Robustified L_2Boosting

Roman Werner Lutz, Markus Kalisch *and Peter Bühlmann

Seminar für Statistik, ETH Zurich, 8092 Zurich, Switzerland

Abstract

Five robustifications of L_2Boosting for linear regression with various robustness properties are considered. The first two use the Huber loss as implementing loss function for boosting and the second two use robust simple linear regression for the fitting in L_2Boosting (i.e. robust base learners). Both concepts can be applied with or without down-weighting of leverage points. Our last method uses robust correlation estimates and appears to be most robust. Crucial advantages of all methods are that they don't compute covariance matrices of all covariates and that they don't have to identify multivariate leverage points. When there are no outliers, the robust methods are only slightly worse than L_2Boosting. In the contaminated case though, the robust methods outperform L_2Boosting by a large margin. Some of the robustifications are also computationally highly efficient and therefore well suited for truly high dimensional problems.

Key words: L_2Boosting, Linear Regression, Robustness

1 Introduction

Freund and Schapire's AdaBoost algorithm for classification (Freund and Schapire, 1996) has attracted much attention in the machine learning community and related fields, mainly because of its good empirical performance. Some boosting algorithms for regression were also proposed, but the first practical algorithm was not possible until Breiman (1999) showed, that boosting can be viewed as a functional gradient descent algorithm. Friedman (2001) then proposed LS_Boost (least squares boosting, we will call it L_2Boosting) and also more robust boosting methods in conjunction with regression trees.

* Corresponding author. Address: Leonhardstr. 27, 8057 Zurich, Switzerland
Email address: kalisch@stat.math.ethz.ch (Markus Kalisch).

Boosting with the L_2-loss (L_2Boosting) and componentwise linear fitting was worked out in detail in Bühlmann (2006). It is essentially the same as Mallat and Zhang's (1993) matching pursuit algorithm in signal processing and very similar to stagewise linear model fitting (see for example Efron et al. (2004)). Boosting is then not just a black box tool, but fits sound linear models. It does variable selection and coefficient shrinkage and for high dimensional problems, it is clearly superior to the classical model selection methods (see Bühlmann, 2006).

The usage of the L_2-loss is dangerous when there are outliers. Friedman (2001) discussed some robust boosting algorithms with regression trees. In this paper we develop some robust boosting algorithms for linear models by using either robust implementing loss functions in boosting or robust estimators as base (weak) learners. They all do variable selection and estimation of regression coefficients. Some of our methods are also well suited for very high dimensional problems with many covariates and/or large sample size. Besides more classical work in robust fitting and variable selection for linear models (Ronchetti and Staudte (1994), Ronchetti et al. (1997), Morgenthaler et al. (2003), McCann and Welsch (2007)), our approaches are closest to "robust LARS" (Van Aelst et al., 2004): L_2-boosting with infinitesimal small ν is equal to forward stagewise, which is again very similar to LARS (see Efron et al., 2004). However, the concepts of robust loss functions and robust base learners are much more general.

2 L_2Boosting with componentwise linear least squares

We consider the linear model $\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\epsilon}$ with $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p) \in \mathbb{R}^{n \times p}$ and will use boosting methods for fitting it. For a boosting algorithm we need a loss function $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+$, that measures how close a fitted value $\hat{F}(x_i)$ comes to the observation y_i and a base learner (simple fitting method), that yields a function estimate $\hat{f} : \mathbb{R}^p \to \mathbb{R}$. L_2Boosting uses the L_2-loss $L(y, F) = (y - F)^2/2$ and as base learner we take componentwise linear least squares, which works as follows: a response $\mathbf{r} \in \mathbb{R}^n$ with $\bar{\mathbf{r}} = 0$ is fitted against $\mathbf{x}_1, \ldots, \mathbf{x}_p$:

Componentwise linear least squares learner

$$\hat{f}(x) = \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \ x \in \mathbb{R}^p$$
$$\hat{\beta}_j = \frac{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T \mathbf{r}}{(\mathbf{x}_j - \bar{\mathbf{x}}_j)^T (\mathbf{x}_j - \bar{\mathbf{x}}_j)}, \quad \hat{\alpha}_j = -\hat{\beta}_j \bar{\mathbf{x}}_j, \quad 1 \le j \le p,$$
$$\hat{s} = \operatorname*{arg\,min}_{1 \le j \le p} ||\mathbf{r} - \hat{\alpha}_j - \hat{\beta}_j \mathbf{x}_j||^2 = \operatorname*{arg\,max}_{1 \le j \le p} |\hat{\beta}_j| \cdot \mathrm{sd}(\mathbf{x}_j) = \operatorname*{arg\,max}_{1 \le j \le p} |\operatorname{Cor}(\mathbf{r}, \mathbf{x}_j)|.$$

In words: we fit a simple linear regression with one selected covariate. The selected covariate is the one which gives the smallest residual sum of squares. This is equivalent to the variable that gives the "largest contribution to the fit" or has the highest absolute correlation with the response $\mathbf{r}$. The requirement $\bar{\mathbf{r}} = 0$ is without loss of generality for boosting since we always center the response variable before, see the algorithm below. The learner can be simplified if all covariates are centered (mean subtracted). Then we can fit simple linear regressions through the origin.

