Bagging, Boosting and Ensemble Methods

Peter Bühlmann

ETH Zürich, Seminar für Statistik, LEO C17, CH-8092 Zürich, Switzerland
buhlmann@stat.math.ethz.ch

1 An introduction to ensemble methods

Ensemble methods aim at improving the predictive performance of a given statistical learning or model fitting technique. The general principle of ensemble methods is to construct a linear combination of some model fitting method, instead of using a single fit of the method.

More precisely, consider for simplicity the framework of function estimation. We are interested in estimating a real-valued function

$$g : \mathbb{R}^d \to \mathbb{R}$$

based on data $(X_1, Y_1), \ldots, (X_n, Y_n)$ where X is a d-dimensional predictor variable and Y a univariate response. Generalizations to other functions $g(\cdot)$ and other data-types are possible. We assume to have specified a *base procedure* which, given some input data (as above), yields an estimated function $\hat{g}(\cdot)$. For example, the base procedure could be a nonparametric kernel estimator (if d is small) or a nonparametric statistical method with some structural restrictions (for $d \geq 2$) such as a regression tree (or class-probability estimates from a classification tree). We can run a base procedure many times when changing the input data: the original idea of ensemble methods is to use reweighted original data to obtain different estimates $\hat{g}_1(\cdot), \hat{g}_2(\cdot), \hat{g}_3(\cdot), \ldots$ based on different reweighted input data. We can then construct an ensemble-based function estimate $g_{ens}(\cdot)$ by taking linear combinations of the individual function estimates $\hat{g}_k(\cdot)$:

$$\hat{g}_{ens}(\cdot) = \sum_{k=1}^{M} c_k \hat{g}_k(\cdot), \qquad (1)$$

where the $\hat{g}_k(\cdot)$ are obtained from the base procedure based on the kth reweighted data-set. For some ensemble methods, e.g. for bagging (see section 2), the linear combination coefficients $c_k \equiv 1/M$ are averaging weights;

for other methods, e.g. for boosting (see section 3), $\sum_{k=1}^{M} c_k$ increases as M gets larger.

Ensemble methods became popular as a relatively simple device to improve the predictive performance of a base procedure. There are different reasons for this: the bagging procedure turns out to be a variance reduction scheme, at least for some base procedures. On the other hand, boosting methods are primarily reducing the (model) bias of the base procedure. This already indicates that bagging and boosting are very different ensemble methods. We will argue in sections 3.1 and 3.6 that boosting may be even viewed as a non-ensemble method which has tremendous advantages over ensemble (or multiple prediction) methods in terms of interpretation.

Random forests ([Bre99b]) is a very different ensemble method than bagging or boosting. The earliest random forest proposal is from Amit and Geman ([AG97]). From the perspective of prediction, random forests is about as good as boosting, and often better than bagging. For further details about random forests we refer to [Bre99b].

Some rather different exposition about bagging and boosting which describes these methods in the much broader context of many other modern statistical methods can be found in [HTF01].

2 Bagging and related methods

Bagging [Bre96a], a sobriquet for **b**ootstrap **agg**regat**ing**, is an ensemble method for improving unstable estimation or classification schemes. Breiman [Bre96a] motivated bagging as a variance reduction technique for a given base procedure, such as decision trees or methods that do variable selection and fitting in a linear model. It has attracted much attention, probably due to its implementational simplicity and the popularity of the bootstrap methodology. At the time of its invention, only heuristic arguments were presented why bagging would work. Later, it has been shown in [BY02] that bagging is a smoothing operation which turns out to be advantageous when aiming to improve the predictive performance of regression or classification trees. In case of decision trees, the theory in [BY02] confirms Breiman's intuition that bagging is a variance reduction technique, reducing also the mean squared error (MSE). The same also holds for subagging (**sub**sample **agg**regat**ing**), defined in section 2.3, which is a computationally cheaper version than bagging. However, for other (even "complex") base procedures, the variance and MSE reduction effect of bagging is not necessarily true; this has also been shown in [BS02] for the simple case where the estimator is a U-statistics.

2.1 Bagging

Consider the regression or classification setting. The data is given as in section 1: we have pairs (X_i, Y_i) $(i = 1, \ldots, n)$, where $X_i \in \mathbb{R}^d$ denotes the

d-dimensional predictor variable and the response $Y_i \in \mathbb{R}$ (regression) or $Y_i \in \{0, 1, \ldots, J-1\}$ (classification with J classes). The target function of interest is usually $\mathbb{E}[Y|X = x]$ for regression or the multivariate function $\mathbb{P}[Y = j|X = x]$ ($j = 0, \ldots, J-1$) for classification. The function estimator, which is the result from a given base procedure, is

$$\hat{g}(\cdot) = h_n((X_1, Y_1), \ldots, (X_n, Y_n))(\cdot) : \mathbb{R}^d \to \mathbb{R},$$

where the function $h_n(\cdot)$ defines the estimator as a function of the data.

Bagging is defined as follows.

Bagging algorithm

Step 1. Construct a bootstrap sample $(X_1^*, Y_1^*), \ldots, (X_n^*, Y_n^*)$ by randomly drawing n times with replacement from the data $(X_1, Y_1), \ldots, (X_n, Y_n)$.

Step 2. Compute the bootstrapped estimator $\hat{g}^*(\cdot)$ by the plug-in principle: $\hat{g}^*(\cdot) = h_n((X_1^*, Y_1^*), \ldots, (X_n^*, Y_n^*))(\cdot)$.

Step3. Repeat steps 1 and 2 M times, where M is often chosen as 50 or 100, yielding $\hat{g}^{*k}(\cdot)$ ($k = 1, \ldots, M$). The bagged estimator is $\hat{g}_{Bag}(\cdot) = M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}(\cdot)$.

In theory, the bagged estimator is

$$\hat{g}_{Bag}(\cdot) = \mathbb{E}^*[\hat{g}^*(\cdot)]. \tag{2}$$

The theoretical quantity in (2) corresponds to $M = \infty$: the finite number M in practice governs the accuracy of the Monte Carlo approximation but otherwise, it shouldn't be viewed as a tuning parameter for bagging. Whenever we discuss properties of bagging, we think about the theoretical version in (2).

This is exactly Breiman's [Bre96a] definition for bagging regression estimators. For classification, we propose to average the bootstrapped probabilities $\hat{g}_j^{*k}(\cdot) = \hat{\mathbb{P}}^*[Y^{*k} = j|X^{*k} = \cdot]$ ($j = 0, \ldots, J-1$) yielding an estimator for $\mathbb{P}[Y = j|X = \cdot]$, whereas Breiman [Bre96a] proposed to vote among classifiers for constructing the bagged classifier.

The empirical fact that bagging improves the predictive performance of regression and classification trees is nowadays widely documented ([Bre96a], [Bre96b], [BY02], [BD02], [BS02], [HL02]). To give an idea about the gain in performance, we cite some of the results of Breiman's pioneering paper [Bre96a]: for 7 classification problems, bagging a classification tree improved over a single classification tree (in terms of cross-validated misclassification error) by

$$33\%, 47\%, 30\%, 23\%, 20\%, 22\%, 27\%;$$

in case of 5 regression data sets, bagging regression trees improved over a single regression tree (in terms of cross-validated squared error) by

$$39\%, 22\%, 46\%, 30\%, 38\%.$$

In both cases, the size of the single decision tree and of the bootstrapped trees was chosen by optimizing a 10-fold cross-validated error, i.e. using the "usual" kind of tree procedure. Besides that the reported improvement in percentages is quite impressive, it is worth pointing out that bagging a decision tree is almost never worse (in terms of predictive power) than a single tree.

A trivial equality indicates the somewhat unusual approach of using the bootstrap methodology:

$$\hat{g}_{Bag}(\cdot) = \hat{g}(\cdot) + (\mathbb{E}^*[\hat{g}^*(\cdot)] - \hat{g}(\cdot)) = \hat{g}(\cdot) + \text{Bias}^*(\cdot),$$

where $\text{Bias}^*(\cdot)$ is the bootstrap bias estimate of $\hat{g}(\cdot)$. Instead of the usual bias correction with a negative sign, bagging comes along with the wrong sign and *adds* the bootstrap bias estimate. Thus, we would expect that bagging has a higher bias than $\hat{g}(\cdot)$, which we will argue to be true in some sense, see section 2.2. But according to the usual interplay between bias and variance in nonparametric statistics, the hope is to gain more by reducing the variance than increasing the bias, so that overall, bagging would pay-off in terms of the MSE. Again, this hope turns out to be true for some base procedures. In fact, Breiman [Bre96a] described heuristically the performance of bagging as follows: the variance of the bagged estimator $\hat{g}_{Bag}(\cdot)$ should be equal or smaller than that for the original estimator $\hat{g}(\cdot)$; and there can be a drastic variance reduction if the original estimator is "unstable".

2.2 Unstable estimators with hard decision indicator

Instability often occurs when hard decisions with indicator functions are involved as in regression or classification trees. One of the main underlying ideas why bagging works can be demonstrated by a simple example.

