETH Zurich
buhlmann@stat.math.ethz.ch

Torsten Hothorn
Institut für Statistik
LMU München
Torsten.Hothorn@stat.uni-muenchen.de

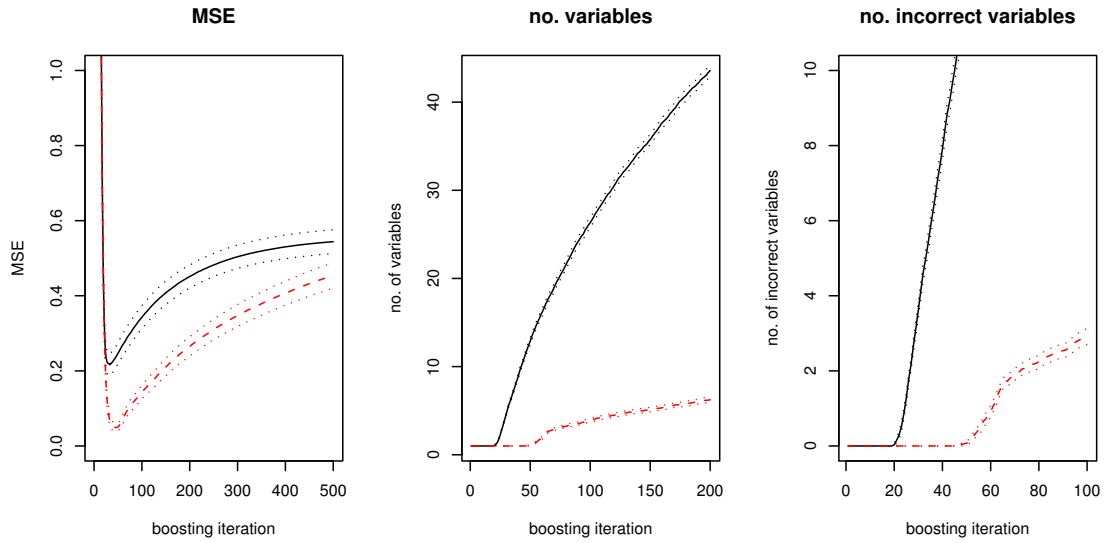

Figure 1: L_2Boosting (solid line) and Twin Boosting (dashed line) with componentwise linear least squares for model (13). Mean squared error (MSE) (left), average number of selected predictor variables (middle) and number of incorrectly selected predictor variables (right) as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively). Simulation accuracy is indicated by dotted lines as 95% confidence intervals.