A boosting algorithm constructs iteratively a function $\hat{F} : \mathbb{R}^p \to \mathbb{R}$ by considering the empirical risk $n^{-1} \sum_{i=1}^n L(y_i, F(x_i)),\ x_i \in \mathbb{R}^p$ and pursuing iterative approximate steepest descent in function space. This means that in each iteration, the negative gradient of the loss function is fitted by the base learner. L_2Boosting is especially simple, because the negative gradient becomes the current residual vector and the algorithm amounts to iteratively fitting of residuals:

L_2Boosting with componentwise linear least squares

(1) Initialize $\hat{F}^{(0)} \equiv \arg\min_{a \in \mathbb{R}} \sum_{i=1}^n L(y_i, a) \equiv \bar{\mathbf{y}}$. Set $m = 0$.

(2) Increase m by 1. Compute the negative gradient (also called pseudo response) that is the current residual vector

$$r_i = -\frac{\partial}{\partial F} L(y, F)|_{F=\hat{F}^{(m-1)}(x_i)} = y_i - \hat{F}^{(m-1)}(x_i),\ i = 1, \ldots, n.$$

(3) Fit the residual vector $(r_1, \ldots, r_n)$ to $\mathbf{x}_1, \ldots, \mathbf{x}_p$ by the componentwise linear least squares base procedure

$$(x_i, r_i)_{i=1}^n \longrightarrow \hat{f}^{(m)}(\cdot).$$

(4) Update $\hat{F}^{(m)}(\cdot) = \hat{F}^{(m-1)}(\cdot) + \nu \cdot \hat{f}^{(m)}(\cdot)$, where $0 < \nu \leq 1$ is a step length factor.

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

The number of iterations $m = m_{stop}$ is usually estimated using a validation set or with cross validation. The step-length factor ν is also called shrinkage factor and is typically less crucial than m_{stop}. The natural value is 1, but Friedman (2001) was first to propose smaller values, and he argued empirically that this is often a better choice. In fact, he demonstrated that small values of ν are good and that the sensitivity of the boosting procedure is low with respect to a whole range of small values of ν. We will always use $\nu = 0.3$. Since the base learner yields a linear model fit in one covariate and because of the linear up-date in step 4, L_2Boosting with componentwise linear least squares yields a linear model fit (with estimated coefficient vector $\hat{\beta}^{(m)}$). Since least squares fitting is used, the method is not robust to outliers.

3 Robustifications

There are several ways to robustify L_2Boosting with componentwise linear least squares as described next. Whenever we need a robust location estimate we will use the Huber estimator with MAD scale (see Huber (1964), Huber (1981) and Hampel et al. (1986)). The Huber Ψ-function is given by

$$\Psi_c(x) = \min\{c, \max\{x, -c\}\} = x \cdot \min\{1, \frac{c}{|x|}\}.$$

As robust scale estimator we use the Q_n estimator of Rousseeuw and Croux (1993). It is defined as

$$Q_n(x_1, \ldots, x_n) = 2.2219 \cdot \{|x_i - x_j|; i < j\}_{(k)},$$

where $k = \binom{\lfloor n/2 \rfloor + 1}{2}$. That is, we take the k-th order statistic of the $\binom{n}{2}$ inter-point distances. The Q_n estimator has a breakdown point of 50% and an efficiency of 82% at the Gaussian distribution (Rousseeuw and Croux, 1993).

3.1 Boosting with a robust implementing loss function

The easiest robustification is to use a robust loss function, e.g. the Huber loss function (the derivation yields the Huber Ψ-function):

$$L_c(y, F) = \begin{cases} (y - F)^2/2, & |y - F| \le c, \\ c * (|y - F| - c/2), & |y - F| > c. \end{cases}$$

The parameter c should be chosen in dependence of the scale of $y - F$. We choose it adaptively in each iteration as $c = 1.345 \cdot \text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i), i = 1, \ldots, n\})$ as proposed in Friedman (2001). The negative gradient in step 2 of the boosting algorithm then becomes the huberized residual vector. As learner we can take componentwise linear least squares as described above. This means we look in each iteration for the covariate that best fits the huberized residuals (the criterion is the huberized residual sum of squares). We found that it is better to also estimate an intercept in each iteration than robust centering the covariates and estimate the intercept only at the beginning. We shall call this version *RobLoss* boosting.

Remark 1 *If we want to exactly apply the Gradient Boost algorithm of Friedman (2001) we have to do an additional line search between step 3 and 4. This*

means each $\hat{f}^{(m)}$ must be multiplied by a constant to minimize the L_c-loss. Since we are going to use shrinkage $\nu = 0.3$ it is not important to know the optimal (greedy) step size exactly and we can omit the additional line search.

It is obvious that *RobLoss* boosting is only robust in "Y-direction" but not in "X-direction". We incorporate "X-direction-robustness" in the base procedure when fitting the negative gradient in step 3 (alternatively, one could also bound the loss function). The idea is to down-weight the leverage points and to use weighted least squares for fitting (a Mallows type estimator). Since every covariate is fitted alone, the weight of an observation is solely determined by the value of the one covariate. Therefore, the same observation can have different weights for the p candidate fits with the p covariates in one iteration, according to its outlyingness of the corresponding coordinate. For the weights we use ($w_{ij}^{position}$ is the weight of observation i when fitting the j-th covariate)

$$w_{ij}^{position} = \min\{1, \frac{1.345}{|(x_{ij} - \text{Huber}(\mathbf{x}_j))/\,\text{MAD}(\mathbf{x}_j)|}\}.$$

We can go one step further and down-weight only the leverage points that have also a large residual (a Schweppe type estimator). The weights we use in iteration m are

$$w_{ij} = \frac{\Psi_{1.345 \cdot w_{ij}^{position}}\left((y_i - \hat{F}^{(m-1)}(x_i))/\,\text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i), i = 1, \ldots, n\})\right)}{\Psi_{1.345}\left((y_i - \hat{F}^{(m-1)}(x_i))/\,\text{MAD}(\{y_i - \hat{F}^{(m-1)}(x_i), i = 1, \ldots, n\})\right)}.$$