Toy example: a simple, instructive analysis

Consider the estimator

$$\hat{g}(x) = \mathbf{1}_{[\overline{Y}_n \le x]}, \;\; x \in \mathbb{R}, \tag{3}$$

where $\overline{Y}_n = n^{-1}\sum_{i=1}^{n} Y_i$ with $Y_1, \ldots, Y_n$ i.i.d. (no predictor variables X_i are used for this example). The target we have in mind is $g(x) = \lim_{n\to\infty} \mathbb{E}[\hat{g}(x)]$. A simple yet precise analysis below shows that bagging is a smoothing operation. Due to the central limit theorem we have

$$n^{1/2}(\overline{Y}_n - \mu) \to_D \mathcal{N}(0, \sigma^2) \;\; (n \to \infty) \tag{4}$$

with $\mu = \mathbb{E}[Y_1]$ and $\sigma^2 = Var(Y_1)$. Then, for x in a $n^{-1/2}$-neighborhood of μ,

$$x = x_n(c) = \mu + c\sigma n^{-1/2}, \tag{5}$$

we have the distributional approximation

$$\hat{g}(x_n(c)) \to_D L(Z) = \mathbf{1}_{[Z \le c]} \ (n \to \infty), \ Z \sim \mathcal{N}(0,1). \tag{6}$$

Obviously, for a fixed c, this is a hard decision function of Z. On the other hand, averaging for the bagged estimator looks as follows. Denote by $\Phi(\cdot)$ the c.d.f. of a standard normal distribution:

$$\begin{aligned}
\hat{g}_{Bag}(x_n(c)) &= \mathbb{E}^*[\mathbf{1}_{[\overline{Y}_n^* \le x_n(c)]}] = \mathbb{E}^*[\mathbf{1}_{[n^{1/2}(\overline{Y}_n^* - \overline{Y}_n)/\sigma \le n^{1/2}(x_n(c) - \overline{Y}_n)/\sigma]}] \\
&= \Phi(n^{1/2}(x_n(c) - \overline{Y}_n)/\sigma) + o_P(1) \\
&\to_D L_{Bag}(Z) = \Phi(c - Z) \ (n \to \infty), \ Z \sim \mathcal{N}(0,1),
\end{aligned} \tag{7}$$

where the first approximation (second line) follows because the bootstrap works for the arithmetic mean $\overline{Y}_n$, i.e.,

$$\sup_{x \in \mathbb{R}} |\mathbb{P}^*[n^{1/2}(\overline{Y}_n^* - \overline{Y}_n)/\sigma \le x] - \Phi(x)| = o_P(1) \ (n \to \infty), \tag{8}$$

and the second approximation (third line in (7)) holds, because of (4) and the definition of $x_n(c)$ in (5). Comparing with (6), bagging produces a soft decision function $L_{Bag}(\cdot)$ of Z: it is a shifted inverse probit, similar to a sigmoid-type function. Figure 1 illustrates the two functions $L(\cdot)$ and $L_{Bag}(\cdot)$. We see that

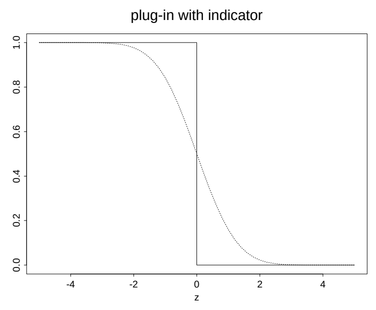


Fig. 1. Indicator estimator from (3) at $x = x_n(0)$ as in (5). Function $L(z) = \mathbf{1}_{[z \le 0]}$ (solid line) and $L_{Bag}(z)$ (dotted line) defining the asymptotics of the estimator in (6) and its bagged version in (7).

bagging is a smoothing operation. The amount of smoothing is determined "automatically" and turns out to be very reasonable (we are not claiming any optimality here). The effect of smoothing is that bagging reduces variance due to a soft- instead of a hard-thresholding operation.

We can compute the first two asymptotic moments in the unstable region with $x = x_n(c)$. Numerical evaluations of these first two moments and

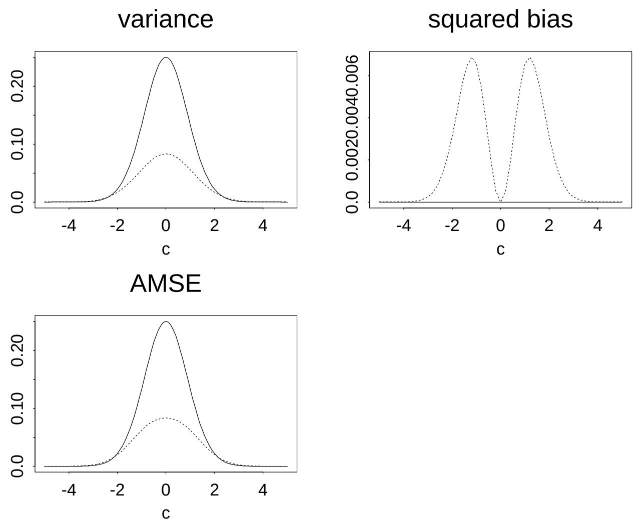


Fig. 2. Indicator estimator from (3) at $x = x_n(c)$ as in (5). Asymptotic variance, squared bias and mean squared error (AMSE) (the target is $\lim_{n\to\infty}\mathbb{E}[\hat{g}(x)]$) for the estimator $\hat{g}(x_n(c))$ from (3) (solid line) and for the bagged estimator $\hat{g}_{Bag}(x_n(c))$ (dotted line) as a function of c.

the mean squared error (MSE) are given in Figure 2. We see that in the approximate range where $|c| \le 2.3$, bagging improves the asymptotic MSE. The biggest gain, by a factor 3, is at the most unstable point $x = \mu = \mathbb{E}[Y_1]$, corresponding to $c = 0$. The squared bias with bagging has only a negligible effect on the MSE (note the different scales in Figure 2). Note that we always give an a-priori advantage to the original estimator which is asymptotically unbiased for the target as defined.

In [BY02], this kind of analysis has been given for more general estimators than $\overline{Y}_n$ in (3) and also for estimation in linear models after testing. Hard decision indicator functions are involved there as well and bagging reduces variance due to its smoothing effect. The key to derive this property is always the fact that the bootstrap is asymptotically consistent as in (8).

Regression trees

We address here the effect of bagging in the case of decision trees which are most often used in practice in conjunction with bagging. Decision trees consist of piecewise constant fitted functions whose supports (for the piecewise constants) are given by indicator functions similar to (3). Hence we expect bagging to bring a significant variance reduction as in the toy example above.

For simplicity of exposition, we consider first a one-dimensional predictor space and a so-called regression stump which is a regression tree with one split and two terminal nodes. The stump estimator (or algorithm) is then defined as the decision tree,

$$\hat{g}(x) = \hat{\beta}_\ell \mathbf{1}_{[x<\hat{d}]} + \hat{\beta}_u \mathbf{1}_{[x\geq\hat{d}]} = \hat{\beta}_\ell + (\hat{\beta}_u - \hat{\beta}_\ell)\mathbf{1}_{[\hat{d}\leq x]}, \tag{9}$$

where the estimates are obtained by least squares as

$$(\hat{\beta}_\ell, \hat{\beta}_u, \hat{d}) = \operatorname{argmin}_{\beta_\ell,\beta_u,d} \sum_{i=1}^{n} (Y_i - \beta_\ell \mathbf{1}_{[X_i<d]} - \beta_u \mathbf{1}_{[X_i\geq d]})^2.$$

These values are estimates for the best projected parameters defined by

$$(\beta_\ell^0, \beta_u^0, d^0) = \operatorname{argmin}_{\beta_\ell,\beta_u,d} \mathbb{E}[(Y - \beta_\ell \mathbf{1}_{[X<d]} - \beta_u \mathbf{1}_{[X\geq d]})^2]. \tag{10}$$

The main mathematical difference of the stump in (9) to the toy estimator in (3) is the behavior of $\hat{d}$ in comparison to the behavior of $\overline{Y}_n$ (and not the constants $\hat{\beta}_\ell$ and $\hat{\beta}_u$ involved in the stump). It is shown in [BY02] that $\hat{d}$ has convergence rate $n^{-1/3}$ (in case of a smooth regression function) and a limiting distribution which is non-Gaussian. This also explains that the bootstrap is not consistent, but consistency as in (8) turned out to be crucial in our analysis above. Bagging is still doing some kind of smoothing, but it is not known how this behaves quantitatively. However, a computationally attractive version of bagging, which has been found to perform often as good as bagging, turns out to be more tractable from a theoretical point of view.

2.3 Subagging

Subagging is a sobriquet for **sub**sample **agg**regat**ing** where subsampling is used instead of the bootstrap for the aggregation. An estimator $\hat{g}(\cdot) = h_n((X_1, Y_1), \ldots, (X_n, Y_n))(\cdot)$ is aggregated as follows:

$$\hat{g}_{SB(m)}(\cdot) = \binom{n}{m}^{-1} \sum_{(i_1,\ldots,i_m)\in\mathcal{I}} h_m((X_{i_1}, Y_{i_1}), \ldots, (X_{i_m}, Y_{i_m}))(\cdot),$$

where $\mathcal{I}$ is the set of m-tuples ($m < n$) whose elements in $\{1, \ldots, n\}$ are all distinct. This aggregation can be approximated by a stochastic computation. The subagging algorithm is as follows.

Subagging algorithm

Step 1. For $k = 1, \ldots, M$ (e.g. $M = 50$ or 100) do:

(i) Generate a random subsample $(X_1^{*k}, Y_1^{*k}), \ldots, (X_m^{*k}, Y_m^{*k})$ by randomly drawing m times without replacement from the data $(X_1, Y_1), \ldots, (X_n, Y_n)$ (instead of resampling with replacement in bagging).

(ii) Compute the subsampled estimator
$\hat{g}^{*k}_{(m)}(\cdot) = h_m((X^{*k}_1, Y^{*k}_1), \ldots, (X^{*k}_m, Y^{*k}_m))(\cdot)$.

Step 2. Average the subsampled estimators to approximate
$\hat{g}_{SB(m)}(\cdot) \approx M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}_{(m)}(\cdot)$.

As indicated in the notation, subagging depends on the subsample size m which is a tuning parameter (in contrast to M).

An interesting case is *half subagging* with $m = [n/2]$. More generally, we could also use $m = [an]$ with $0 < a < 1$ (i.e. m a fraction of n) and we will argue why the usual choice $m = o(n)$ in subsampling for distribution estimation [PRW99] is a bad choice. Half subagging with $m = [n/2]$ has been studied also in [BS02]: in case where $\hat{g}$ is a U-statistic, half subagging is exactly equivalent to bagging, and subagging yields very similar empirical results to bagging when the estimator $\hat{g}(\cdot)$ is a decision tree. Thus, if we don't want to optimize over the tuning parameter m, a good choice in practice is very often $m = [n/2]$. Consequently, half subagging typically saves more than half of the computing time because the computational order of an estimator $\hat{g} = \hat{g}_{(n)}$ is usually at least linear in n.

