Proceedings of the 3rd International Workshop
on Distributed Statistical Computing (DSC 2003)
March 20–22, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

Boosting Methods: Why they can be useful for High-Dimensional Data

Peter Bühlmann
Seminar für Statistik
ETH Zürich
CH-8092 Zürich
Switzerland

Abstract

We present an extended abstract about boosting. We describe first in section 1 (in a self-contained way) a generic functional gradient descent algorithm, which yields a general representation of boosting. Properties of boosting or functional gradient descent are then very briefly summarized in section 2.

1 Boosting

Originally, boosting has been proposed in the 90's (Freund and Schapire, 1996) as a multiple prediction and aggregation scheme for classification: a fitting method or estimator, called the base learner, is fitted multiple times on re-weighted data and the final boosting estimator is then constructed via a linear combination of such multiple predictions or estimates. In most empirical studies, the base learner is a regression or classification tree, and the improved performance through boosting has often been demonstrated to be impressive. Recently, some progress has been made in understanding how boosting algorithms work.

1.1 Functional Gradient Descent

Breiman (1999) has shown that boosting can be viewed as an optimization algorithm in function space. This important interpretation opened the door to understand

what boosting does and to bring boosting from classification to other settings such as regression, survival analysis or nonlinear time series. An interesting aspect is also that boosting is not an "intrinsic" multiple prediction scheme; the latter seems often somewhat "mysterious" and "ad-hoc". We will demonstrate that boosting methods can be successfully used for estimation in structured problems such as additive or interaction modelling.

We outline now a functional gradient descent method which yields a general representation of boosting algorithms. Consider the task of estimating a function $F : \mathbb{R}^p \to \mathbb{R}$, minimizing an expected loss

$$\mathbb{E}[\rho(Y, F(X))], \ \rho(\cdot,\cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+ \tag{1}$$

based on data (Y_i, X_i) $(i = 1, \ldots n)$; X_i denotes a p-dimensional predictor variable and Y_i a response. For expository simplicity, we consider the case with univariate response Y which can be continuous (regression problem) or discrete (classification problem). The loss function $\rho(\cdot,\cdot)$ is assumed to be smooth and convex in the second argument to ensure that the gradient method works well. The most prominent examples are:

$$\begin{aligned}
&\rho(y,f) = \exp(yf) \text{ with } y \in \{-1,1\}\text{: loss function for AdaBoost,}\\
&\rho(y,f) = \log_2(1+\exp(-2yf)) \text{ with } y \in \{-1,1\}\text{: loss function for LogitBoost,}\\
&\rho(y,f) = (y-f)^2/2 \text{ with } y \in \mathbb{R} \text{ or } \in \{-1,1\}\text{: loss function for } L_2\text{Boost.}
\end{aligned} \tag{2}$$

The LogitBoost loss function is equivalent to the log-likelihood function in a binomial model. The population minimizers of (1) are then (cf. Friedman et al., 2000)

$$\begin{aligned}
&F(x) = \frac{1}{2}\log\left(\frac{\mathbb{P}[Y=1|X=x]}{\mathbb{P}[Y=-1|X=x]}\right) \text{ for AdaBoost and LogitBoost loss,}\\
&F(x) = \mathbb{E}[Y|X=x] \text{ for } L_2\text{Boost loss.}
\end{aligned} \tag{3}$$

Estimation of such an $F(\cdot)$ from data can be done via a constrained minimization of the empirical risk

$$n^{-1}\sum_{i=1}^{n}\rho(Y_i, F(X_i)). \tag{4}$$

by applying functional gradient descent. The minimizer of (4), with respect to $F(\cdot)$, is imposed to satisfy a "smoothness" constraint in terms of an additive expansion of ("simple") base learners (fitted functions) which we denote by

$$h(\cdot, \hat{\gamma})_{U,X}, \ x \in \mathbb{R}^d, \tag{5}$$

where $\hat{\gamma}$ is a finite or infinite-dimensional parameter which is estimated from data $(U, X) = \{(U_i, X_i); i = 1, \ldots, n\}$, where the U_i's denote some generalized residuals or pseudo-response variables (see below). For example, the learner $h(\cdot, \hat{\gamma})_{U,X}$ could be a regression tree where $\hat{\gamma}$ describes the axis to be split, the split points and the

fitted values for every terminal node (the constants in the piecewise constant fitted function). How to fit $h(\cdot, \gamma)$ from data is part of the learner. For example, when using least squares

$$\hat{\gamma}_{U,X} = \text{argmin}_\gamma \sum_{i=1}^{n} (U_i - h(X_i; \gamma))^2,$$

or in the nonparametric context, we may think of penalized least squares or local least squares, localized by a kernel function.