So far, we only discussed how to fit the pseudo response to a covariate, but the selection of the "best" covariate is equally important. It seems quite natural to select the covariate that gives the smallest weighted residual sum of squares. But since the p simple linear fits in each iteration use different weights for the same observation, this can lead to bad choices. It is better to use the estimated $\hat{\beta}_j$'s and to select the variable that has highest $|\hat{\beta}_j| \cdot Q_n(\mathbf{x}_j)$: note that this is of the form as used in the classical componentwise linear least squares base procedure in section 2. Roughly speaking we choose the covariate that contributes the most to the fit. We shall call this version *RobLossW* boosting. Here is the formal description of the learner (w_{ij} as described above with r_i instead of $y_i - \hat{F}^{(m-1)}(x_i)$):

Componentwise linear weighted least squares learner

$$\begin{aligned}\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\ (\hat{\alpha}_j, \hat{\beta}_j) &= \arg\min_{\alpha,\beta} \sum_{i=1}^{n} w_{ij}(r_i - \hat{\alpha}_j - \hat{\beta}_j x_{ij})^2,\end{aligned}$$

$$\hat{s} = \underset{1 \le j \le p}{\arg\max} |\hat{\beta}_j| \cdot Q_n(\mathbf{x}_j).$$

3.2 Boosting with robust regression learner

Here we use the idea of iteratively fitting of residuals. Instead of fitting the ordinary residuals by least squares, we use a robust linear regression:

Componentwise robust linear regression learner

$$\begin{aligned}
\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\
(\hat{\alpha}_j, \hat{\beta}_j) &= \text{robust linear fit of } \mathbf{r} \text{ against } \mathbf{x}_j, \\
\hat{s} &= \underset{1 \le j \le p}{\arg\max} |\hat{\beta}_j| \cdot \mathit{scale}(\mathbf{x}_j).
\end{aligned}$$

In each iteration of the boosting algorithm we calculate a robust linear regression with each covariate alone (and an intercept). This needs more computation than the *RobLoss* methods, since the robust fits are usually calculated iteratively itself (the *RobLoss* methods do in some sense only the first iteration of the iteration). As criterion for the variable selection we use again $\arg\max_j |\hat{\beta}_j| \cdot \text{scale}(\mathbf{x}_j)$, where $\text{scale}(\mathbf{x}_j)$ is a scale estimate that will be specified below. This is much better than using a robust estimation of residual standard error.

For the robust linear fit of the base procedure we use two different types: M-regression with Huber's Ψ-function (and rescaled MAD of the residuals) and a Schweppe type bounded influence (BI) regression (see for example Hampel et al. (1986)) with Huber's Ψ-function and position weights as described in section 3.1. We chose these types of robust regression to have a direct comparison to the *RobLoss* methods. One could even use MM-regression, but this would be computationally very expensive. The proposed algorithms will be called *RobRegM* boosting and *RobRegBI* boosting. For the former method, that is robust in "Y-direction" but not in "X-direction", we use the standard deviation as scale estimate for the variable selection and for the latter, that is robust in "Y- and X-direction", we use the Q_n estimator.

We expect that *RobLoss* and *RobRegM* boosting perform similar and likewise for RobLossW and *RobRegBI* boosting. The former methods first huberize the residuals and then use (weighted) least squares and the latter methods use robust methods with the same huberization and weighting. As already mentioned, the *RobLoss* methods do in some sense the first iteration of the robust fitting of the *RobReg* methods. The great advantage of the former methods is that they are much faster. Thus, the latter methods must achieve better performance to be worthwhile.

3.3 Boosting with robust correlation learner

It is also possible to use robust correlation estimators to construct a base procedure. The idea is the following: in each iteration, the covariate with the highest robust correlation with the residuals is chosen, see also the selection in classical componentwise least squares described in section 2. We shall call this version *RobCor* boosting:

Robust correlation learner

$$\begin{aligned}\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\ \hat{\beta}_j &= \mathrm{RobCor}(\mathbf{x}_j, \mathbf{r}) \cdot Q_n(\mathbf{r})/Q_n(\mathbf{x}_j), \quad \hat{\alpha}_j = \mathrm{Huber}(\mathbf{r}) - \hat{\beta}_j \cdot \mathrm{Huber}(\mathbf{x}_j), \\ \hat{s} &= \operatorname*{arg\,max}_{1 \le j \le p} |\,\mathrm{RobCor}(\mathbf{x}_j, \mathbf{r})|.\end{aligned}$$

Recall that it is not important to have very accurate $\hat{\beta}_j$'s because we use shrinkage $\nu = 0.3$. As robust correlation estimate we use a proposal from Huber (1981) with the Q_n estimator as module:

$$\mathrm{RobCor}(\mathbf{x}, \mathbf{y}) = \frac{Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} + \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2 - Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} - \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2}{Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} + \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2 + Q_n\left(\frac{\mathbf{x}}{Q_n(\mathbf{x})} - \frac{\mathbf{y}}{Q_n(\mathbf{y})}\right)^2}.$$

3.4 Stopping the boosting iteration

To stop the boosting iteration, we propose to use cross validation or a separate validation set. After each iteration we use the actual model to predict on the validation sample and we measure the quality of the fit. For L_2Boosting we use the mean squared prediction error on the validation set and for the robust methods, we use a robust measure of prediction error. Ronchetti et al. (1997) propose to use the Huber loss of the errors of the validation set. We found that using the Q_n-estimator of the errors on the validation set gives better results and therefore, we tune all robust boosting methods with the Q_n-estimator.