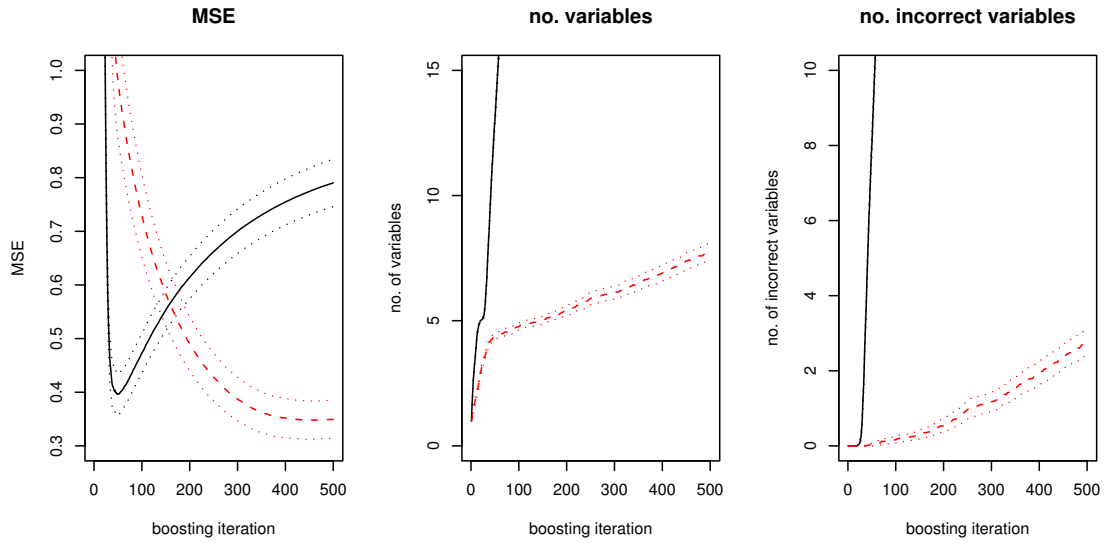

Figure 2: L_2Boosting (solid line) and Twin L_2Boosting (dashed line) with componentwise linear least squares for model (14). Other specifications as in Figure 1.

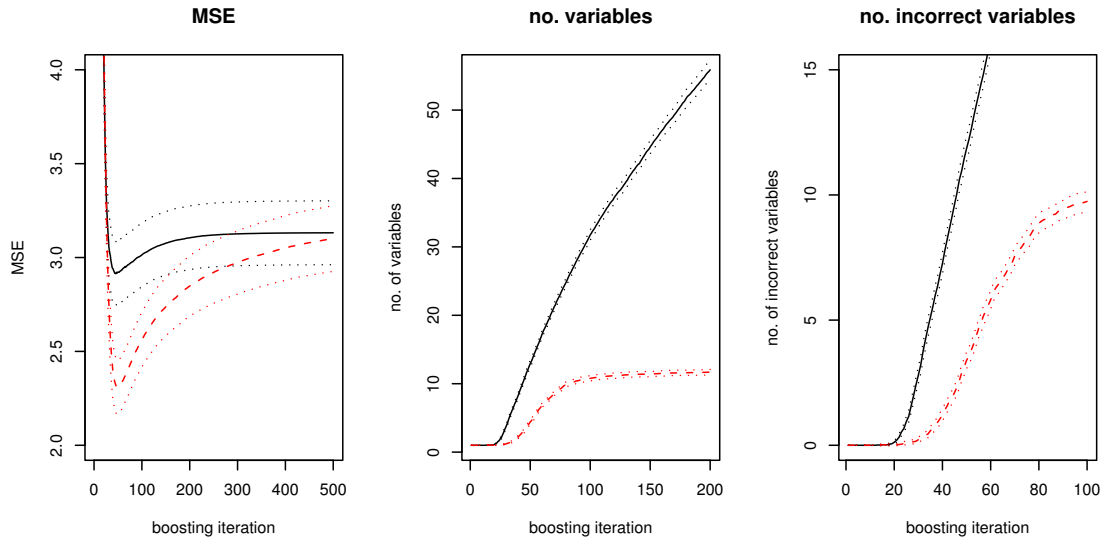

Figure 3: L_2Boosting (solid line) and Twin L_2Boosting (dashed line) with stumps for model (13). Other specifications as in Figure 1.