Subagging regression trees

We describe here in a non-technical way the main mathematical result from [BY02] about subagging regression trees.

The underlying assumptions for some mathematical theory are as follows. The data generating regression model is

$$Y_i = g(X_i) + \varepsilon_i, \; i = 1, \ldots, n,$$

where $X_1, \ldots, X_n$ and $\varepsilon_1, \ldots, \varepsilon_n$ are i.i.d. variables, independent from each other, and $\mathbb{E}[\varepsilon_1] = 0, \mathbb{E}|\varepsilon_1|^2 < \infty$. The regression function $g(\cdot)$ is assumed to be smooth and the distribution of X_i and ε_i are assumed to have suitably regular densities.

It is then shown in [BY02] that for $m = [an]$ $(0 < a < 1)$,

$$\limsup_{n \to \infty} \frac{\mathbb{E}[(\hat{g}_{SB(m)}(x) - g(x))^2]}{\mathbb{E}[(\hat{g}_n(x) - g(x))^2]} < 1,$$

for x in suitable neighborhoods (depending on the fraction a) around the best projected split points of a regression tree (e.g. the parameter d^0 in (10) for a stump), and where $g(x) = \lim_{n\to\infty} \mathbb{E}[\hat{g}(x)]$. That is, subagging asymptotically reduces the MSE for x in neighborhoods around the unstable split points, a fact which we may also compare with Figure 2. Moreover, one can argue that globally,

$$\mathbb{E}[(\hat{g}_{SB(m)}(X) - g(X))^2] \overset{\text{approx.}}{<} \mathbb{E}[(\hat{g}(X) - g(X))^2]$$

for n large, and where the expectations are taken also over (new) predictors X.

For subagging with small order $m = o(n)$, such a result is no longer true: the reason is that small order subagging will then be dominated by a large bias (while variance reduction is even better than for fraction subagging with $m = [an],\ 0 < a < 1$).

Similarly as for the toy example in section 2.2, subagging smoothes the hard decisions in a regression tree resulting in reduced variance and MSE.

2.4 Bagging more "smooth" base procedures and bragging

As discussed in sections 2.2 and 2.3, (su-)bagging smoothes out indicator functions which are inherent in some base procedures such as decision trees. For base procedures which are "smoother", e.g. which do not involve hard decision indicators, the smoothing effect of bagging is expected to cause only small effects.

For example, in [BS02] it is proved that the effect of bagging on the MSE is only in the second order term if the base procedure is a U-statistic. Similarly, citing [CH03]: "... when bagging is applied to relatively conventional statistical problems, it cannot reliably be expected to improve performance". On the other hand, we routinely use nowadays "non-conventional" methods: a simple example is variable selection and fitting in a linear model where bagging has been demonstrated to improve predictive performance ([Bre96a]).

In [BD02], the performance of bagging has been studied for MARS, projection pursuit regression and regression tree base procedures: most improvements of bagging are reported for decision trees. In [BY02], it is shown that bagging the basis function in MARS essentially doesn't change the asymptotic MSE. In [Buh03] it is empirically demonstrated in greater detail that for finite samples, bagging MARS is by far less effective - and sometimes very destructive - than bagging decision trees.

(Su-)bagging may also have a positive effect due to averaging over different selected predictor variables; this is an additional effect besides smoothing out indicator functions. In case of MARS, we could also envision that such an averaging over different selected predictor variables would have a positive effect: in the empirical analysis in [Buh03], this has been found to be only true when using a robust version of aggregation, see below.

2.5 Bragging

Bragging stands for **b**ootstrap **r**obust **agg**regat**ing** ([Buh03]): it uses the sample median over the M bootstrap estimates $\hat{g}^{*k}(\cdot)$, instead of the sample mean in Step 3 of the bagging algorithm.

While bragging regression trees was often found to be slightly less improving than bagging, bragging MARS seems better than the original MARS and much better than bagging MARS.

2.6 Out-of-bag error estimation

Bagging "automatically" yields an estimate of the out-of-sample error, sometimes referred to as the generalization error. Consider a loss $\ell(Y, \hat{g}(X))$, measuring the discrepancy between an estimated function $\hat{g}$, evaluated at X, and the corresponding response Y, e.g. $\ell(Y, \hat{g}(X)) = |Y - \hat{g}(X)|^2$. The generalization error is then

$$err = \mathbb{E}[\ell(Y, \hat{g}(X))],$$

where the expectation $\mathbb{E}$ is over the training data $(X_1, Y_1), \ldots, (X_n, Y_n)$ (i.i.d. or stationary pairs), $\hat{g}(\cdot)$ a function of the training data, and (X, Y) is a new test observation, independent from the training data but having the same distribution as one training sample point (X_i, Y_i).

In a bootstrap sample (in the bagging procedure), roughly $exp(-1) \approx 37\%$ of the original observations are left out: they are called "out-of-bag" observations ([Bre96b]). Denote by $\mathcal{B}oot^k$ the original sample indices which were resampled in the kth bootstrap sample; note that the out-of-bag sample observations (in the kth bootstrap resampling stage) are then given by $\{1, \ldots, n\} \setminus \mathcal{B}oot^k$ which can be used as test sets. The out-of-bag error estimate of bagging is then defined as

$$\widehat{err}_{OB} = n^{-1} \sum_{i=1}^{n} N_M^{-1} \sum_{k=1}^{M} \mathbf{1}_{[(X_i, Y_i) \notin \mathcal{B}oot^k]} \ell(Y_i, \hat{g}^{*k}(X_i)),$$
$$N_M = \sum_{k=1}^{M} \mathbf{1}_{[(X_i, Y_i) \notin \mathcal{B}oot^k]}.$$

In [Byl02], a correction of the out-of-bag error estimate is proposed. Out-of-bag estimation can also be used for other tasks, e.g. for more honest class probability estimates in classification when bagging trees ([Bre96b]).

2.7 Disadvantages

The main disadvantage of bagging, and other ensemble algorithms, is the lack of interpretation. A linear combination of decision trees is much harder to interpret than a single tree. Likewise: bagging a variable selection - fitting algorithm for linear models (e.g. selecting the variables using the AIC criterion within the least-squares estimation framework) gives little clues which of the predictor variables are actually important.

One way out of this lack of interpretation is sometimes given within the framework of bagging. In [ET98], the bootstrap has been justified to judge the importance of automatically selected variables by looking at relative appearance-frequencies in the bootstrap runs. The bagging estimator is the average of the fitted bootstrap functions, while the appearance frequencies of selected variables or interactions may serve for interpretation.

2.8 Other references

Bagging may also be useful as a "module" in other algorithms: BagBoosting [BY00] is a boosting algorithm (see section 3) with a bagged base-procedure, often a bagged regression tree. The theory about bagging supports the finding that BagBoosting using bagged regression trees, which have smaller asymptotic MSEs than trees, is often better than boosting with regression trees. This is empirically demonstrated for a problem about tumor classification using microarray gene expression predictors ([Det04]).

Bundling classifiers ([HL02]), which is a more complicated aggregation algorithm but related to bagging, seems to perform better than bagging for classification. In [Rid02], bagging is used in conjunction with boosting (namely for stopping boosting iterations) for density estimation. In [DF03], bagging is used in the unsupervised context of cluster analysis, reporting improvements when using bagged clusters instead of original cluster-outputs.

3 Boosting

Boosting algorithms have been prpoposed in the machine learning literature by Schapire ([Sch90]) and Freund ([Fre95], [FS96]), see also [Sch02]. These first algorithms have been developed as ensemble methods. Unlike bagging which is a parallel ensemble method, boosting methods are sequential ensemble algorithms where the weights c_k in (1) are depending on the previous fitted functions $\hat{g}_1, \ldots, \hat{g}_{k-1}$. Boosting has been empirically demonstrated to be very accurate in terms of classification, notably the so-called AdaBoost algorithm ([FS96]).

We will explain below that boosting can be viewed as a nonparametric optimization algorithm in function space, as first pointed out by Breiman ([Bre98], [Bre99a]). This view turns out to be very fruitful to adapt boosting for other problems than classification, including regression and survival analysis.

Maybe it is worth mentioning here that boosting algorithms have often better predictive power than bagging, cf. [Bre98]; of course, such a statement has to be read with caution, and methods should be tried out on individual data-sets, including e.g. cross-validation, before selecting one among a few methods.

To give an idea, we report here some empirical results from [Bre98] for classification: we show below the gains (in percentage) of boosting trees over bagging trees:

"normal" size data-sets:	$64.3\%, 10.8\%, 20.3\%, -4.6\%, 6.9\%, 16.2\%,$
large data-sets:	$37.5\%, 12.6\%, -50.0\%, 4.0\%, 28.6\%.$

For all data-sets, boosting trees was better than a single classification tree. The biggest loss of 50% for boosting in comparison with bagging is for a data-set with very low misclassification error, where bagging achieves 0.014% and boosting 0.021%.

3.1 Boosting as functional gradient descent

Rather than looking through the lenses of ensemble methods, boosting algorithms can be seen as functional gradient descent techniques ([Bre98],[Bre99a]). The goal is to estimate a function $g : \mathbb{R}^d \to \mathbb{R}$, minimizing an expected loss

$$\mathbb{E}[\ell(Y, g(X))],\ \ell(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+, \tag{11}$$

based on data (X_i, Y_i) $(i = 1, \dots n)$ as in section 2.1. The loss function ℓ is typically assumed to be convex in the second argument. We consider here both cases where the univariate response Y is continuous (regression problem) or discrete (classification problem), since boosting is potentially useful in both cases.