3.5 Properties of the robust boosting methods

An unaesthetic property of *RobCor* boosting with $\nu = 1$ is that the same covariate can be chosen consecutively. This is because the robust correlation between the residuals and a covariate is usually not equal to zero after fitting the covariate in the iteration before. This is in contrast to L_2Boosting and the *RobReg* methods, where, with $\nu = 1$, we cannot improve the fit by selecting

and fitting the same covariate as in the iteration before. For the *RobLoss* methods it can also happen that they choose the same covariate consecutively (even with $\nu = 1$). Note however that for the advocated proposal with $\nu < 1$, the differences in qualitative behavior disappear.

RobCor boosting empirically shows the following behavior: after running for a large number of iterations, it always selects the same variable and gets stuck. Then the estimated coefficients are all of approximately equal size and successive coefficients are of opposite sign. This means that *RobCor* boosting estimates the coefficient of the selected variable too high and in the next iteration it undoes the previous step. The good thing is that this doesn't happen until over-fitting occurs, so we would stop the iteration before anyway. We can also delay getting stuck by choosing a smaller ν.

Regarding the break down point we can state the following simple proposition:

Proposition 1 *The boosting method inherits the breakdown point of the base procedure.*

Since we are performing only a finite number of iterations, the whole boosting algorithm can only break down whenever the base learner breaks down in one iteration. *RobCor* boosting has therefore a breakdown point of 0.5.

4 Simulation study

In this section we compare the different boosting methods on simulated data sets from a linear model $\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\epsilon}$ as described at the beginning of section 2 and hence, all our results are only for linear regression. In this simulation study we mainly limit ourselves to giving an overview of the behavior of the different robust boosting methods. To this end, we will analyze the methods in a rather low and a rather high-dimensional setting; one could analyze the methods over a wider range of parameter settings, but this is beyond the scope of this paper. Using real data, we will also show the computational feasibility for very high-dimensional settings.

4.1 Design

The sample size of the training set (and also the validation set, used to stop the iteration) is $n = 100$ and the number of covariates is $p = 10$ in the first example and $p = 100$ in the second example (there is no test set needed, since the true parameters of the model are known). The true coefficient vector β is an arbitrary permutation of $(8, 7, 6, 5, 4, 0, 0, 0, 0, 0)^T$ for

$p = 10$ and $(18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 0, \ldots, 0)^T$ for $p = 100$. Thus, we have $p_{eff} = 5$ effective and $p_{noise} = 5$ noise variables for $p = 10$ or $p_{eff} = 10$ and $p_{noise} = 90$ for $p = 100$, respectively. We use two design matrices for the covariates and four error distributions. For the first design (normal design), the covariates are generated according to a multivariate normal distribution with mean zero and $\mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) = \Sigma_{ij} = 0.5^{|i-j|}$. The second design (leverage design) is the same, except that 10% of the data-points are shifted by $4\sqrt{p_{eff}}$ in the direction $(1, 1, \ldots, 1)^T$. This is done after the true y-values have been determined. Therefore we have bad leverage points. The error distributions are defined as follows:

e1 Standard Normal $\mathcal{N}(0, 1^2)$
e2 90% $\mathcal{N}(0, 1^2)$ and 10% $\mathcal{N}(0, 5^2)$
e3 90% $\mathcal{N}(0, 1^2)$ and 10% Slash (i.e. $\mathcal{N}(0, 1^2)/\mathcal{U}(0, 1)$)
e4 90% $\mathcal{N}(0, 1^2)$ and 10% Cauchy (location zero and scale five)

The errors are multiplied by a constant to give a signal-to-noise ratio of 4 in the first example and 9 in the second example for normal errors. Each setting is replicated 100 times.

4.2 Performance measure

Our main performance measure is the mean squared prediction error, which can be calculated as

$$\widehat{intercept}^2 + (\hat{\beta} - \beta)^T \mathbf{\Sigma} (\hat{\beta} - \beta).$$

where $\hat{\beta}_0$ is the estimated intercept (the true intercept being zero), $\hat{\beta}$ is the estimated parameter vector, β is the true parameter vector (as defined in section 4.1) and $\mathbf{\Sigma}$ is the covariance matrix of the covariates (as defined in section 4.1). We also assess the variable selection performance by using ROC curves. Moreover, we give the results of a small runtime study.

4.3 Results for $p = 10$

Table 1 gives the average over 100 replicates of the mean squared prediction error when stopping with the validation set. The results are as expected. For the normal design with error $e1$, L_2Boosting performs significantly best. The robust methods are only slightly worse, except perhaps *RobCor* boosting which is significantly worse than the other robust methods. For the normal design with error $e2$, $e3$ and $e4$, L_2Boosting performs much worse than the robust methods, and *RobCor* boosting is still worse than the other robust methods.