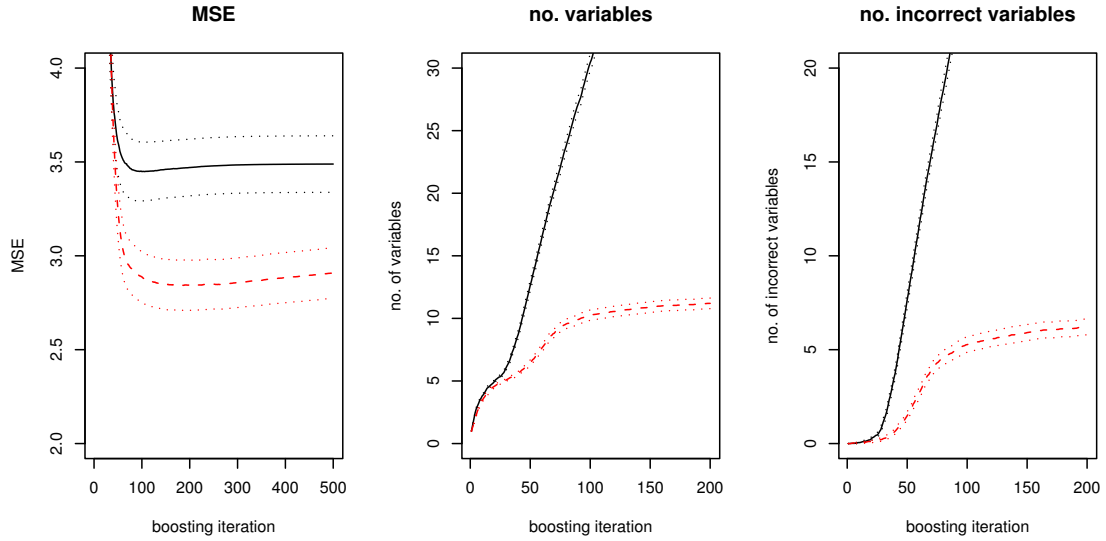

Figure 4: L_2Boosting (solid line) and Twin L_2Boosting (dashed line) with stumps for model (14). Other specifications as in Figure 1.

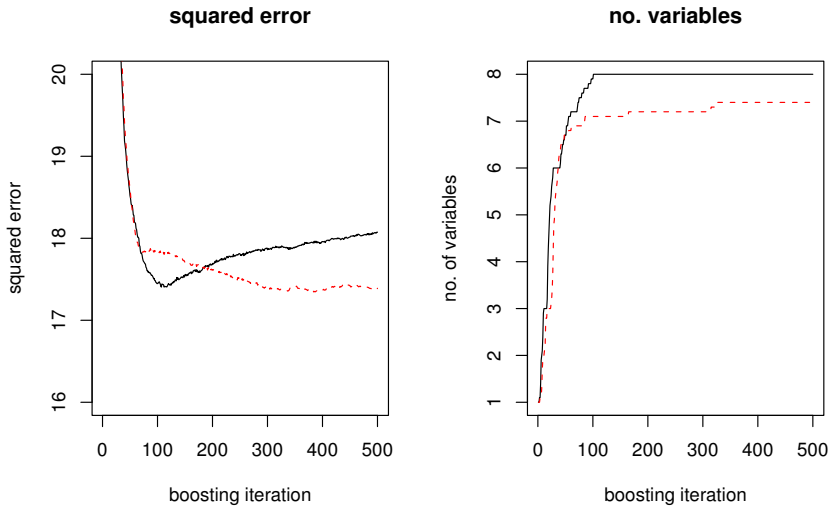

Figure 5: L_2Boosting (solid line) and Twin Boosting (dashed line) with stumps for *Ozone* data. 10-fold cross-validation of: Squared error (left) and number of selected predictor variables (right), as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively).