As we will see in section 3.2, boosting algorithms are pursuing a "small" empirical risk

$$n^{-1} \sum_{i=1}^{n} \ell(Y_i, g(X_i))$$

by selecting a g in the linear hull of some function class, i.e. $g(\cdot) = \sum_k c_k g_k(\cdot)$ with $g_k(\cdot)$'s from a function class such as trees.

The most popular loss functions, for regression and binary classification, are given in Table 1.

boosting	loss function	population minimizer for (11)
L_2Boost	$\ell(y, g) = (y - g)^2$	$g(x) = \mathbb{E}[Y \vert X = x]$
LogitBoost	$\ell(y, g) = log_2(1 + \exp(-2(y-1)g))$	$g(x) = 0.5 \cdot \text{logit}(\mathbb{P}[Y = 1 \vert X = x])$
AdaBoost	$\ell(y, g) = \exp(-(2y-1)g)$	$g(x) = 0.5 \cdot \text{logit}(\mathbb{P}[Y = 1 \vert X = x])$

Table 1. The squared error, binomial negative log-likelihood and exponential loss functions and their population minimizers; $\text{logit}(p) = \log(p/(1-p))$.

While the squared error loss is mainly used for regression (see [BY03] for classification with the squared error loss), the log-likelihood and the exponential loss are for binary classification only.

The margin for classification

The form of the log-likelihood loss may be somewhat unusual: we norm it, by using the base 2 so that it "touches" the misclassification error as an upper bound (see Figure 3), and we write it as a function of the so-called *margin* $\tilde{y}g$, where $\tilde{y} = 2y - 1 \in \{-1, 1\}$ is the usual labeling from the machine learning community. Thus, the loss is a function of the margin $\tilde{y}g$ only; and the same is true with the exponential loss and also the squared error loss for classification since

$$(\tilde{y} - g)^2 = \tilde{y}^2 - 2\tilde{y}g + g^2 = 1 - 2\tilde{y}g + (\tilde{y}g)^2,$$

using $\tilde{y}^2 = 1$.

The misclassification loss, or zero-one loss, is $\mathbf{1}_{[\tilde{y}g<0]}$, again a function of the margin, whose population minimizer is $g(x) = \mathbf{1}_{[\mathbf{P}[Y=1|X=x]>1/2]}$. For readers less familiar with the concept of the margin, this can also be understood as follows: the Bayes-classifier which minimizes the misclassification risk is

$$g_{Bayes}(x) = \mathbf{1}_{[\mathbf{P}[Y=1|X=x]>1/2]}.$$

We can now see that a misclassification occurs, if $y = 0$, $g_{Bayes}(x) = 1$ or $y = 1$, $g_{Bayes}(x) = 0$, which is equivalent to $2(y-1)g_{Bayes}(x) < 0$ or $\tilde{y}g_{Bayes}(x) < 0$.

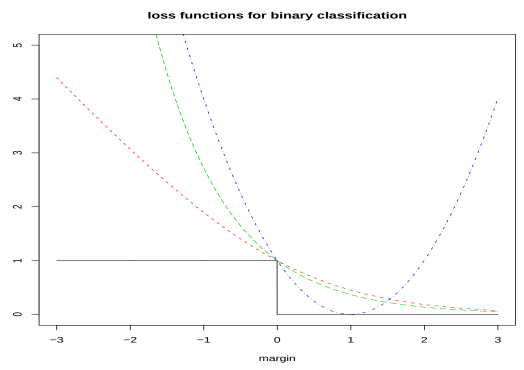


Fig. 3. Loss functions of the margin for binary classification. Zero-one misclassification loss (black), log-likelihood loss (red), exponential loss (green), squared error loss (blue). The loss-functions are described in Table 1.

The (surrogate) loss functions given in Table 1 are all convex functions of the margin $\tilde{y}g$ which bound the zero-one misclassification loss from above, see Figure 3. The convexity of these surrogate loss functions is computationally important for empirical risk minimization; minimizing the empirical zero-one loss is computationally intractable.

3.2 The generic boosting algorithm

Estimation of the function $g(\cdot)$, which minimizes an expected loss in (11), is pursued by a constrained minimization of the empirical risk $n^{-1} \sum_{i=1}^{n} \ell(Y_i, g(X_i))$.

The constraint comes in algorithmically (and not explicitly), by the way we are attempting to minimize the empirical risk, with a so-called functional gradient descent. This gradient descent view has been recognized and refined by various authors (cf. [Bre98], [Bre99a], [MBBF00], [FHT00], [Fri01], [BY03]). In summary, the minimizer of the empirical risk is imposed to satisfy a "smoothness" constraint in terms of a linear expansion of ("simple") fits from a real-valued base procedure function estimate.

Generic functional gradient descent

Step 1 (initialization). Given data $\{(X_i, Y_i); i = 1, \ldots, n\}$, apply the base procedure yielding the function estimate

$$\hat{F}_1(\cdot) = \hat{g}(\cdot),$$

where $\hat{g} = \hat{g}_{X,Y} = h_n((X_1, Y_1), \ldots, (X_n, Y_n))$ is a function of the original data. Set $m = 1$.

Step 2 (projecting gradient to learner). Compute the negative gradient vector

$$U_i = -\frac{\partial \ell(Y_i, g)}{\partial g}|_{g=\hat{F}_m(X_i)}, \; i = 1, \ldots, n,$$

evaluated at the current $\hat{F}_m(\cdot)$. Then, apply the base procedure to the gradient vector

$$\hat{g}_{m+1}(\cdot),$$

where $\hat{g}_{m+1} = \hat{g}_{X,U} = h_n((X_1, U_1), \ldots, (X_n, U_n))$ is a function of the original predictor variables and the current negative gradient vector as pseudo-response.

Step 3 (line search). Do a one-dimensional numerical search for the best step-size

$$\hat{s}_{m+1} = \text{argmin}_s \sum_{i=1}^{n} \ell(Y_i, \hat{F}_m(X_i) + s\hat{g}_{m+1}(X_i)).$$

Update,

$$\hat{F}_{m+1}(\cdot) = \hat{F}_m(\cdot) + \hat{s}_{m+1}\hat{g}_{m+1}(\cdot).$$

Step 4 (iteration). Increase m by one and repeat Steps 2 and 3 until a stopping iteration M is achieved.

The number of iterations M is the tuning parameter of boosting. The larger it is, the more complex the estimator. But the complexity, for example the variance of the estimator, is not linearly increasing in M: instead, it increases very slowly as M gets larger, see also Figure 4 in section 3.5.

Obviously, the choice of the base procedure influences the boosting estimate. Originally, boosting has been mainly used with tree-type base procedures, typically with small trees such as stumps (two terminal nodes) or trees having say 8 terminal nodes (cf. [Bre98], [Bre04], [BK99], [FHT00], [DB03]); see also section 3.8. But we will demonstrate in section 3.6 that boosting may be very worthwhile within the class of linear, additive or interaction models, allowing for good model interpretation.

The function estimate $\hat{g}_{m+1}$ in Step 2 can be viewed as an estimate of $\mathbb{E}[U_i|X = x]$, the expected negative gradient given the predictor X, and takes values in $\mathbb{R}$, even in case of a classification problem with Y_i in a finite set (this is different from the AdaBoost algorithm, see below).

We call $\hat{F}_M(\cdot)$ the L_2Boost-, LogitBoost- or AdaBoost-estimate, according to the implementing loss function $(y-g)^2$, $log_2(1+\exp(-2(y-1)g))$ or $\ell(y,g) = \exp(-(2y-1)g)$, respectively; see Table 1.

The original AdaBoost algorithm for classification is actually a bit different: the base procedure fit is a classifier, and not a real-valued estimator for the conditional probability of Y given X; and Steps 2 and 3 are also somewhat different. Since AdaBoost's implementing exponential loss function is not well established in statistics, we refer for a detailed discussion to [FHT00]. From a statistical perspective, the squared error loss and log-likelihood loss functions are most prominent and we describe below the corresponding boosting algorithms in detail.

L_2Boost

Boosting using the squared error loss, L_2Boost, has a simple structure: the negative gradient in Step 2 is the classical residual vector and the line search in Step 3 is trivial when using a base procedure which does least squares fitting.

L_2Boost algorithm

Step 1 (initialization). As in Step 1 of generic functional gradient descent.

Step 2. Compute residuals $U_i = Y_i - \hat{F}_m(X_i)$ $(i = 1, \ldots, n)$ and fit the real-valued base procedure to the current residuals (typically by (penalized) least squares) as in Step 2 of the generic functional gradient descent; the fit is denoted by $\hat{g}_{m+1}(\cdot)$.
Update

$$\hat{F}_{m+1}(\cdot) = \hat{F}_m(\cdot) + \hat{g}_{m+1}(\cdot).$$

We remark here that, assuming the base procedure does some (potentially penalized) least squares fitting of the residuals, the line search in Step 3 of the generic algorithm becomes trivial with $\hat{s}_{m+1} = 1$.

Step 3 (iteration). Increase iteration index m by one and repeat Step 2 until a stopping iteration M is achieved.

The estimate $\hat{F}_M(\cdot)$ is an estimator of the regression function $\mathbb{E}[Y|X = \cdot]$. L_2Boosting is nothing else than repeated least squares fitting of residuals (cf. [Fri01], [BY03])). With $m = 2$ (one boosting step), it has already been proposed by Tukey ([Tuk77]) under the name "twicing". In the non-stochastic context, the L_2Boosting algorithm is known as "Matching Pursuit" ([MZ93]) which is popular in signal processing for fitting overcomplete dictionaries.

LogitBoost

Boosting using the log-likelihood loss for binary classification (and more generally for multi-class problems) is known as LogitBoost ([FHT00]). LogitBoost uses some Newton-stepping with the Hessian, rather than the line search in Step 3 of the generic boosting algorithm:

LogitBoost algorithm

Step 1 (initialization). Start with conditional probability estimates $\hat{p}_1(X_i) = 1/2$ $(i = 1, \ldots, n)$ (for $\mathbb{P}[Y = 1|X = X_i]$). Set $m = 1$.