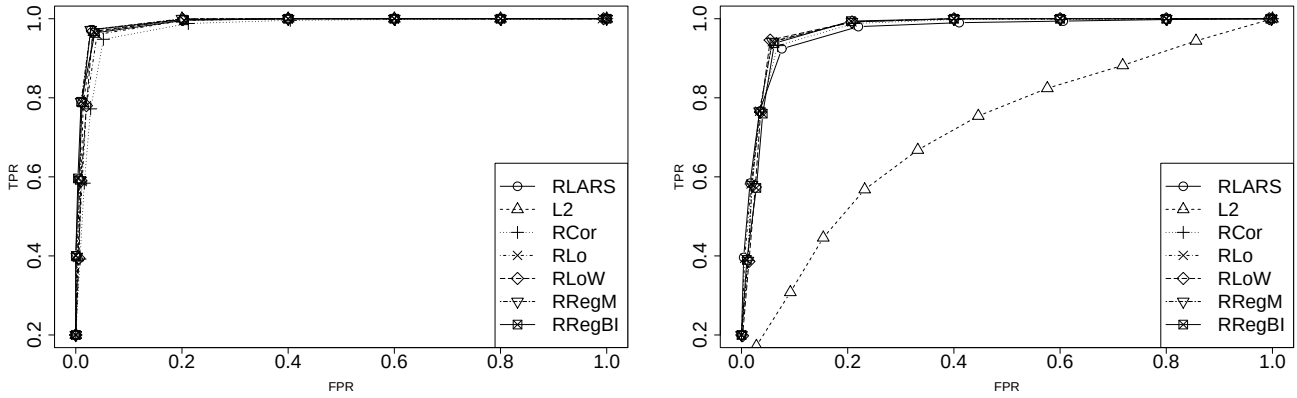
Design	Method	$e1$	$e2$	$e3$	$e4$
Normal	L_2	7.7 (0.4)	25.5 (1.4)	2600 (2000)	1874 (1038)
	RobLoss	8.9 (0.5)	11.4 (0.6)	9.9 (0.5)	12.5 (0.7)
	RobRegM	8.8 (0.5)	11.4 (0.6)	9.7 (0.5)	12.5 (0.7)
	RobLossW	9.1 (0.5)	11.5 (0.6)	10.5 (0.5)	12.3 (0.7)
	RobRegBI	9.1 (0.5)	11.8 (0.6)	10.4 (0.5)	12.1 (0.7)
	RobCor	11.3 (0.7)	13.8 (0.8)	12.2 (0.7)	13.1 (0.7)
Leverage	L_2	242 (2)	258 (2)	2895 (2136)	1894 (994)
	RobLoss	245 (2)	251 (2)	247 (2)	259 (2)
	RobRegM	245 (2)	251 (2)	248 (2)	258 (2)
	RobLossW	101 (5)	180 (8)	147 (8)	199 (9)
	RobRegBI	99 (5)	162 (7)	132 (6)	168 (7)
	RobCor	16 (1)	22 (1)	18 (1)	21 (1)

Table 1
Mean squared prediction error of the different boosting methods when stopping with a validation set, averaged over 100 replicates for $p = 10$. The standard errors are given in parentheses.

The leverage design shows more differences. L_2Boosting performs quite badly, while *RobLoss* and *RobRegM* boosting are the worst of the robust methods. *RobCor* boosting outperforms the remaining methods by far.

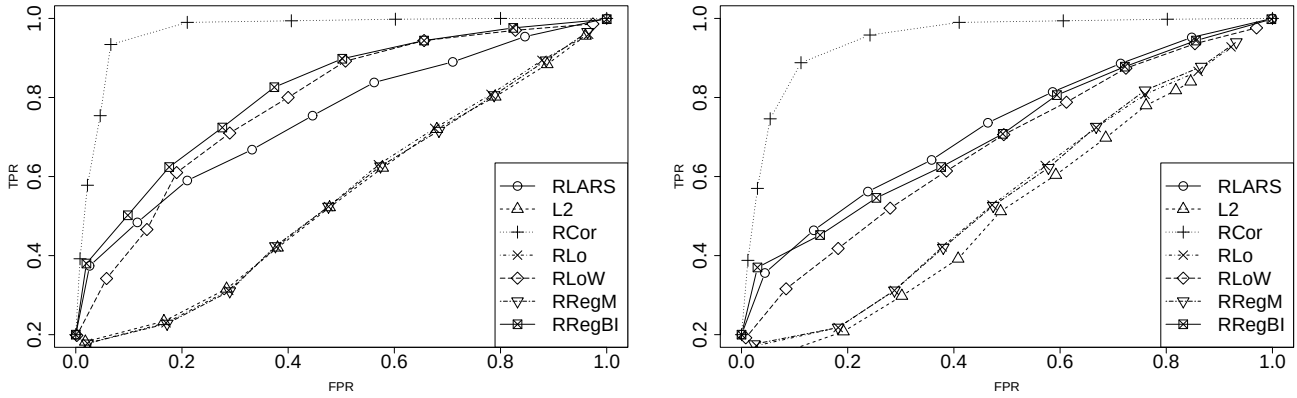
Furthermore, we'd like to compare the robust boosting methods with alternative methods such as robust LARS (*RLARS*). However, the methodology and available software for *RLARS* does not readily yield estimates for parameters and is rather intended for variable selection. Therefore, we compare the robust boosting methods and *RLARS* using ROC curves. In order to keep the number of figures at a manageable size, we only show the ROC curves for $e1$ and $e4$ for both designs (averaged over 100 repetitions). For the normal design and error $e1$ (Figure 1a), all methods perform equally well. In the presence of severe outliers (Figure 1b), L_2 boosting performs very badly, while the remaining methods perform comparably well. In the leverage design (Figure 2), *RobCor* is clearly dominant for both error settings.

In order to give an impression of the runtime, we give the processor time for one method call averaged over 100 repetitions. We only show the results for error $e1$ and both designs. All calculations were done on an AMD Athlon 64 X2 Dual Core Processor 5000+ with 2.6 GHz and 4 GB RAM running on Linux and using R 2.5.1. For the boosting methods, the maximal number of iterations was chosen large enough such that the estimates (with the maximal



(a) Normal design - $e1$ (b) Normal design - $e4$

Fig. 1. $p = 10$**, Normal design:** ROC curves comparing the variable selection properties of different robust boosting methods and *RLARS*.



