Robustified L_2Boosting

Roman Werner Lutz and Peter Bühlmann

December 20, 2006

Abstract

We consider five robustifications of L_2Boosting for linear regression with various robustness properties. 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.

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.

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, Hastie, Johnstone and Tibshirani 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.

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, Field and Blanchard 1997, Morgenthaler, Welsch and Zenide

2003), our approaches are closest to "robust LARS" (Van Aelst, Khan and Zamar 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

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

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

2. Increase m by 1. Compute the negative gradient (also called pseudo response), which 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 smaller values have empirically proven to be a better choice. 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, Ronchetti, Rousseeuw and Stahel 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):

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

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. This would yield factors slightly greater than 1. 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". But it is not possible to incorporate "X-direction-robustness" in the loss function and therefore, we do it in the base procedure when fitting the negative gradient in step 3. 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

$$\begin{aligned}
\hat{f}(x) &= \hat{\alpha}_{\hat{s}} + \hat{\beta}_{\hat{s}} x_{\hat{s}}, \\
(\hat{\alpha}_j, \hat{\beta}_j) &= \arg\min_{\alpha,\beta} \sum_{i=1}^{n} w_{ij}(r_i - \hat{\alpha}_j - \hat{\beta}_j x_{ij})^2, \\
\hat{s} &= \arg\max_{1 \le j \le p} |\hat{\beta}_j| \cdot Q_n(\mathbf{x}_j).
\end{aligned}$$

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

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

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, which 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, which 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

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

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:

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

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 $\nu = 1$).

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 to 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 the 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 when 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.

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. 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 5 effective and 5 noise variables for $p = 10$ or 10 and 90 for $p = 100$, respectively. We use two design matrices for the covariates and two 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 multiplied by 5. This is done after the true y-values have been determined. Therefore we have bad leverage points. The error distributions are standard normal (N) and 90% standard normal and 10% from $\mathcal{N}(0, 5^2)$ (10%N5). 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

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

We use paired sample Wilcoxon tests to examine whether a method is significantly better than another. Additionally, as a measure of variable selection accuracy, we count how many variables have been chosen until the 5 (respectively 10) effective covariates have been selected (optimal would be 5 or 10, respectively).

4.3 Results for $p = 10$

	Normal design				Leverage design			
	N		10%N5		N		10%N5	
L2	7.7	(4.1)	25.5	(13.9)	138.4	(24.5)	148.4	(27.9)
RobLoss	8.9	(5.0)	11.4	(5.8)	87.2	(41.3)	102.8	(47.5)
RobRegM	8.8	(5.3)	11.4	(5.9)	86.1	(41.0)	99.9	(44.4)
RobLossW	9.1	(5.2)	11.5	(6.0)	19.7	(8.8)	25.4	(11.8)
RobRegBI	9.1	(4.8)	11.8	(6.0)	19.3	(8.8)	25.4	(12.2)
RobCor	11.3	(6.8)	13.8	(7.7)	15.5	(9.5)	18.8	(9.4)

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 deviations are given in parentheses.

Table 1 gives the average of the mean squared prediction error when stopping with the validation set. The results are as expected. For the normal design with normal errors, 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 10%N5, L_2Boosting performs much (and significantly) worse than the robust methods, and RobCor boosting is still significantly worse than the other robust methods.

The leverage design shows the biggest differences. L_2Boosting performs really bad and RobLoss and RobRegM boosting are not much better. RobCor boosting performs best and all differences are highly significant (except the pairs RobLoss-RobRegM and RobLossW-RobRegBI).

In table 2, we examine whether we can stop the boosting iteration at a good point. Given is the percentage loss when using a validation set (of same size as the training set) to stop the iteration compared to optimal stopping (stopping at the iteration which gives smallest

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	13	10	48	45
RobLoss	25	22	14	19
RobRegM	25	21	13	16
RobLossW	24	21	19	16
RobRegBI	23	24	19	19
RobCor	27	26	33	25

Table 2: Percentage loss of stopping with a validation set compared to optimal stopping for $p = 10$.

mean squared prediction error, also called "oracle" stopping). The stopping works quite satisfyingly for L_2Boosting for the normal design, but not so well for the robust methods. They lose about 20% in performance compared to optimal (oracle) stopping.