The general description of functional gradient descent is as follows (see also Friedman, 2001).

Generic functional gradient descent

Step 1 (initialization). Given data $\{(Y_i, X_i); i = 1, \ldots, n\}$, fit a real-valued, (initial) learner

$$\hat{F}_0(x) = h(x; \hat{\gamma}_{Y,X}).$$

When using least squares, $\hat{\gamma}_{Y,X} = \text{argmin}_\gamma \sum_{i=1}^{n} (Y_i - h(X_i; \gamma))^2$. Set $m = 0$.

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

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

evaluated at the current $\hat{F}_m(\cdot)$. Then, fit the real-valued learner to the gradient vector

$$\hat{f}_{m+1}(x) = h(x, \hat{\gamma}_{U,X}).$$

When using least squares, $\hat{\gamma}_{U,X} = \text{argmin}_\gamma \sum_{i=1}^{n} (U_i - h(X_i; \gamma))^2$.

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

$$\hat{w}_{m+1} = \text{argmin}_w \sum_{i=1}^{n} \rho(Y_i, \hat{F}_m(X_i) + w_{m+1} \hat{f}_{m+1}(X_i)).$$

Update,

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

Step 4 (iteration). Increase m by one and repeat Steps 2 and 3.

The learner $h(x, \hat{\gamma}_{U,X})$ in Step 2 can be viewed as an estimate of $\mathbb{E}[U_i | X = x]$ and takes values in $\mathbb{R}$, even in case of a classification problem with Y_i in a finite set. We call $\hat{F}_m(\cdot)$ the real AdaBoost-, LogitBoost- or L_2Boost-estimate, according to the implementing loss function in (2). We point out that the real AdaBoost method as described here is slightly different from the popular AdaBoost as proposed by Freund and Schapire (1996) for classification; Friedman et al. (2000) give a more detailed description.

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.

L_2Boost algorithm

Step 1 (initialization). As in Step 1 of generic functional gradient descent, using a least squares fit (maybe including some regularization).

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

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

Step 3 (iteration). Increase iteration index m by one and repeat Step 2.

L_2Boosting is thus nothing else than repeated least squares fitting of residuals (cf. Friedman, 2001). With $m = 1$ (one boosting step), it has already been proposed by Tukey (1977) under the name "twicing".

For a continuous $Y \in \mathbb{R}$, a regression estimate for $\mathbb{E}[Y|X = x]$ is directly given by the L_2Boost-estimate $\hat{F}_m(\cdot)$. For a two-class problem with $Y \in \{-1, 1\}$, a classifier under equal misclassification costs is given by

$$sign(\hat{F}_m(x))$$

according to the population minimizers in (3).

1.2 Early stopping and resistance to overfitting

In the early area of boosting, it was "commonly believed" that boosting will never overfit as the boosting or functional gradient descent iteration continues to go on! By now, it is known that this is not true. We should stop boosting, or the functional gradient descent algorithm, before numerical convergence (in cases it would converge numerically). This is a way to regularize and to avoid overfitting.

In some cases, it can be proved that boosting converges numerically to the fully saturated model, i.e. $\hat{F}_\infty(X_i) = Y_i$ for all $i = 1, \ldots, n$, fitting the data exactly. But typically, this convergence becomes very slow and it can also be proved for some learners (e.g. smoothing spline base learners), that the bias of $\hat{F}_m(\cdot)$ converges exponentially fast as $m \to \infty$ while the variance increases by geometrically diminishing magnitudes only. This is in sharp contrast to the usual linear increase in variance when fitting additional terms in a basis expansion or a regression model. It implies that overfitting comes in very slowly, particularly when the base learner is weak (low variance). This property turns out to be often beneficial for tuning boosting methods: due to the resistance against overfitting, tuning boosting via stopping (choosing the number of boosting iterations) is typically easy. For example, we can often reliably estimate the number of boosting iterations by cross-validation.

2 Properties of boosting: some references

We touch here very briefly some selected aspects and properties about boosting or functional gradient descent.