(a) Leverage design - $e1$ (b) Leverage design - $e4$

Fig. 2. $p = 10$**, Leverage design:** ROC curves comparing the variable selection properties of different robust boosting methods and *RLARS*.

number of iterations) overfit slightly. The result is given in table 2. For all methods, the runtime for the leverage design is larger. *RobCor* and *RLARS* are especially fast in the leverage design (L_2 needs more iterations to reach its optimum which explains why it is slower).

4.4 Results for $p = 100$

Table 3 gives the average (over 100 and 20 replicates for normal and leverage design, respectively) of the mean squared prediction error when stopping with the validation set.

In the normal design, L_2 boosting with error $e1$ performs best and *RobCor*

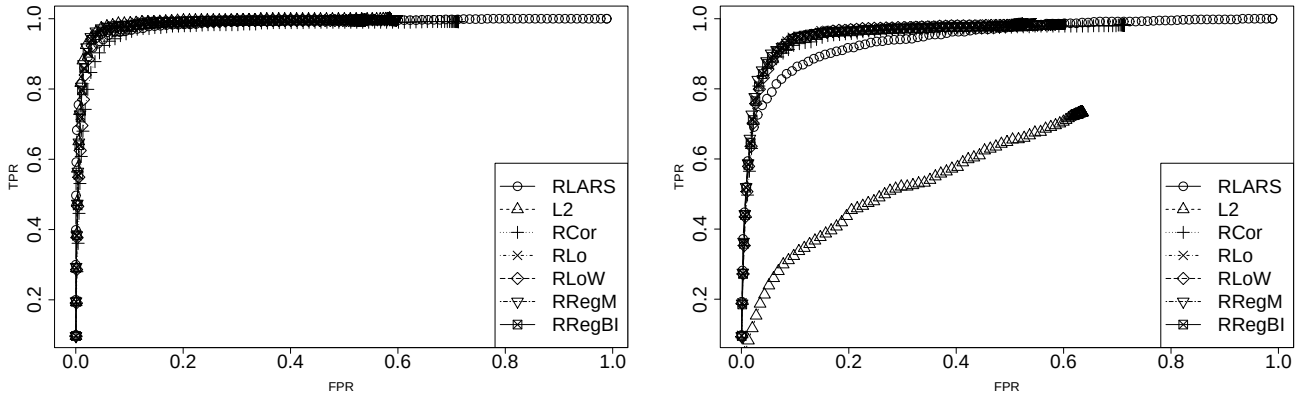
Method	$ave(t_{Normal})$ [s]	$ave(t_{Leverage})$ [s]
L_2	2.5 (0.3)	7.3 (0.5)
RobLoss	3.6 (0.5)	11.0 (0.6)
RobRegM	51 (7)	153 (9)
RobLossW	3.7 (0.4)	7.4 (0.6)
RobRegBI	22 (3)	59 (7)
RobCor	3.9 (0.3)	4.2 (0.4)
RLARS	2.7 (0.1)	2.8 (0.1)

Table 2
Average runtime in seconds over 100 repetitions for both designs. Standard errors are given in parentheses. *RobRegM* is particularly slow, while *RLARS* and *RobCor* are very fast, especially in the leverage design.

Design	Method	$e1$	$e2$	$e3$	$e4$
Normal	L_2	241 (9)	744 (30)	34680 (32287)	40706 (25760)
	RobLoss	267 (10)	399 (17)	330 (13)	436 (20)
	RobRegM	273 (10)	397 (16)	336 (13)	435 (18)
	RobLossW	293 (10)	413 (18)	347 (12)	457 (20)
	RobRegBI	286 (10)	420 (18)	347 (12)	455 (19)
	RobCor	381 (13)	519 (22)	448 (18)	481 (16)
Leverage	L_2	1279 (33)	1742 (72)	1888 (144)	77419 (73321)
	RobLoss	1352 (39)	1598 (72)	1430 (44)	1548 (45)
	RobRegM	1350 (39)	1581 (66)	1433 (43)	1548 (46)
	RobLossW	1422 (37)	1664 (57)	1523 (47)	1660 (43)
	RobRegBI	1429 (38)	1644 (54)	1516 (47)	1697 (50)
	RobCor	1691 (74)	1739 (74)	1748 (105)	1629 (83)

Table 3
Mean squared prediction error of the different boosting methods when stopping with a validation set, averaged over 100 replicates for the normal and 20 replicates for the leverage design for $p = 100$. The standard errors are given in parentheses.

worst. If the outliers get more severe, L_2 boosting performs worse. *RobCor* stays the worst of the robust boosting methods in terms of mean squared prediction error. In the setting with leverage design, the performance of L_2 boosting decreases with increasing severity of the outliers. At first sight it might surprise, that the prediction error in the normal design and with error $e3$ is so much larger than the one in the leverage design. However, the standard error in the normal design is huge, as well. This indicates that during the 100