Step 2. Compute the pseudo-response (negative gradient)

$$U_i = \frac{Y_i - \hat{p}_m(X_i)}{\hat{p}_m(X_i)(1 - \hat{p}_m(X_i))},$$

and the weights

$$w_i = \hat{p}_m(X_i)(1 - \hat{p}_m(X_i)).$$

Fit the real-valued base procedure to the current pseudo-response U_i $(i = 1, \ldots, n)$ by weighted least squares, using the current weights w_i $(i = 1, \ldots n)$; the fit is denoted by $\hat{g}_{m+1}(\cdot)$. Update

$$\hat{F}_{m+1}(\cdot) = \hat{F}_m(\cdot) + 0.5 \cdot \hat{g}_{m+1}(\cdot)$$

and

$$\hat{p}_{m+1}(X_i) = \frac{\exp(\hat{F}_{m+1}(X_i))}{\exp(\hat{F}_{m+1}(X_i)) + \exp(-\hat{F}_{m+1}(X_i))}.$$

Step 3 (iteration). Increase iteration index m by one and repeat Step 2 until a stopping iteration M is achieved.

The estimate $\hat{F}_M(\cdot)$ is an estimator for half of the log-odds ratio $0.5 \cdot \text{logit}(\mathbb{P}[Y = 1|X = \cdot]$ (see Table 1). Thus, a classifier (under equal misclassification loss for the labels $Y = 0$ and $Y = 1$) is

$$\operatorname{sign}(\hat{F}_M(\cdot)),$$

and an estimate for the conditional probability $\mathbb{P}[Y = 1|X = \cdot]$ is

$$\hat{p}_M(\cdot) = \frac{\exp(\hat{F}_M(\cdot))}{\exp(\hat{F}_M(\cdot)) + \exp(-\hat{F}_M(\cdot))}.$$

A requirement for LogitBoost is that the base procedure has the option to be fitted by *weighted* least squares.

Multi-class problems

The LogitBoost algorithm described above can be modified for multi-class problems where the response variable takes values in a finite set $\{0, 1, \ldots, J - 1\}$ with $J > 2$ by using the multinomial log-likelihood loss ([FHT00]). But sometimes it can be advantageous to run instead a binary classifier (e.g. with boosting) for many binary problems. The most common approach is to code for J binary problems where the jth problem assigns the response

$$Y^{(j)} = \begin{cases} 1, & \text{if } Y = j, \\ 0, & \text{if } Y \neq j. \end{cases}$$

i.e. the so-called "one versus all" approach. For example, if single class-label can be distinguished well from all others, the "one versus all" approach seems adequate: empirically, this has been reported for classifying tumor types based on microarray gene expressions when using a LogitBoost algorithm ([DB03].

Other codings of a multi-class into into multiple binary problems are discussed in [ASS01].

3.3 Small step size

It is often better to use small step sizes instead of using the full line search step-length $\hat{s}_{m+1}$ from Step 3 in the generic boosting algorithm (or $\hat{s}_{m+1} \equiv 1$ for L_2Boost or $\hat{s}_{m+1} \equiv 0.5$ for LogitBoost). We advocate here to use the step-size

$$\nu \hat{s}_{m+1}, \;\; 0 < \nu \leq 1,$$

where ν is constant during boosting iterations and small, e.g. $\nu = 0.1$. The parameter ν can be seen as a simple shrinkage parameter, where we use the shrunken $\nu \hat{g}_{m+1}(\cdot)$ instead of the unshrunken $\hat{g}_{m+1}(\cdot)$. Small step-sizes (or shrinkage) make the boosting algorithm slower and require a larger number M of iterations. However, the computational slow-down often turns out to be advantageous for better out-of-sample prediction performance, cf. [Fri01], [BY03]. There are also some theoretical reasons to use boosting with ν (infinitesimally) small ([EHJT03]).

3.4 The bias-variance trade-off for L_2Boost

We discuss here the behavior of boosting in terms of model-complexity and estimation error when the number of iterations increase. This is best understood in the framework of squared error loss and L_2Boosting.

We represent the base procedure as an operator

$$\mathcal{S} : \mathbb{R}^n \to \mathbb{R}^n,\ (U_1, \dots, U_n)^T \mapsto (\hat{U}_1, \dots, \hat{U}_n)^T$$

which maps a (pseudo-)response vector $(U_1, \dots, U_n)^T$ to its fitted values; the predictor variables X are absorbed here into the operator notation. That is,

$$\mathcal{S}(U_1, \dots, U_n)^T = (\hat{g}(X_1), \dots, \hat{g}(X_n))^T,$$

where $\hat{g}(\cdot) = \hat{g}_{X,U}(\cdot)$ is the estimate from the base procedure based on data $(X_i, U_i),\ i = 1, \dots, n$. Then, the boosting operator in iteration m equals

$$\mathcal{B}_m = I - (I - \mathcal{S})^m$$

and the fitted values of boosting after m iterations are

$$\mathcal{B}_m Y = \mathbf{Y} - (I - \mathcal{S})^m \mathbf{Y},\ \mathbf{Y} = (Y_1, \dots, Y_n)^T.$$

Heuristically, if the base procedure satisfies $\|I - \mathcal{S}\| < 1$ 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 $\mathbf{Y}$ as $m \to \infty$, interpolating the response data exactly. Thus, we have to stop the boosting algorithm at some suitable iteration number $m = M$, and we see that a bias-variance trade-off is involved when varying the iteration number M.

3.5 L_2Boost with smoothing spline base procedure for one-dimensional curve estimation

The case where the base procedure is a smoothing spline for a one-dimensional predictor $X \in \mathbb{R}^1$ is instructive, although being only a toy example within the range of potential applications of boosting algorithms.

In our notation from above, $\mathcal{S}$ denotes a smoothing spline operator which is the solution $(\mathcal{S}\mathbf{Y} = g(X_1), \dots, f(X_n))$ of the following optimization problem (cf. [Wah90])

$$\text{argmin}_g n^{-1} \sum_{i=1}^{n} (Y_i - g(X_i))^2 + \lambda \int g''(x)^2 dx.$$

The smoothing parameter λ controls the bias-variance trade-off, and tuning the smoothing spline estimator usually boils down to estimating a good value of λ. Alternatively, the L_2Boosting approach for curve-estimation with a smoothing spline base procedure is as follows.

Choosing the base procedure

Within the class of smoothing spline base procedures, we choose a spline by fixing a smoothing parameter λ. This should be done such that the base procedure has low variance but potentially high bias: for example, we may choose λ such that the degrees of freedom $df = \mathrm{trace}(\mathcal{S})$ is low, e.g. $df = 2.5$. Although the base procedure has typically high bias, we will reduce it by pursuing suitably many boosting iterations. Choosing the df is not really a tuning parameter: we only have to make sure that df is small enough, so that the initial estimate (or first few boosting estimates) are not already overfitting. This is easy to achieve in practice and a theoretical characterization is described in [BY03]).

Related aspects of choosing the base procedure are described in sections 3.6 and 3.8. The general "principle" is to choose a base procedure which has low variance and having the property that when taking linear combinations thereof, we obtain a model-class which is rich enough for the application at hand.

MSE trace and stopping

As boosting iterations proceed, the bias of the estimator will go down and the variance will increase. However, this bias-variance exhibits a very different behavior as when classically varying the smoothing parameter (the parameter λ). It can be shown that the variance increases with exponentially small

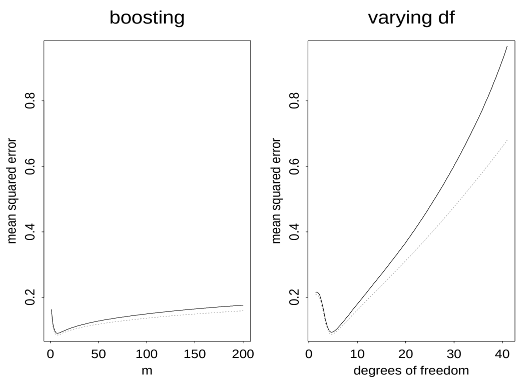


Fig. 4. Mean squared error $\mathbb{E}[(g(X) - \hat{g}(X))^2]$ for new predictor X (solid line) and $n^{-1}\sum_{i=1}^{n}\mathbb{E}[(\hat{F}_m(X_i) - g(X_i))^2]$ (dotted line) from 100 simulations of a nonparametric regression model with smooth regression function and Unif.$[-1/2, 1/2]$-distributed design points. Sample size is $n = 100$. Left: L_2Boost with cubic smoothing spline having $df = 3$, as a function of boosting iterations m. Right: Cubic smoothing spline for various degrees of freedom (various amount of smoothing).

increments of the order $\exp(-Cm)$, $C > 0$, while the bias decays quickly: the optimal mean squared error for the best boosting iteration m is (essentially) the same as for the optimally selected tuning parameter λ ([BY03]), but the trace of the mean squared error is very different, see Figure 4. The L_2Boosting method is much less sensitive to overfitting and hence often easier to tune. The mentioned insensitivity about overfitting also applies to higher-dimensional problems, implying potential advantages about tuning.

Asymptotic optimality

Such L_2Boosting with smoothing splines achieves the asymptotically optimal minimax MSE rates, and the method can even adapt to higher order smoothness of the true underlying function, without knowledge of the true degree of smoothness ([BY03]).

3.6 L_2Boost for additive and interaction regression models

In section 3.4, we already pointed out that L_2Boosting yields another way of regularization by seeking for a compromise between bias and variance. This regularization turns out to be particularly powerful in the context with many predictor variables.