2.1 Boosting is mainly useful for data with high-dimensional predictors

It is empirically illustrated in Bühlmann and Yu (2003) that boosting has mainly an advantage for data with high-dimensional predictors. In more classical settings with reasonable ratio of predictor dimension p and sample size n, flexible modern regression or classification methods perform as well as boosting.

Consider the Friedman #1 model as an example:

$$\begin{aligned} X &= (X_1, \ldots, X_p) \ \sim \text{Unif.}([0,1]^p) \\ Y &= 10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 \ + \varepsilon, \ \varepsilon \sim \mathcal{N}(0,1). \end{aligned} \quad (6)$$

Sample size is chosen as $n = 50$ and $p \in \{10, 20\}$; particularly for $p = 20$, we have have high-dimensional predictors relative to sample size. Figure 1 displays the

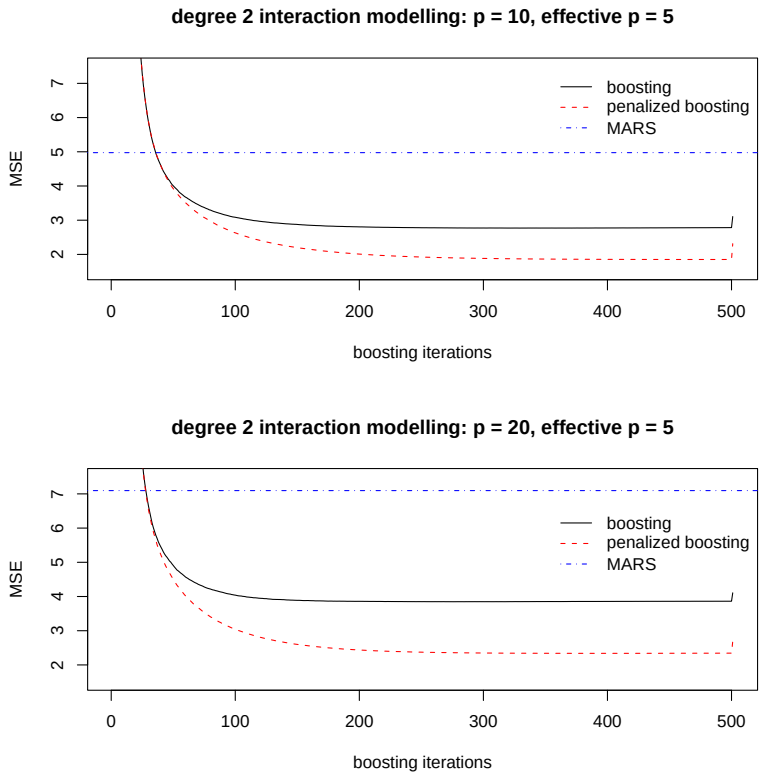

Figure 1: Mean squared error for L_2Boosting using componentwise thin plate splines, a penalized version of L_2Boosting, and MARS in the Friedman #1 model (6) with sample size $n = 50$. Top: $p = 10$. Bottom: $p = 20$. The performance with AIC-stopped boosting iterations is indicated at abscissa $x = 501$.

performance of L_2Boosting and of a better version based on penalization (see section 2.1.1), and we also compare with MARS (restricted to second order interactions). The base learner in L_2Boosting is a componentwise thin plate spline which chooses the pair of predictor variables (X_{j_1}, X_{j_2}) $(j_1, j_2 \in \{1, \ldots, p\},\ j_1 \neq j_2)$ such that smoothing against the chosen two-dimensional predictors yields the smallest residual sum of squares among all possible pairs (while keeping the degrees of freedom fixed (low) for all pairs (X_{j_1}, X_{j_2})).

2.1.1 Penalized L_2Boosting

The L_2Boost algorithm is greedy, aiming to maximally reduce the residual sum of squares in every boosting iteration with a given base learner. The analogue in the population case is a (theoretical) algorithm which would reduce the mean squared error (MSE) most in every iteration. Although the MSE is unknown, we can estimate it via a penalized residual sum of squares: penalized L_2Boost then modifies Step 2 of the L_2Boost algorithm by fitting the residual vector with the base learner such that

$$\text{residual sum of squares of boosting } + \text{AIC-penalty}$$

becomes minimal. For example in additive model fitting with a componentwise smoothing spline as a base learner, it can be argued that the increase of the AIC penalty in an additional boosting iteration for fitting a particular predictor variable, say j^*, is small if variable j^* has already been selected many times in previous boosting iterations, and vice versa. Thus, penalized boosting tends to solutions with fewer selected predictor variables.