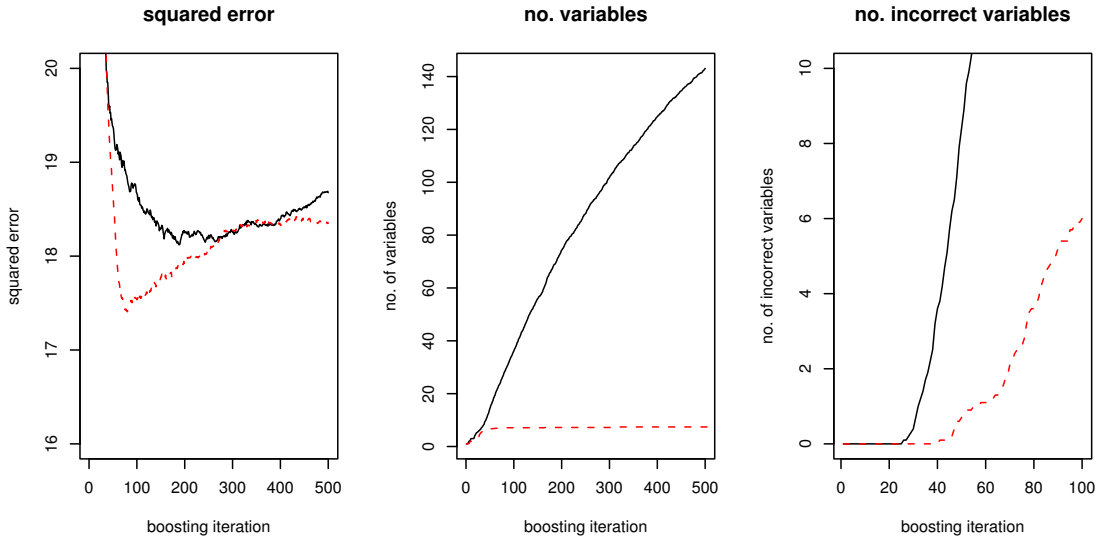

Figure 6: L_2Boosting (solid line) and Twin Boosting (dashed line) with stumps for *Ozone* data with synthetically enlarged predictor space ($p = 508$). 10-fold cross-validation of: Squared error (left), number of selected predictor variables (middle) and number of obviously incorrectly selected variables (right), as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively).

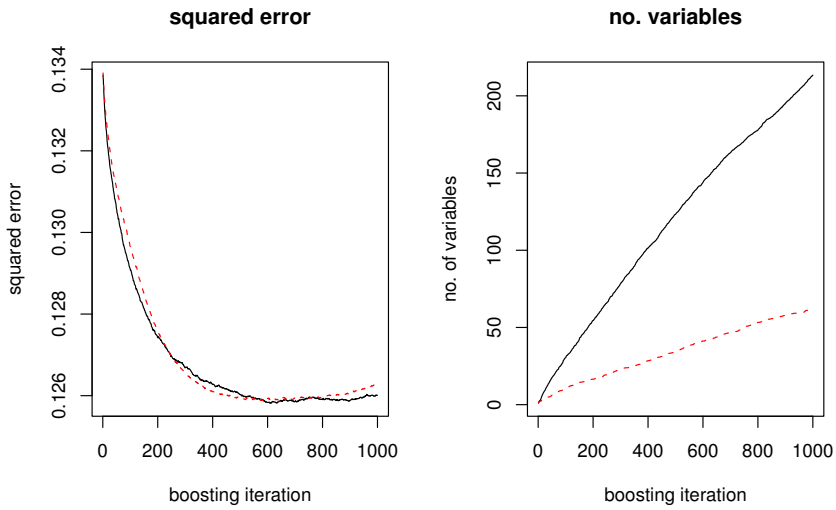

Figure 7: L_2Boosting (solid line) and Twin L_2Boosting (dashed line) with componentwise linear least squares for *Motif* regression data. Other specifications as in Figure 5.