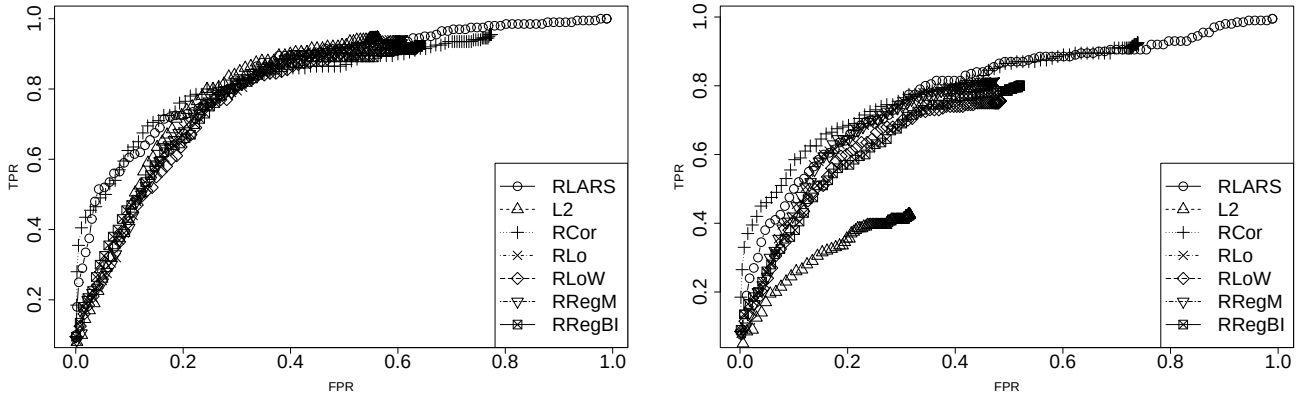
(a) Normal design - $e1$ (b) Normal design - $e4$

Fig. 3. $p = 100$**, Normal design:** ROC curves comparing the variable selection properties of different robust boosting methods and *RLARS*. In the case of error $e1$ all methods perform comparably well. If the severity of the outliers is increased, L_2 boosting breaks down and the robust boosting methods become slightly dominant over *RLARS*.

repetitions of the normal design setting a huge outlier was generated that accounts for most of the variance in the data. This rare event didn't occur in the leverage design setting, where only 20 replicates were computed. However, by inspecting the leverage setting using error $e4$ one sees that L_2 boosting in the leverage design is as vulnerable to outliers as in the normal design. The performance of the other robust boosting methods seems rather stable. Again, *RobCor* is the worst of the robust boosting methods in terms of prediction error.

As before, we compare the variable selection properties of the methods using ROC curves and show again only the ROC curves for $e1$ and $e4$ for both designs (averaged over 100 repetitions in the normal and 20 repetitions in the leverage design). For the normal design and error $e1$ (Figure 3a), all methods perform equally well. In the presence of severe outliers (Figure 3b), L_2 boosting performs very badly. All robust boosting methods slightly outperform *RLARS*. In the leverage design and with error $e1$ (Figure 4a), *RobCor* and *RLARS* are slightly dominant, while with error $e4$ *RobCor* becomes the dominant method (Figure 4b).

In order to give an impression of the runtime, we give the processor time for one method call averaged over 100 repetitions in the normal and 20 repetitions in the leverage design. We only show the results for error $e1$ and both designs (see section 4.3 for details on hardware and software) in table 4. The runtime for the leverage design is in general larger. *RobCor* is in the normal design among the three fastest methods but outperforms the others in the leverage design by far.



(a) Leverage design - $e1$ (b) Leverage design - $e4$

Fig. 4. $p = 100$**, Leverage design:** ROC curves comparing the variable selection properties of different robust boosting methods and *RLARS*. In the case of error $e1$ *RobCor* and *RLARS* are for small FPR slightly dominant over the other methods. With error $e4$, *RobCor* becomes the dominant method for small FPR.

Method	t_{Normal} [s]	$t_{Leverage}$ [s]
L_2	15.1 (0.1)	207 (17)
RobLoss	18.6 (0.7)	376 (41)
RobRegM	346 (14)	5267 (596)
RobLossW	33 (1.6)	293 (32)
RobRegBI	178 (6)	1037 (121)
RobCor	22.7 (0.4)	37 (4)
RLARS	204.4 (0.1)	219 (1.8)

Table 4
Average Runtime in seconds over 100 repetitions for normal and 20 repetitions for leverage design. Standard errors are given in parentheses. *RobRegM* is particularly slow and *RobCor* is very fast, especially in the leverage design.

5 Real data

5.1 Small Scale

As a real data set we analyze the measurements of Maguna et al. (2003) (see also Maronna et al. (2006)). There are 38 observations (17 monocarboxylic, 9 dicarboxylic and 12 unsaturated carboxylic acids) and the goal is to predict the logarithm of the aquatic toxicity (y) from nine molecular descriptors ($x_1, \ldots, x_9$). The scatterplot matrix of the data (not included) shows a quite good linear dependence between y and x_1 except for some outliers. The other

covariates have no clear "univariate" influence on y.

We applied the boosting methods with shrinkage factor $\nu = 0.3$ and used 5-fold cross-validation to stop the boosting iterations. The results are as follows: L_2-, *RobLoss* and *RobRegM* boosting select several times x_1 and x_3 at the beginning and then also some other covariates. The residual plots (not included) show no outliers. *RobLossW*, *RobRegBI* and *RobCor* boosting select only several times x_1 and then stop. The residual plots (not included) indicate some clear outliers that are leverage points.

A closer look at the data shows that all the clear outliers are unsaturated carboxylic acids. *RobLossW*, *RobRegBI* and *RobCor* boosting lead to the insight that there is no linear model which fits all the data well. L_2-, *RobLoss* and *RobRegM* boosting find a second variable (x_3) that seems to explain also the unsaturated carboxylic acids, but this is doubtful.

