Conjugate direction boosting

June 21, 2005
Revised Version

Abstract

Boosting in the context of linear regression become more attractive with the invention of least angle regression (LARS), where the connection between the Lasso and forward stagewise fitting (boosting) has been established. Earlier it has been found that boosting is a functional gradient optimization. Instead of the gradient, we propose a conjugate direction method (CDBoost). As a result, we obtain a fast forward stepwise variable selection algorithm. The conjugate direction of CDBoost is analogous to the approximate gradient in boosting. Using this analogy, we generalize CDBoost to: (i) include small step sizes (shrinkage) which often improves prediction accuracy; (ii) the non-parametric setting with fitting methods such as trees or splines, where least angle regression and the Lasso seem to be unfeasible. The role of the step size in CDBoost can also be seen as governing the degree between ℓ_0- and ℓ_1-penalisation: in linear regression with orthonormal design, the optimal (large) or infinitesimally small step sizes correspond to the ℓ_0- or ℓ_1-penalty respectively. This makes CDBoost surprisingly flexible. We compare the different methods on simulated and real datasets. CDBoost achieves the best predictions mainly in complicated settings with correlated covariates, where it is difficult to determine the contribution of a given covariate to the response. The gain of CDBoost over boosting is especially high in sparse cases with high signal to noise ratio and few effective covariates.

Key words: conjugate direction optimization, forward stepwise variable selection, high-dimensional linear regression, least angle regression, non-parametric regression, orthogonal greedy algorithm.

1 Introduction

The problem of subset selection in linear regression has been revived by the invention of least angle regression (LARS; Efron, Hastie, Johnstone and Tibshirani 2004), where a connection between the Lasso (Tibshirani 1996) and forward stagewise linear regression has been established. A closely related idea of forward stagewise fitting originates from the machine learning community under the name of boosting (Schapire 1990, Freund 1995, Freund and Schapire 1996). As an ensemble scheme, a fitting method (called the weak or base learner) is repeatedly applied to reweighted data and its outputs are averaged to form the boosting estimator. Freund and Schapire's (1996) AdaBoost for binary classification is most popular, very often in conjunction with trees as base learner. An extensive overview over boosting is given for example in Meir and Rätsch (2003).

Further important work (Breiman 1998, 1999, Friedman 2001, Rätsch, Onoda and Müller 2001) has revealed that boosting is a functional gradient descent method. It can be applied in a variety of settings with different loss-functions and many fitting methods. The use of the squared error loss for linear and non-parametric regression was proposed by Friedman (2001) under the name LS_Boosting and further extended and analyzed by Bühlmann and Yu (2003) under the name L_2Boosting. Other work on boosting for regression include Duffy and Helmbold (2000), Zemel and Pitassi (2001) and Rätsch, Demiriz and Bennett (2002).

In this paper we propose CDBoost, a method for linear regression with many covariates (or linear basis expansions with overcomplete dictionaries) which uses a conjugate direction method instead of the gradient method. As a result, we obtain a fast method for some type of forward stepwise variable selection in linear regression (and linear basis expansions). It turns out that it produces the same solutions as the orthogonal greedy algorithm in function approximation (cf. Temlyakov 2000), although the algorithms are different. It is well known that forward stepwise variable selection algorithms can be unstable resulting in poor predictive performance. In analogy to boosting using the concept of a conjugate direction instead of an approximate gradient, we can include small step sizes or shrinkage in CDBoost when moving along conjugate directions. The algorithm decelerates, becomes more stable and the prediction accuracy can be greatly enhanced. For linear models with orthonormal design, the optimal (large) step size yields the hard-threshold estimator (ℓ_0-penalisation), while the infinitesimally small step size results in the soft-threshold estimator (ℓ_1-penalty). The latter is also true for boosting. For general designs, however, the role of the step-size in boosting is very different because we will not get a forward stepwise variable selection method (i.e. forward ℓ_0-penalisation). Varying the step-size thus makes CDBoost surprisingly flexible. Another advantage of conjugate directions is that we can use our method with any fitting method such as trees and smoothing splines. While this is analogous to boosting, it marks an essential difference to the Lasso or least angle regression (LARS).