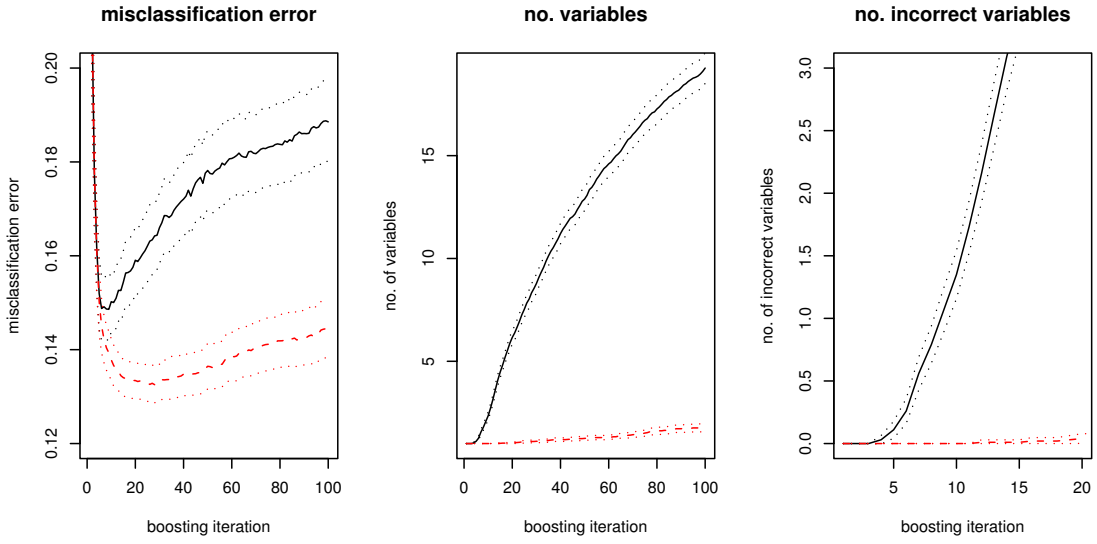

Figure 8: LogitBoosting (solid line) and corresponding Twin Boosting (dashed line) with componentwise linear least squares in model (15). Misclassification error (left), average number of selected predictor variables (middle) and average number of incorrectly selected predictor variables (right) as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively). Simulation accuracy is indicated by dotted lines as 95% confidence intervals.

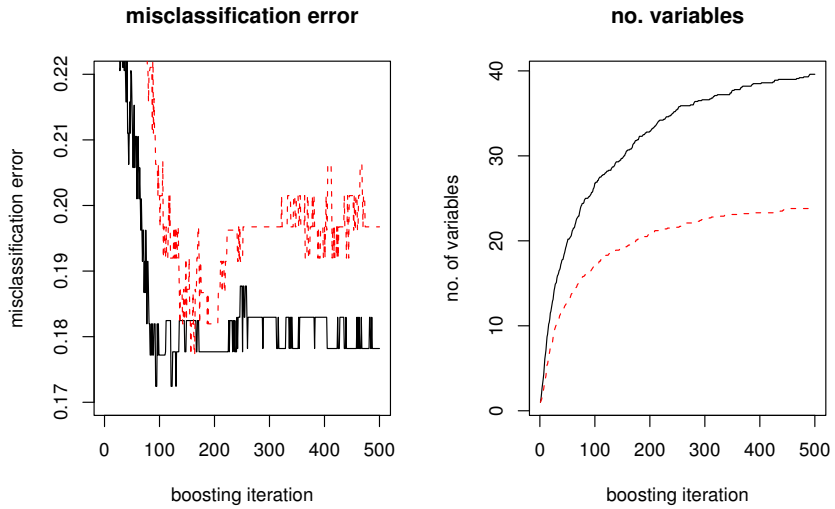

Figure 9: LogitBoosting (solid line) and corresponding Twin Boosting (dashed line) with stumps for *Sonar* data. 10-fold cross-validation of: Misclassification error rate (left) and number of selected predictor variables (right, as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively).