The AIC-penalty (or corrected AIC) can be computed from the degrees of freedom of L_2Boosting (the trace of the L_2Boost hat matrix if the base learner is linear). It is worth mentioning that penalized L_2Boost is not an L_2 boosting algorithm anymore since the base learner depends on the residual sum of squares of the current boosting solution, and not just on the current residuals.

2.2 Boosting does variable selection and assigns variable amount of degrees of freedom

A possible explanation why boosting performs well in the presence of high-dimensional predictors is that it does variable selection (assuming the base learner does variable selection) *and* it assigns variable amount of degrees of freedom to the selected predictor variables or terms. So far, theory is lacking whether this kind of allocation for different amount of degrees of freedom is optimal in any sense. It is worth mentioning that with more classical methods and in the presence of high-dimensional predictors, forward variable or term selections is still feasible but the problem of assigning varying amount of complexity to the selected predictor variables or terms becomes computationally very difficult or intractable.

We show in Figure 2 the evolution of selecting predictor variables and degrees of freedom in additive model fitting using boosting. The true underlying model is

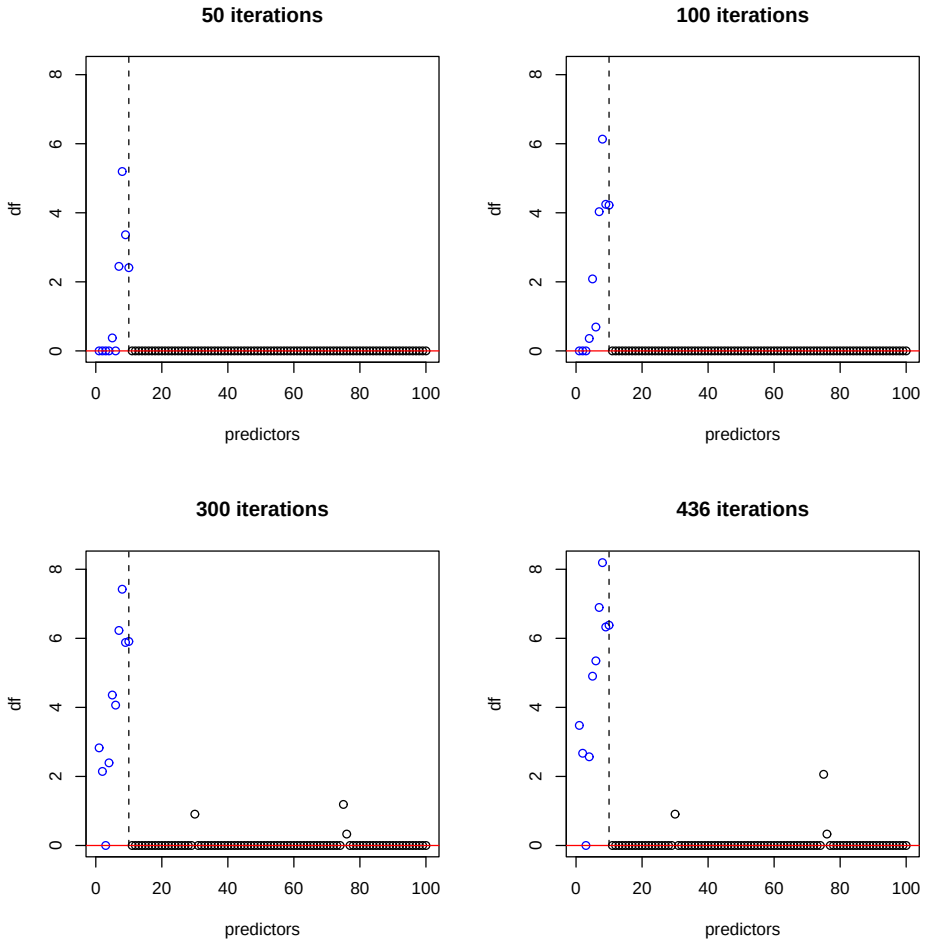

Figure 2: Degrees of freedom in additive model fitting for all 100 predictor variables which were assigned during the process of L_2Boosting with componentwise smoothing splines; the first ten predictor variables (separated by the dashed vertical line) are effective. The data is generated from model (7) having sample size $n = 200$. The figure on the lower right corner corresponds to the optimal number of boosting iterations with respect to the corrected AIC statistic.