2 Linear Regression

In univariate linear regression we assume a continuous response $\mathbf{y} \in \mathbb{R}^n$ and a set of d covariates $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) \in \mathbb{R}^{n \times d}$. Here n denotes the sample size. The response $\mathbf{y}$ is modeled as a linear combination of the covariates plus a random error. In matrix notation:

$$\begin{aligned} &\mathbf{y} = \mathbf{X}\beta + \epsilon, \qquad \beta \in \mathbb{R}^d,\ \epsilon \in \mathbb{R}^n, \\ &\epsilon_1, \ldots, \epsilon_n \text{ i.i.d. with } \mathbf{E}[\epsilon_i] = 0,\ \mathrm{Var}(\epsilon_i) = \sigma^2. \end{aligned}$$

We assume w.l.o.g. that all covariates and the response are centered to have mean zero, so we need not worry about an intercept. We always assume that rank($\mathbf{X}$) = min($n-1, d$); note the reduced rank $n-1$ (if $d \geq n-1$) due to the centering of the covariates. Parameter estimation is most often done by least squares, minimizing the loss function

$$L(\beta) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\beta\|^2 = \frac{1}{2}\beta^T\mathbf{X}^T\mathbf{X}\beta - \mathbf{y}^T\mathbf{X}\beta + \frac{1}{2}\mathbf{y}^T\mathbf{y}.$$

Assuming $d \leq n-1$ and $\mathbf{X}$ of full rank d, we find the unique solution $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

2.1 Methods for variable selection

For $d > n-1$, the matrix $\mathbf{X}^T\mathbf{X}$ becomes singular and the least squares solution is not unique. For fairly large $d \leq n-1$, the least squares solution will often over-fit the data. In both cases we need a regularisation strategy. We focus here on methods that do variable selection.

2.1.1 Forward stepwise variable selection

We start with the empty model and sequentially add that covariate to the model which most improves the fit (most reduces the loss function L) when (re-) estimating regression coefficients by least squares corresponding to previously and currently selected covariates. Forward stepwise variable selection is a very greedy method. Covariates which are highly correlated with variables already included in the model have a small chance of being chosen.

2.1.2 L_2Boosting

We present here only the key ideas of L_2Boosting with componentwise linear least squares. Technical details are given in section 3, where we show that L_2Boosting is an approximate gradient optimization. More details can also be found in Friedman (2001) and Bühlmann and Yu (2003).

L_2Boosting is an iterative method which starts with the empty model (all β-coefficients equal to zero). In each iteration, the current residuals are fitted against the one covariate which gives the best least squares fit (using only one single covariate). The model is updated by adjusting only the coefficient corresponding to the chosen covariate; all other coefficients remain unchanged. This is a main difference to forward stepwise variable selection: one boosting update is computationally faster, but the coefficients in every iteration are no longer least squares estimates (in sub-models). But it is possible to choose a covariate again to adjust a coefficient which was badly estimated in an earlier step.

2.1.3 Conjugate direction boosting

Here, we give a brief sketch of our conjugate direction boosting algorithm (CDBoost). It turns out that our algorithm yields the same solutions as the orthogonal greedy algorithm which is known in nonlinear function approximation (cf. Temlyakov 2000). But our formulation with conjugate directions easily allows for extensions which include shrinkage (small step sizes) or which are applicable to the non-parametric settings.

CDBoost is a hybrid between forward stepwise variable selection and L_2Boosting. The selection of a new covariate is as in boosting: take the one covariate which fits the current residuals best. The coefficient update is like forward stepwise variable selection: compute the least squares solution in the model with the previously and currently selected covariates, by using a conjugate direction method. In other words: the conjugate direction boosting selects the covariate which would most improve the fit without adjusting the coefficients of the other variables. But after the selection, we adjust all the non-zero coefficients. Therefore, conjugate direction boosting (like forward stepwise variable selection but unlike L_2Boosting) finds the least squares solution (or a perfect fit) in $\min(n-1, d) = \operatorname{rank}(\mathbf{X})$ steps (note the reduced rank $n-1$ (if $d > n-1$) due to centering of the covariates).

3 L_2Boosting

We denote by $F(\cdot) : \mathbb{R}^d \to \mathbb{R}$ the (linear) regression function of interest. The L_2Boosting algorithm with componentwise linear least squares as briefly outlined in section 2.1.2 can then be described as follows:

L_2Boosting algorithm for linear regression (function version):

Step 1: Center $\mathbf{x}_i$ and $\mathbf{y}$ to mean zero. Initialize $\hat{F}^{(0)}(x) \equiv 0$ and $m = 1$.
Step 2: Compute current residuals $r_i = y_i - \hat{F}^{(m-1)}(x_i)$ $(i = 1, \ldots, n)$.
Compute the coefficients of all simple linear regressions of $\mathbf{r}$ $(\in \mathbb{R}^n)$ against each covariate alone: $\hat{\beta}_j = \mathbf{x}_j^T\mathbf{r}/\mathbf{x}_j^T\mathbf{x}_j$ $(j = 1, \ldots, d)$.
Select the covariate which most reduces the L_2-loss: $\hat{k}^{(m)} = \arg\min_j \frac{1}{2}\|\mathbf{r} - \hat{\beta}_j\mathbf{x}_j\|^2$.
Update $\hat{F}^{(m)}(x) = \hat{F}^{(m-1)}(x) + \hat{\beta}_{\hat{k}^{(m)}} x_{\hat{k}^{(m)}}$.
Step 3: Increase iteration index m by one and go back to Step 2.