Additive modeling

Consider the component-wise smoothing spline which is defined as a smoothing spline with *one selected* predictor variable $X^{(\hat{\iota})}$ ($\hat{\iota} \in \{1, \ldots, d\}$), where

$$\hat{\iota} = \mathrm{argmin}_{\iota} \sum_{i=1}^{n} (Y_i - \hat{g}_{\iota}(X_i^{(\iota)}))^2,$$

and $\hat{g}_{\iota}$ are smoothing splines with single predictors $X^{(j)}$, all having the same low degrees of freedom df, e.g. $df = 2.5$.

L_2Boost with component-wise smoothing splines yields an additive model, since in every boosting iteration, a function of one selected predictor variable is linearly added to the current fit and hence, we can always rearrange the summands to represent the boosting estimator as an additive function in the original variables, $\sum_{j=1}^{d} \hat{m}_j(x_j)$, $x \in \mathbb{R}^d$. The estimated functions $\hat{m}_j(\cdot)$ are fitted in a stage-wise fashion and they are different from the backfitting estimates in additive models (cf. [HT90]). Boosting has much greater flexibility to add complexity, in a stage-wise fashion: in particular, boosting does variable selection, since some of the predictors will never be chosen, and it assigns variable amount of degrees of freedom to the selected components (or function estimates); the degrees of freedom are defined below. An illustration of this interesting way to fit additive regression models with high-dimensional predictors is given in Figures 5 and 6 (actually, a penalized version of L_2Boosting, as described below, is shown).

When using regression stumps (decision trees having two terminal nodes) as the base procedure, we also get an additive model fit (by the same argument as with component-wise smoothing splines). If the additive terms $m_j(\cdot)$ are smooth functions of the predictor variables, the component-wise smoothing spline is often a better base procedure than stumps ([BY03]). For the purpose of classification, e.g. with LogitBoost, stumps often seem to do a decent job; also, if the predictor variables are non-continuous, component-wise smoothing splines are often inadequate.

Finally, if the number d of predictors is "reasonable" in relation to sample size n, boosting techniques are not necessarily better than more classical estimation methods ([BY03]). It seems that boosting has most potential when the predictor dimension is very high ([BY03]). Presumably, more classical methods become then very difficult to tune while boosting seems to produce a set of solutions (for every boosting iteration another solution) whose best member, chosen e.g. via cross-validation, has often very good predictive performance. A reason for the efficiency of the trace of boosting solutions is given in section 3.9.

Degrees of freedom and AIC_c-stopping estimates

For component-wise base procedures, which pick one or also a pair of variables at the time, all the component-wise fitting operators are involved: for simplicity, we focus on additive modeling with component-wise fitting operators $\mathcal{S}_j$, $j = 1, \ldots, d$, e.g. the component-wise smoothing spline.

The boosting operator, when using the step size $0 < \nu \leq 1$, is then of the form

$$\mathcal{B}_m = I - (I - \nu\mathcal{S}_{\hat{\iota}_1})(I - \nu\mathcal{S}_{\hat{\iota}_2}) \ldots (I - \nu\mathcal{S}_{\hat{\iota}_m}),$$

where $\hat{\iota}_i \in \{1, \ldots, d\}$ denotes the component which is picked in the component-wise smoothing spline in the ith boosting iteration.

If the $\mathcal{S}_j$'s are all linear operators, and ignoring the effect of selecting the components, it is reasonable to define the degrees of boosting as

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

We can represent

$$\mathcal{B}_m = \sum_{j=1}^{d} M_j,$$

where $M_j = M_{j,m}$ is the linear operator which yields the fitted values for the jth additive term, e.g. $M_j\mathbf{Y} = (\hat{m}_j(X_1), \ldots, \hat{m}_j(X_n))^T$. Note that the M_j's can be easily computed in an iterative way by up-dating in the ith boosting iteration as follows:

$$M_{\hat{\iota}_i,new} \leftarrow M_{\hat{\iota}_i,old} + \nu \mathcal{S}_{\hat{\iota}_i}(I - \mathcal{B}_{i-1})$$

and all other $M_j,\ j \neq \hat{\iota}_i$ do not change. Thus, we have a decomposition of the total degrees of freedom into the d additive terms:

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

The individual degrees of freedom $df_{j,m}$ are a useful measure to quantify the complexity of the jth additive function estimate $\hat{m}_j(\cdot)$ in boosting iteration m. Note that $df_{j,m}$ will increase very sub-linearly as a function of boosting iterations m, see also Figure 4.

Having some degrees of freedom at hand, we can now use the AIC, or some corrected version thereof, to define a stopping rule of boosting without doing some sort of cross-validation: the corrected AIC statistic ([HST98]) for boosting in the mth iteration is

$$AIC_c = \log(\hat{\sigma}^2) + \frac{1 + \mathrm{trace}(\mathcal{B}_m)/n}{1 - (\mathrm{trace}(\mathcal{B}_m) + 2)/n}, \tag{12}$$

$$\hat{\sigma}^2 = n^{-1}\sum_{i=1}^{n}(Y_i - (\mathcal{B}_m\mathbf{Y})_i)^2. \tag{13}$$

Alternatively, we could use generalized cross-validation (cf. [HTF01]), which involves degrees of freedom. This would exhibit the same computational advantage, as AIC_c, over cross-validation: instead of running boosting multiple times, AIC_c and generalized cross-validation need only one run of boosting (over a suitable number of iterations).

Penalized L_2 boosting

When viewing the AIC_c criterion in (12) as a reasonable estimate of the true underlying mean squared error (ignoring uninteresting constants), we may attempt to construct a boosting algorithm which reduces in every step the AIC_c statistic (an estimate of the out-sample MSE) most, instead of maximally reducing the in-sample residual sum of squares.

We describe here penalized boosting for additive model fitting using individual smoothing splines:

Penalized L_2Boost with additive smoothing splines

Step 1 (initialization). As in Step 1 of L_2Boost by fitting a component-wise smoothing spline.

Step 2. Compute residuals $U_i = Y_i - \hat{F}_m(X_i)$ $(i = 1, \ldots, n)$. Choose the individual smoothing spline which reduces AIC_c most: denote the selected component by $\hat{\iota}_{m+1}$ and the fitted function, using the selected component $\hat{\iota}_{m+1}$ by $\hat{g}_{m+1}(\cdot)$.
Update

$$\hat{F}_{m+1}(\cdot) = \hat{F}_m(\cdot) + \nu \hat{g}_{m+1}(\cdot).$$

for some step size $0 < \nu \leq 1$.

Step 3 (iteration). Increase iteration index m by one and repeat Step 2 until the AIC_c criterion in (12) cannot be improved anymore.

This algorithm cannot be written in terms of fitting a base procedure multiple times since selecting the component $\hat{\iota}$ in Step 2 not only depends on the residuals $U_1, \ldots, U_n$, but also on the degrees of boosting, i.e. trace$(\mathcal{B}_{m+1})$; the latter is a complicated, although linear function, of the boosting iterations $m' \in \{1, 2, \ldots, m\}$. Penalized L_2Boost yields more sparse solutions than the corresponding L_2Boost (with component-wise smoothing splines as corresponding base procedure). The reason is that $df_{j,m}$ increases only little in iteration $m + 1$, if the jth selected predictor variables has already been selected many times in previous iterations; this is directly connected to the slow increase in variance and overfitting as exemplified in Figure 4.

An illustration of penalized L_2Boosting with individual smoothing splines is shown in Figures 5 and 6, based on simulated data. The simulation model is

$$\begin{aligned} &X_1, \ldots, X_n \text{ i.i.d. } \sim \text{ Unif.}[0,1]^{100}, \\ &Y_i = \sum_{j=1}^{10} m_j(X^{(j)}) + \varepsilon_i \ (i = 1, \ldots, n), \\ &\varepsilon_1, \ldots, \varepsilon_n \text{ i.i.d. } \sim \mathcal{N}(0, 0.5), \end{aligned} \tag{14}$$

where the m_j's are smooth curves having varying curve complexities, as illustrated in Figure 6. Sample size is $n = 200$ which is small in comparison to $d = 100$ (but the effective number of predictors is only 10).

In terms of prediction performance, penalized L_2Boosting is not always better than L_2Boosting; Figure 7 illustrates an advantage of penalized L_2Boosting. But penalized L_2Boosting is always sparser (or at least not less sparse) than the corresponding L_2Boosting.

Obviously, penalized L_2Boosting can be used for other than additive smoothing spline model fitting. The modifications are straightforward as long as the individual base procedures are linear operators.

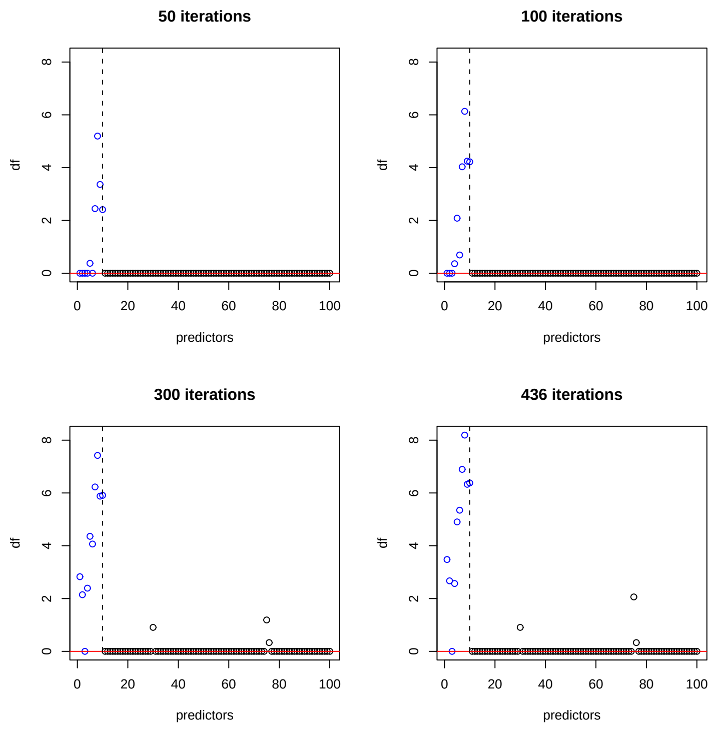


Fig. 5. Degrees of freedom (*df*) in additive model fitting for all 100 predictor variables (from model (14)) during the process of penalized L_2Boosting with individual smoothing splines (having $df = \text{trace}(\mathcal{S}_j) = 2.5$ for each spline). The first ten predictor variables (separated by the dashed line) are effective. The result is based on one realization from model (14) with sample size $n = 200$. The plot on the lower right corresponds to the estimated optimal number of boosting iterations using the AIC_c criterion in (12). Only three non-effective predictors have been selected (and assigned small amount of *df*), and one effective predictor has not been selected (but whose true underlying function is close to the zero-line, see Figure 6).