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	5.2	5.7	6.7	7.1
RobLoss	5.2	5.3	6.6	6.7
RobRegM	5.1	5.2	6.6	6.9
RobLossW	5.2	5.2	5.9	6.1
RobRegBI	5.2	5.2	5.4	5.7
RobCor	5.3	5.4	5.4	5.6

Table 3: Average number of covariates that have been chosen until all 5 effective variables have been selected for $p = 10$.

In table 3 we state how many variables have been chosen (on average) until all 5 effective covariates have been selected. The outcomes confirm more or less the results of table 1, in the sense that the methods with bad prediction performance usually also select too many variables. The big difference is that RobRegBI boosting selects the variables better than RobLossW boosting, although they have comparable predictive power. This indicates that some noise variables in the fitted model with small coefficients don't hurt for prediction.

4.4 Results for $p = 100$

The results for $p = 100$ are contained in tables 4, 5 and 6. This setting is much harder. The methods not only have to cope with outliers but also with high dimensional observations. The results are qualitatively more or less the same as for $p = 10$. The main differences are: for the normal design, RobCor performs much worse than the other robust methods. For the leverage design, the differences between the methods are less pronounced but there is now a significant difference between RobLossW and RobRegBI boosting.

The stopping of the boosting iteration is relatively easier in this high dimensional setting. The robust methods lose only about 10% compared to optimal stopping.

	Normal design				Leverage design			
	N		10%N5		N		10%N5	
L2	112	(41)	371	(166)	1060	(231)	1203	(220)
RobLoss	124	(45)	186	(81)	538	(210)	668	(252)
RobRegM	125	(45)	189	(77)	560	(242)	665	(245)
RobLossW	138	(49)	196	(87)	417	(170)	569	(221)
RobRegBI	134	(48)	200	(84)	389	(148)	525	(196)
RobCor	187	(77)	268	(129)	326	(149)	472	(231)

Table 4: Mean squared prediction error of the different boosting methods when stopping with a validation set, averaged over 100 replicates for $p = 100$. The standard deviations are given in parentheses.

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	4	6	126	68
RobLoss	9	8	8	13
RobRegM	10	8	12	11
RobLossW	9	9	6	9
RobRegBI	8	9	8	10
RobCor	8	7	8	13

Table 5: Percentage loss of stopping with a validation set compared to optimal stopping for $p = 100$.

	Normal design		Leverage design	
	N	10%N5	N	10%N5
L2	10.9	16.8	26.3	48.5
RobLoss	11.0	12.2	27.3	33.1
RobRegM	10.9	12.0	27.7	34.3
RobLossW	11.4	12.4	25.5	34.8
RobRegBI	11.1	12.2	20.9	26.9
RobCor	12.1	13.9	16.9	20.3

Table 6: Average number of covariates that have been chosen until all 10 effective variables have been selected for $p = 100$.

5 Real data

As a real data set we analyze the measurements of Maguna, Núñez, Okulik and Castro (2003) (see also Maronna, Martin and Yohai 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, which 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.

6 Conclusions

We compared several robust boosting methods 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 computation of RobRegM boosting does not pay off. It has no advantages over RobLoss boosting. The case is different for RobRegBI boosting. It is better than RobLossW boosting in variable selection and the predictions are more accurate in high dimensions. The prediction and variable selection performance of RobCor boosting are surprisingly good in the contaminated case.

It seems quite natural that it is harder to stop the robust boosting methods than L_2Boosting in the normal case, since they use a robust measure of prediction error. In the leverage case though, the stopping works quite well for the robust methods and disastrously for L_2Boosting. The stopping works also well in the high dimensional setting, for which the boosting methods are mainly designed.

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. and Tibshirani, R. (2004). Least angle regression, *Annals of Statistics* **32(2)**: 407–451.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm, *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. and 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.

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

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

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

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

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

Ronchetti, E., Field, C. and Blanchard, W. (1997). Robust linear model selection by cross-validation, *Journal of the American Statistical Association* **92**: 1017–1023.

Rousseeuw, P. J. and 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. and Zamar, R. H. (2004). Robust linear model selection based on least angle regression. *http://hajek.stat.ubc.ca/˜ruben/website/cv/RobLARS.pdf*