5.2 Large Scale

In addition to the small scale problem of the last section, we'd like to explore the feasibility of variable selection using robust boosting in very high-dimensional settings. As seen for high dimensions in section 4.4, *RobCor* is faster than most of the other robust methods tested and in addition to that, it is very competitive in variable selection. Therefore, *RobCor* seems to be the candidate of choice for robust variable selection using boosting in high-dimensions. In order to test the feasibility of the robust methods, we analyzed a dataset with $n = 71$ samples consisting of one continuous response variable and $p = 4088$ explanatory variables. The dataset arose from Riboflavin production in Bacillus subtilis in collaboration with DSM Nutritional Products: There are 4088 genes whose expression levels have a potential influence on the production of Riboflavin, which is the continuous response variable. The relation was measured in 71 different settings.

We used *RobCor* boosting and *RLARS* in order to identify the most important genes for Riboflavin production. Since this project is still ongoing, there is no definitive list of genes with which the results could be compared.

However, we compare the stability of both methods when introducing outliers in the original data set (10% leverage points with $p_{eff} = 5$, see section 4.1). We compute the intersection of the best 20 and 50 variables selected using both methods. This is done for the original data with and without outliers. In order to check the variability of the results, we used 10-fold cross-validation and averaged the results over the individual runs. The results are shown in Table 5. It can be seen, that this intersection rate between the methods stays constant even though outliers were introduced. The intersection rate is to be

interpreted as follows: from a total of $p = 4088$ genes, when selecting 20 or 50 genes with either method, about 5 or 10 of these genes were chosen by both methods, respectively. In such high-dimensional problems, the set of genes which are chosen by both methods can be a valuable source of information. Moreover, the runtime for each CV step was noted. *RobCor* took on average 275 seconds (s.e. 31 seconds), while *RLARS* took on average 5330 seconds (s.e. 5 seconds). I.e., *RobCor* was about an order of magnitude faster than *RLARS*.

	20 variables	50 variables
Original data	0.27 (0.05)	0.19 (0.03)
Original data with outliers	0.27 (0.04)	0.19 (0.03)

Table 5
Intersection rates: Intersection of selected variables using *RobCor* and *RLARS* divided by the number of variables selected (20 and 50). When introducing outliers, the ratio stays constant.

6 Conclusions

We compared several robust boosting methods and robust LARS to L_2Boosting. For the ideal normal case, the robust methods are only slightly worse than L_2Boosting. In the contaminated case though, the robust methods outperform L_2Boosting by a large margin. An advantage of the boosting methods (for example over robust LARS) is that they don't have to compute covariance matrices of the covariates or to identify multivariate leverage points.

RobLoss, *RobLossW* and *RobCor* boosting are computationally efficient and hence well suited also for truly high dimensional problems. In the high dimensional setting, the differences between the methods are less pronounced, because the methods not only have to cope with outliers but also with high dimensional observations.

The additional computations of *RobRegM* and *RobRegBI* boosting do not pay off. They have no clear advantage over *RobLoss* and *RobLossW* boosting in terms of prediction error or success in variable selection as measured by the ROC curve. However, they come with an immensely increased computational burden.

RobCor performs especially well at variable selection: In several settings it outperforms all other methods we tested and is never worse. In addition to that, *RobCor* is comparatively fast. Especially in the high-dimensional contaminated case, *RobCor* becomes the by far fastest method. The combination of computational efficiency and superior variable selection properties makes

RobCor the best of the analyzed candidates for high-dimensional robust variable selection.

In practice, it is always a good advice to employ more than one method and to compare the results. Our robustified versions of L_2Boosting offer additional possibilities for good, advanced data analysis.

References

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

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

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., 2004. Least angle regression. Annals of Statistics 32(2), 407–451.

Freund, Y., Schapire, R. E., 1996. Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning. pp. 148–156.

Friedman, J. H., 2001. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29 (5), 1189–1232.

Hampel, F., Ronchetti, E., Rousseeuw, P., Stahel, W., 1986. Robust Statistics: The Approach Based on Influence Functions. Wiley Series in Probability and Math. Statistics. Wiley, New York.

Huber, P. J., 1964. Robust estimation of a location parameter. Annals of mathematical statistics 35, 73–101.

Huber, P. J., 1981. Robust Statistics. Wiley, N. Y.

McCann, L., Welsch, R., 2007. Robust variable selection using least angle regression and elemental set sampling. Computational Statistics and Data Analysis 52, 249–257.

Maguna, F. P., Núñez, M. B., Okulik, N. B., Castro, E. A., 2003. Improved QSAR analysis of the toxicity of aliphatic carboxylic acids. Russian Journal of General Chemistry 73, 1792–1798.

Mallat, S., Zhang, Z., 1993. Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing 41(12), 3397–3415.

Maronna, R. A., Martin, R. D., Yohai, V. J., 2006. Robust Statistics, Theory and Methods. Wiley Series in Probability and Statistics. John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England.

Morgenthaler, S., Welsch, R. E., Zenide, A., 2003. Theory and Applications of Recent Robust Methods. Birkhäuser, Ch. Algorithms for Robust Model Selection in Linear Regression.

Ronchetti, E., Field, C., Blanchard, W., 1997. Robust linear model selection by

cross-validation. Journal of the American Statistical Association 92, 1017–1023.

Ronchetti, E., Staudte, R. G., 1994. A robust version of Mallow's c_p. Journal of the American Statistical Association 89, 550–559.

Rousseeuw, P. J., Croux, C., 1993. Alternatives to the median absolute deviation. Journal of the American Statistical Association 88 (424), 1273–1283.

Van Aelst, S., Khan, J. A., Zamar, R. H., 2004. Robust linear model selection based on least angle regression, http://hajek.stat.ubc.ca/˜ruben/website/cv/RobLARS.pdf.