Interaction modeling

L_2Boosting for additive modeling can be easily extended to interaction modeling (having low degree of interaction). Among the most prominent case is the second order interaction model $\sum_{j,k=1}^{d} \hat{m}_{j,k}(x_j, x_k)$, where $\hat{m}_{j,k} : \mathbb{R}^2 \to \mathbb{R}$.

Boosting with a pairwise thin plate spline, which selects the best pair of predictor variables yielding lowest residual sum of squares (when having the same degrees of freedom for every thin plate spline), yields a second-order interaction model. We demonstrate in Figure 7 the effectiveness of this procedure in comparison with the second-order MARS fit ([Fri91]). The underlying

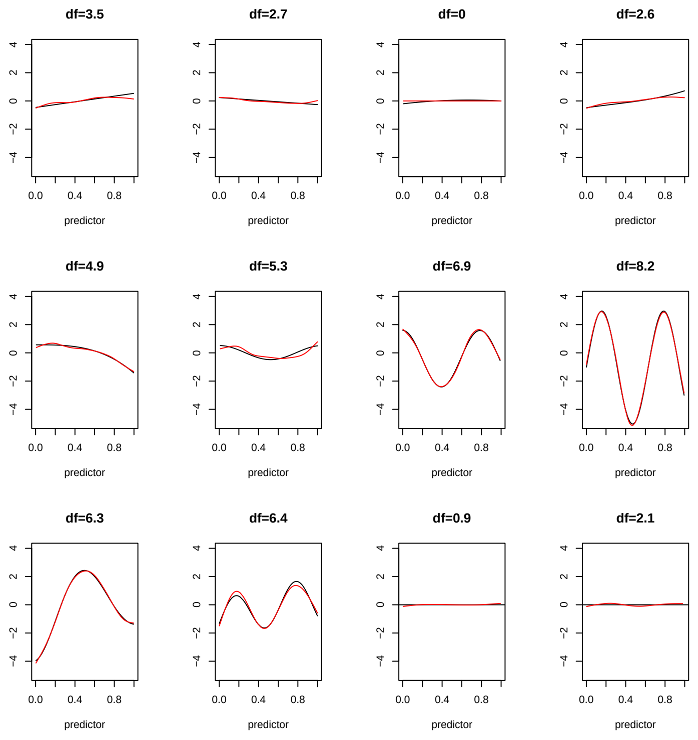


Fig. 6. True underlying additive regression curves (black) and estimates (red) from penalized L_2Boosting as described in Figure 5 (using 436 iterations, estimated from (12)). The last two plots correspond to non-effective predictors (the true functions are the zero-line), where L_2Boosting assigned most *df* among non-effective predictors.

model is the Friedman #1 model:

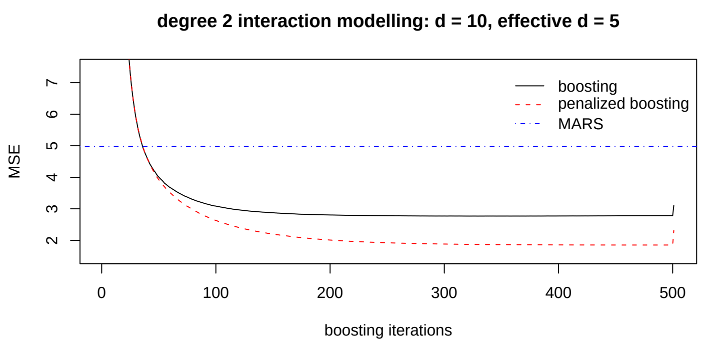
$$\begin{aligned}&X_1,\ldots,X_n \text{ i.i.d. } \sim \text{ Unif.}([0,1]^d),\ d \in \{10,20\},\\&Y_i = 10\sin(\pi X^{(1)}X^{(2)}) + 20(X^{(3)}-0.5)^2 + 10X^{(4)} + 5X^{(5)} + \varepsilon_i\ (i=1,\ldots,n),\\&\varepsilon_1,\ldots,\varepsilon_n \text{ i.i.d } \sim \mathcal{N}(0,1). \end{aligned} \tag{15}$$

The sample size is chosen as $n = 50$ which is small in comparison to $d = 20$.

In high-dimensional settings, it seems that such interaction L_2Boosting is clearly better than the more classical MARS fit, while both of them share the same superb simplicity of interpretation.

3.7 Linear modeling

L_2Boosting turns out to be also very useful for linear models when there are many predictor variables. An attractive base procedure is component-



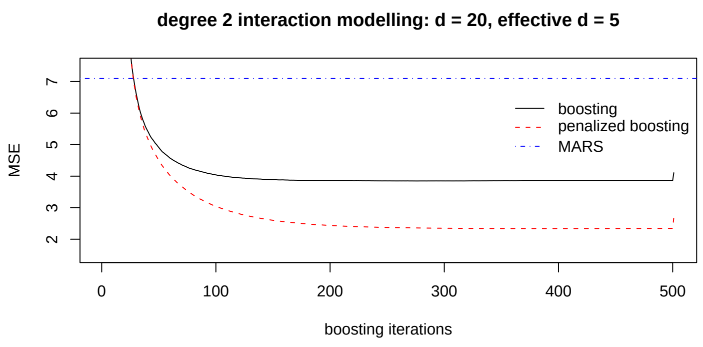


Fig. 7. Mean squared errors for L_2Boost with pairwise thin-plate splines (of two predictor variables, having $df = \text{trace}(\mathcal{S}_{j,k}) = 2.5$) (black), its penalized version (red) and MARS restricted to the (correct) second order interactions (blue). The point with abscissa x=501 for the boosting methods corresponds to the performance when estimating the number of iterations using (12). Based on simulated data from model (15) with $n = 50$.

wise linear least squares regression, using the one selected predictor variables which reduces residual sum of squares most.

This method does variable selection, since some of the predictors will never be picked during boosting iterations; and it assigns variable amount of degrees of freedom (or shrinkage), as discussed for additive models above. Recent theory shows that this method is consistent for very high-dimensional problems where the number of predictors $d = d_n$ is allowed to grow like $\exp(Cn)$ $(C > 0)$, but the true underlying regression coefficients are sparse in terms of their ℓ_1-norm, i.e. $\sup_n \|\beta\|_1 = \sup_n \sum_{j=1}^{d_n} |\beta_j| < \infty$, where β is the vector of regression coefficients ([Buh04]).

3.8 Boosting trees

The most popular base procedures for boosting, at least in the machine learning community, are trees. This may be adequate for classification, but when it comes to regression, or also estimation of conditional probabilities $\mathbb{P}[Y = 1|X = x]$ in classification, smoother base procedures often perform better if the underlying regression or probability curve is a smooth function of continuous predictor variables ([BY03]).

Even when using trees, the question remains about the size of the tree. A guiding principle is as follows: take the smallest trees, i.e. trees with the smallest number k of terminal nodes, such that the class of linear combinations of k-node trees is sufficiently rich for the phenomenon to be modeled; of course, there is also here a trade-off between sample size and the complexity of the function class.

For example, when taking stumps with $k = 2$, the set of linear combinations of stumps is dense in (or "yields" the) set of additive functions ([Bre04]). In [FHT00], this is demonstrated from a more practical point of view. When taking trees with three terminal nodes ($k = 3$), the set of linear combinations of 3-node trees yields all second-order interaction functions. Thus, when aiming for consistent estimation of the full regression (or conditional class-probability) function, we should choose trees with $k = d + 1$ terminal nodes (in practice only if the sample size is "sufficiently large" in relation to d), cf. [Bre04]).

Consistency of the AdaBoost algorithm is proved in [Jia04], for example when using trees having $d + 1$ terminal nodes. More refined results are given in [MMZ02], [ZY03] for modified boosting procedures with more general loss functions.

Interpretation

The main disadvantage from a statistical perspective is the lack of interpretation when boosting trees. This is in sharp contrast to boosting for linear, additive or interaction modeling. An approach to enhance interpretation is described in [Fri01].

3.9 Boosting and ℓ_1-penalized methods (Lasso)

Another method which does variable selection and variable amount of shrinkage is basis pursuit ([CDS99]) or Lasso ([Tib96]) which employs an ℓ_1-penalty for the coefficients in the log-likelihood.

Interestingly, in case of linear least squares regression with a "positive cone condition" on the design matrix, an approximate equivalence of (a version of) L_2Boosting and Lasso has been demonstrated in [EHJT03]. More precisely, the set of boosting solutions, when using an (infinitesimally) small step size (see section 3.3), over all the different boosting iterations, equals approximately the set of Lasso solutions when varying the ℓ_1-penalty parameter. Moreover, the approximate set of boosting solutions can be computed very efficiently by the so-called least angle regression algorithm ([EHJT03]).

It is not clear to what extent this approximate equivalence between boosting and Lasso carries over to more general design matrices in linear regression or to other problems than regression with other loss functions. But the approximate equivalence in the above mentioned special case may help to understand boosting from a different perspective.