$$X = (X_1, \ldots, X_{100}) \ \sim \text{Unif.}([0,1]^{100})$$
$$Y = \sum_{j=1}^{10} f_j(X_j) + \varepsilon, \ \varepsilon \ \sim \mathcal{N}(0, 0.5), \tag{7}$$

with nonparametric functions $f_j(\cdot)$ having varying curve complexities. Sample size is chosen as $n = 200$. The individual signal to noise ratios (SNR) $\text{Var}(f_j(X_j))/\text{Var}(\varepsilon)$ are as follows:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
SNR	0.17	0.04	0.01	0.21	0.65	0.24	3.79	14.19	7.00	2.00

Thus, the function f_8 is most complex, followed by f_9 and f_7. We use L_2Boosting with componentwise cubic smoothing splines which chooses the predictor variable X_j ($j \in \{1, \ldots, p\}$) such that smoothing against the chosen one-dimensional predictor yields the smallest residual sum of squares among all possible predictors (while keeping the degrees of freedom fixed (low) for all predictors X_j). This yields an additive model fit, very much different from backfitting, since a linear combination of estimated functions of selected predictors can be represented as an additive function in the original predictor variables.

It is interesting to see from Figure 2 that after the AIC-estimated 436 boosting iterations, only 3 non-effective predictors have been selected and were assigned relatively low degrees of freedom. Also, most degrees of freedom were assigned to the 8th predictor variable which is most important (f_8 has highest signal to noise ratio); and the third predictor was never selected because the function f_3 has only very low signal to noise ratio.

2.3 Boosting is asymptotically optimal in the toy problem of 1-dimensional curve estimation

It is shown in Bühlmann and Yu (2003) that L_2Boosting with smoothing splines for a one-dimensional predictor is asymptotically optimal, i.e. it achieves the minimax mean squared error rate. As an interesting addition, L_2Boosting also adapts to higher order smoothness of the true underlying function. For example, using cubic smoothing spline base learners, the boosted cubic smoothing spline can achieve a faster MSE rate than $n^{-4/5}$ if the underlying function is smooth enough. Note that this faster rate cannot be achieved by an ordinary, non-boosted cubic smoothing spline (we would need to take a higher order spline).

2.4 Boosting yields consistent function approximations

There are now several results about consistent function approximation by boosting algorithms where the function has any fixed finite-dimensional predictor space as its domain. Most of these works consider regression or classification trees as base learners. Some references include Jiang (2000), Lugosi and Vayatis (2001), Zhang (2001), Mannor et al. (2002), Bühlmann (2002) and Zhang and Yu (2003). The argument in Bühlmann (2002) can be carried out further to the case with fast

growing number of predictors, or even infinite-dimensional predictors, where the underlying true function is "sparse": this may give a clue why boosting can cope well with very high-dimensional predictors as long as the true underlying function is within reasonable complexity (Bühlmann and Yu; in progress).

References

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

[2] Bühlmann, P. (2002). Consistency for L_2Boosting and matching pursuit with trees and tree-type basis functions. Preprint, available from `http://stat.ethz.ch/~buhlmann/bibliog.html`

[3] Bühlmann, P. and Yu, B. (2003). Boosting with the L_2 loss: regression and classification. J. Amer. Statist. Assoc. **98**, 324–339.

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

[5] Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. Ann. Statist. **29**, 1189–1232.

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

[7] Jiang, W. (2000). Process consistency for AdaBoost. To appear in Ann. Statist.

[8] Lugosi, G. and Vayatis, N. (2001). On the Bayes-risk consistency of boosting methods. To appear in Ann. Statist.

[9] Mannor, S., Meir, R. and Zhang, T. (2002). The consistency of greedy algorithms for classification. Proc. Fifteenth Annual Conference on Computational Learning Theory 2002 (COLT).

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

[11] Zhang, T. (2001). Statistical behavior and consistency of classification methods based on convex risk minimization. To appear in Ann. Statist.

[12] Zhang, T. and Yu, B. (2003). Boosting with early stopping: convergence and consistency. Tech. Report 635, Department of Statistics, UC Berkeley. available from `http://www.stat.berkeley.edu/~binyu/publications.html`