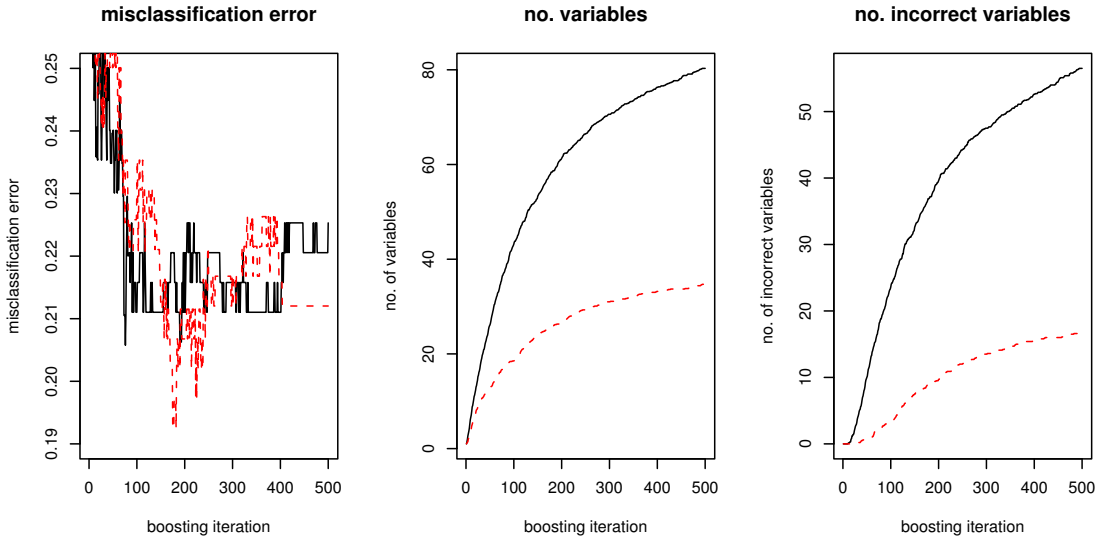

Figure 10: LogitBoosting (solid line) and corresponding Twin Boosting (dashed line) with stumps for *Sonar* data with synthetically enlarged predictor space. 10-fold cross-validation of: Misclassification error rate (left), number of selected predictor variables (middle) and number of obviously incorrectly selected variables (right), as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively).

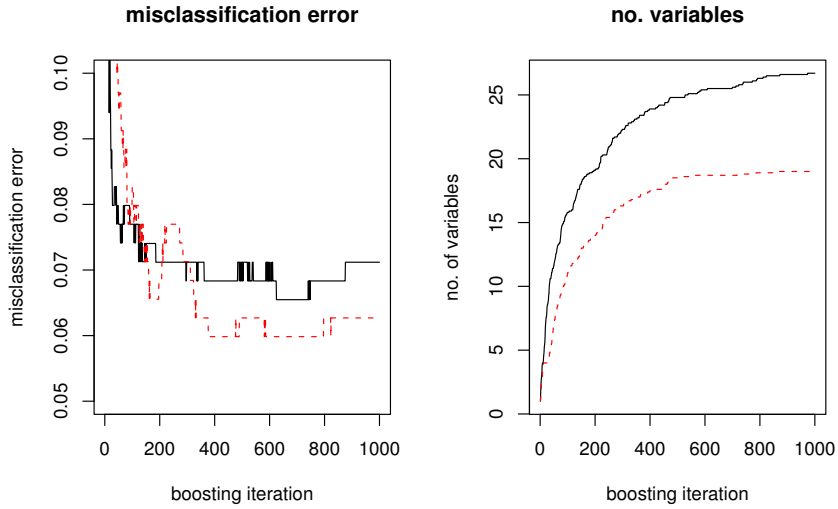

Figure 11: LogitBoosting (solid line) and corresponding Twin Boosting (dashed line) with stumps for *Ionosphere* data. Other specifications as in Figure 9.

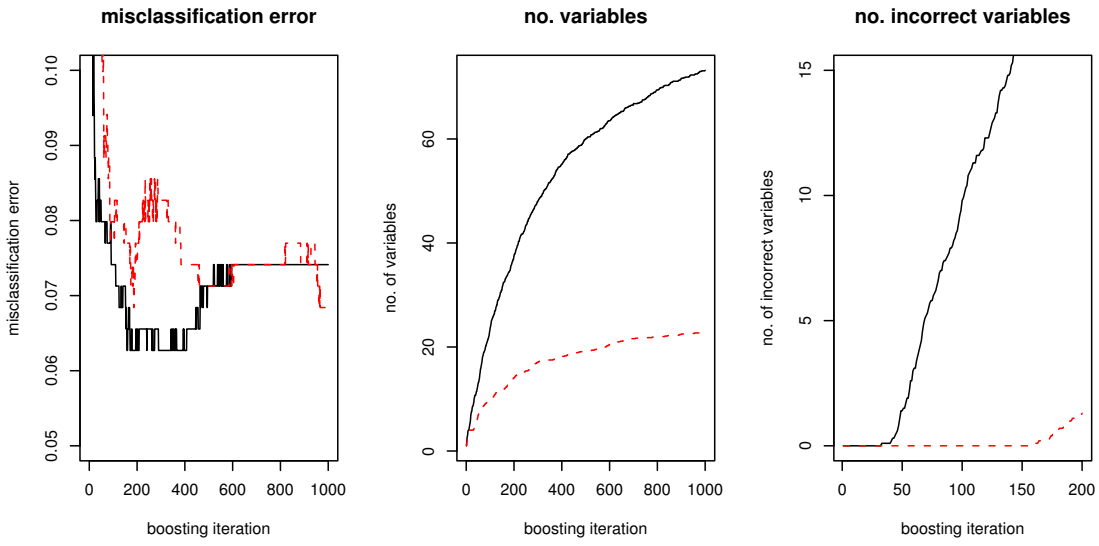

Figure 12: LogitBoosting (solid line) and corresponding Twin Boosting (dashed line) with stumps for *Ionosphere* data with synthetically enlarged predictor space. Other specifications as in Figure 10.