In the machine learning community, there has been a substantial focus on consistent estimation in the convex hull of function classes (cf. [Bar03], [BJM03], [LV04]). For example, one may want to estimate a regression or probability function which can be written as

$$\sum_{k=1}^{\infty} w_k g_k(\cdot),\ w_k \geq 0,\ \sum_{k=1}^{\infty} w_k = 1,$$

where the $g_k(\cdot)$'s belong to a function class such as stumps or trees with a fixed number of terminal nodes. The quantity above is a convex combination of individual functions, in contrast to boosting which pursues linear combination of individual functions. By scaling, which is necessary in practice and theory (cf. [LV04]), one can actually look at this as a linear combination of functions whose coefficients satisfy $\sum_k w_k = \lambda$. This then represents an ℓ_1-constraint as in Lasso, a relation which we have already outlined above.

3.10 Other references

Boosting, or functional gradient descent, has also been proposed for other settings than regression or classification, including survival analysis ([Ben02]), ordinal response problems ([TH03]) and high-multivariate financial time series ([ABu03], [ABa03]).

Support vector machines (cf. [Vap98], [HTF01], [SS02]) have become very popular in classification due to their good performance in a variety of data sets, similarly as boosting methods for classification. A connection between boosting and support vector machines has been made in [RZH03], suggesting also a modification of support vector machines to more sparse solutions ([ZRHT03]).

Acknowledgments: I would like to thank Marcel Dettling for some constructive comments.

References

[ASS01] Allwein, E., Schapire, R., Singer, Y.: Reducing multiclass to binary: a unifying approach for margin classifiers. J. Machine Learning Research **1**, 113–141 (2001).

[AG97] Amit, Y., Geman, D.: Shape quantization and recognition with randomized trees. Neural Computation **9**, 1545–1588 (1997).

[ABa03] Audrino F., Barone-Adesi G.: A multivariate FGD technique to improve VaR computation in equity markets. To appear in Computational Management Science.

[ABu03] Audrino, F., Bühlmann, P.: Volatility estimation with functional gradient descent for very high-dimensional financial time series. J. Computational Finance **6**(3), 65–89 (2003).

[Bar03] Bartlett, P.L.: Prediction algorithms: complexity, concentration and convexity. In: Proceedings of the 13th IFAC Symposium on System Identification, pp. 1507–1517 (2003).

[BJM03] Bartlett, P.L., Jordan, M.I., McAuliffe, J.D.: Convexity, classification, and risk bounds. Technical Report 638, Dept. of Statistics, Univ. of Calif. (2003). Available from http://www.stat.berkeley.edu/tech-reports/index.html

[BK99] Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: bagging, boosting and variants. Machine Learning, **36**, 1545–1588 (1999).

[Ben02] Benner, A.: Application of "aggregated classifiers" in survival time studies. In: COMPSTAT 2002 - Proceedings in Computational Statistics - 15th Symposium held in Berlin (Eds. Härdle, W. and Rönz, B.), Physika Verlag, Heidelberg (2002).

[Bre96a] Breiman, L.: Bagging predictors. Machine Learning, **24**, 123–140 (1996)

[Bre96b] Breiman, L.: Out-of-bag estimation. Technical Report (1996). Available from ftp://ftp.stat.berkeley.edu/pub/users/breiman/

[Bre98] Breiman, L.: Arcing classifiers. Annals of Statistics **26**, 801–824 (1998).

[Bre99a] Breiman, L.: Prediction games & arcing algorithms. Neural Computation **11**, 1493–1517 (1999).

[Bre99b] Breiman, L.: Random Forests. Preprint. Available from http://stat-www.berkeley.edu/users/breiman/rf.html

[Bre04] Breiman, L.: Population theory for boosting ensembles. To appear in Annals of Statistics (2004).

[Buh03] Bühlmann, P.: Bagging, subagging and bragging for improving some prediction algorithms. In: Recent Advances and Trends in Nonparametric Statistics (Eds. Akritas, M.G., Politis, D.N.)), Elsevier (2003).

[Buh04] Bühlmann, P.: Boosting for high-dimensional linear models. In preparation (2004).

[BY00] Bühlmann, P., Yu, B: Discussion on Additive logistic regression: a statistical view of boosting (Auths. Friedman,J., Hastie, T., Tibshirani,R.) Annals of Statistics **28**, 377–386 (2000).

[BY02] Bühlmann, P., Yu, B.: Analyzing bagging. Annals of Statistics **30**, 927–961 (2002).

[BY03] Bühlmann, P., Yu, B.: Boosting with the L_2loss: regression and classification. J. American Statistical Association **98**, 324–339 (2003).

[BS02] Buja, A., Stuetzle, W.: Observations on bagging. Preprint (2002). Available from http://ljsavage.wharton.upenn.edu/~buja/

[Byl02] Bylander, T.:Estimating generalization error on two-class datasets using out-of-bag estimates. Machine Learning **48**, 287–297 (2002).

[CDS99] Chen, S.S., Donoho, D.L., Saunders, M.A.: Atomic decomposition by basis pursuit. SIAM J. Scientific Computing **20**(1), 33–61 (1999).

[CH03] Chen, S.X., Hall, P.: Effects of bagging and bias correction on estimators defined by estimating equations. Statistica Sinica **13**, 97–109 (2003).

[Det04] Dettling, M.: Bag-Boosting for tumor classification. In preparation (2004).

[DB03] Dettling, M., Bühlmann, P. Boosting for tumor classification with gene expression data. Bioinformatics **19**(9), 1061–1069 (2003).

[BD02] Borra, S., Di Ciaccio, A.: Improving nonparametric regression methods by bagging and boosting. Computational Statistics & Data Analysis **38**, 407–420 (2002).

[DF03] Dudoit, S., Fridlyand, J.: Bagging to improve the accuracy of a clustering procedure. Bioinformatics **19**(9), 1090–1099 (2003).

[ET98] Efron, B., Tibshirani, R.: The problem of regions. Annals of Statistics **26**, 1687–1718 (1998).

[EHJT03] Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. To appear in Annals of Statistics (2004).

[Fre95] Freund, Y. (1995): Boosting a weak learning algorithm by majority. Information and Computation **121**, 256–285 (1995).

[FS96] Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In Machine Learning: Proc. Thirteenth International Conference, pp. 148–156. Morgan Kauffman, San Francisco (1996).

[Fri91] Friedman, J.H.: Multivariate adaptive regression splines. Annals of Statistics **19**, 1–141 (with discussion) (1991).

[Fri01] Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of Statistics **29**, 1189–1232 (2001).

[FHT00] Friedman, J.H., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Annals of Statistics **28**, 337–407 (with discussion) (2000).

[HT90] Hastie, T.J., Tibshirani, R.J.: Generalized Additive Models. Chapman & Hall, London (1990).

[HTF01] Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Data Mining, Inference and Prediction. Springer, New York (2001).

[HL02] Hothorn, T., Lausen, B.: Bundling classifiers by bagging trees. Preprint (2002). Available from http://www.mathpreprints.com/math/Preprint/blausen/20021016/1/

[HST98] Hurvich, C.M., Simonoff, J.S., Tsai, C.-L.: Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion. J. Royal Statistical Society, Series B, **60**, 271–293 (1998).

[Jia04] Jiang, W.: process consistency for AdaBoost. To appear in Annals of Statistics (2004).

[LV04] Lugosi, G., Vayatis, N. On the Bayes-risk consistency of regularized boosting methods. To appear in Annals of Statistics (2004).

[MZ93] Mallat, S., Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. IEEE Transactions Signal Processing **41**, 3397–3415 (1993).

[MMZ02] Mannor, S., Meir, R., Zhang, T.: The consistency of greedy algorithms for classification. Proceedings COLT02, Vol. 2375 of LNAI, pp. 319–333, Sydney, Springer (2002).

[MBBF00] Mason, L., Baxter, J. Bartlett, P., Frean, M.: Functional gradient techniques for combining hypotheses. In: advances in Large Margin Classifiers (Eds. Smola, A.J., Bartlett, P.J., Schölkopf, B., Schuurmans, D.). MIT Press, Cambridge, MA (2000).

[PRW99] Politis, D.N., Romano, J.P., Wolf, M.: Subsampling. Springer, New York (1999).

[Rid02] Ridgeway, G.: Looking for lumps: boosting and bagging for density estimation. Computational Statistics and Data Analysis **38(4)**, 379–392 (2002).

[RZH03] Rosset, S., Zhu, J., Hastie, T. Margin maximizing loss functions. Accepted poster for NIPS (2003). Available from http://www-stat.stanford.edu/~hastie/pub.htm

[Sch90] Schapire, R.E.: The strength of weak learnability. Machine Learning **5**, 197–227 (1990).

[Sch02] Schapire, R.E.: The boosting approach to machine learning: an overview. In: MSRI Workshop on Nonlinear Estimation and Classification (Eds. Denison, D.D., Hansen, M.H., Holmes, C.C., Mallick, B., Yu, B). Springer, New York (2002).

[SS02] Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002).

[Tib96] Tibshirani, R.: Regression shrinkage and selection via the lasso. J. Royal Statistical Society, Series B, **58**, 267–288 (1996).

[Tuk77] Tukey, J.W.: Exploratory data analysis. Addison-Wesley, Reading, MA (1977).

[TH03] Tutz, G., Hechenbichler, K.: Aggregating Classifiers With Ordinal Response Structure, SFB 386 Discussion Paper No. 359 (2003). Available from http://www.stat.uni-muenchen.de/sfb386/

[Vap98] Vapnik, V.N.: Statistical Learning Theory. Wiley, New York (1998).

[Wah90] Wahba, G.: Spline Models for Observational Data. Society for Industrial and Applied Mathematics (1990).

[ZY03] Zhang, T., Yu, B.: Boosting with early stopping: convergence and consistency. Technical Report 635, Dept. of Statistics, Univ. of Calif., Berkeley (2003). Available from http://www.stat.berkeley.edu/users/binyu/publications.html

[ZRHT03] Zhu, J., Rosset, S., Hastie, T., Tibshirani, R.: 1-norm support vector machines. Accepted spotlight poster for NIPS (2003). Available from http://www-stat.stanford.edu/~hastie/pub.htm