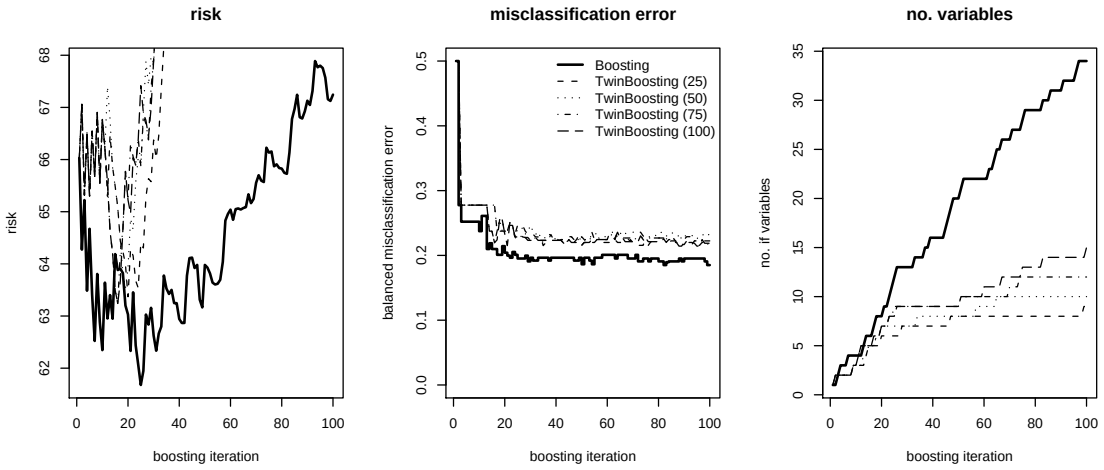

Figure 13: LogitBoosting (solid line) and corresponding Twin Boosting, using various values of m_1 (various dashed lines), with stumps for *Arcene* data. Validation set error of log-likelihood (left) and misclassification rate (middle), and number of selected variables (right), as a function of boosting iterations (or iterations from the second round in Twin Boosting, respectively).

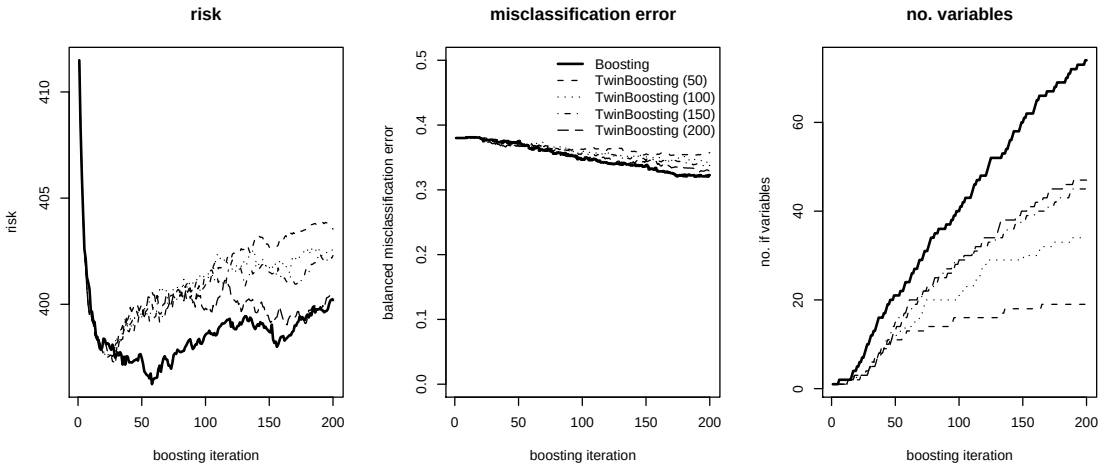

Figure 14: LogitBoosting (solid line) and corresponding Twin Boosting, using various values of m_1 (various dashed lines), with stumps for *Madelon* data. Other specifications as in Figure 13.

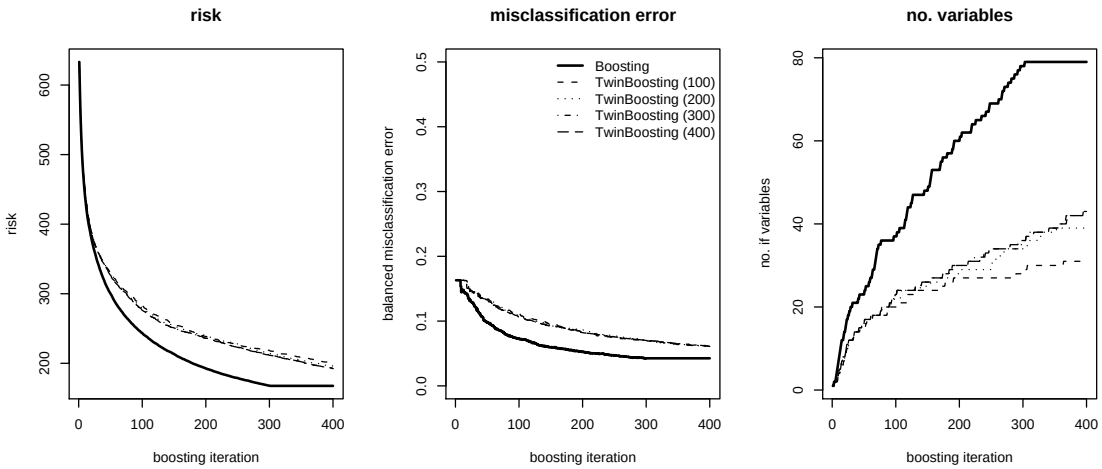

Figure 15: LogitBoosting (solid line) and corresponding Twin Boosting, using various values of m_1 (various dashed lines), with stumps for *Gisette* data. Other specifications as in Figure 13.

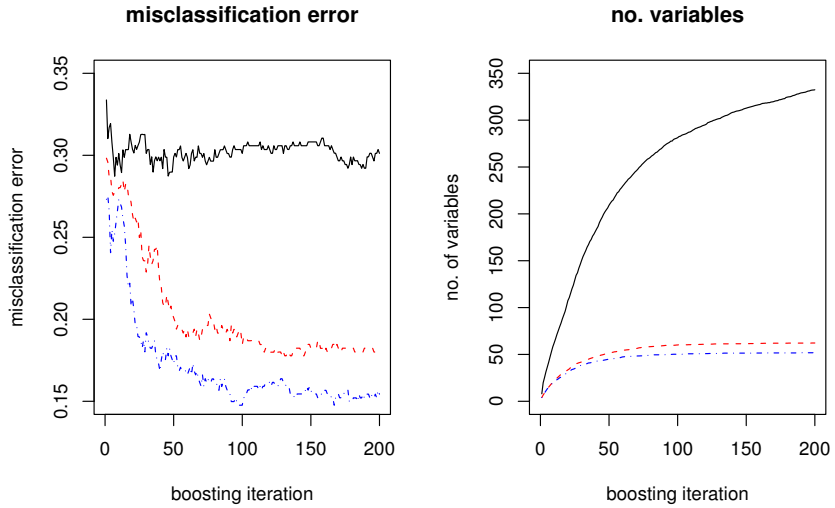

Figure 16: LogitBoosting (solid line), corresponding Twin Boosting (dashed line) and Triple Boosting (dashed-dotted line) with larger trees for *Monk1* data with synthetically enlarged predictor space. Twin and Triple Boosting with random feature subsets. Other specifications as in Figure 10.