The number of iterations is usually estimated using a validation set or with cross validation.
In signal processing, this algorithm is known as matching pursuit (Mallat and Zhang 1993) and is used to decompose a signal into a linear expansion of waveforms.
The L_2Boosting algorithm can also be written as an approximate gradient method. This is not exactly the same as the functional gradient approach of Breiman (1998, 1999), Friedman (2001) and Rätsch et al. (2001), because we work entirely in the parameter space of β.
For a given β, the negative gradient of the loss function is

$$-\nabla L(\beta) = -(\mathbf{X}^T\mathbf{X})\beta + \mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta).$$

The idea is to optimize in the coordinate direction (thereby changing only one entry of β) which is most parallel to the negative gradient. This means we look for the $\hat{k}$ for which $| < -\nabla L, \mathbf{e}_{\hat{k}} > |$ becomes maximal, where $\mathbf{e}_{\hat{k}} \in \mathbb{R}^d$ is the unit vector with entry 1 at position $\hat{k}$. $\hat{k}$ is therefore merely the component of the negative gradient with the highest absolute value. We shall call $\mathbf{e}_{\hat{k}}$ gradient approximation.
Note that this alternative formulation is only equivalent to the above cited function version when all covariates are scaled to the same variance. We can then rewrite the L_2Boosting algorithm:

L_2Boosting algorithm for linear regression (gradient/coefficient version):

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length $(\mathbf{x}_i^T\mathbf{x}_i = 1)$. Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$ and $m = 1$.
Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and determine the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j | - \nabla L_j^{(m)}|$.
Update $\hat{\beta}$: component $\hat{k}^{(m)}$ changes to

$$\hat{\beta}_{\hat{k}^{(m)}}^{(m)} = \hat{\beta}_{\hat{k}^{(m)}}^{(m-1)} + \mathbf{x}_{\hat{k}^{(m)}}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and all other components remain unchanged.
Step 3: Increase iteration index m by one and go back to Step 2.

4 Conjugate direction boosting

4.1 Conjugate direction and gradient optimization

Instead of the gradient method, we can use a *conjugate direction method* to minimize the quadratic function

$$L(\beta) = \frac{1}{2}\beta^T\mathbf{X}^T\mathbf{X}\beta - \mathbf{y}^T\mathbf{X}\beta + \frac{1}{2}\mathbf{y}^T\mathbf{y} =: \frac{1}{2}\beta^T\mathbf{A}\beta - \mathbf{b}^T\beta + c,$$
$$\mathbf{A} = \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d} \text{ symmetric, positive definite}, \mathbf{b} = \mathbf{X}^T\mathbf{y} \in \mathbb{R}^d.$$

If $d > n-1$ (note the centering of the covariates to mean zero), the matrix $\mathbf{A}$ is not positive definite. This does not matter because we actually only use sub-matrices of $\mathbf{X}$ so that the corresponding $\mathbf{A}$ remains positive definite.

Conjugacy is a property similar to orthogonality. A set of nonzero vectors $\{\mathbf{p}_1, \ldots, \mathbf{p}_d\}, \mathbf{p}_i \in \mathbb{R}^d$ is said to be conjugate with respect to the symmetric positive definite matrix $\mathbf{A}$ if

$$\mathbf{p}_i^T\mathbf{A}\mathbf{p}_j = 0 \quad \text{for all } i \neq j.$$

The importance in conjugacy lies in the fact that we can minimize L in d ($\leq n-1$) steps by minimizing along the individual directions in a conjugate set. A conjugate direction method takes an arbitrary set of d conjugate directions and does individual minimization of L along these directions. In contrast to the gradient method, we reach the minimum after d ($\leq n-1$) steps.

The question is how to find a set of conjugate directions. The canonical *conjugate gradient method* does the job very efficiently during the optimization process and not in advance. It takes the negative gradient as the first direction $\mathbf{p}_1$. For $k > 1$, $\mathbf{p}_k$ is a linear combination of the actual negative gradient and the previous $\mathbf{p}_{k-1}$ only:

$$\mathbf{p}_k = -\nabla L^{(k)} + \frac{\nabla L^{(k)T}\mathbf{A}\mathbf{p}_{k-1}}{\mathbf{p}_{k-1}^T\mathbf{A}\mathbf{p}_{k-1}}\mathbf{p}_{k-1}.$$

We do not need to store all the previous elements $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{k-2}$: $\mathbf{p}_k$ is automatically conjugate to these vectors. A proof of this remarkable property can be found for example in Nocedal and Wright (1999).

The *conjugate gradient method* is the same as *Partial Least Squares* (see for example Phatak and de Hoog 2002) and does not employ variable selection.

4.2 CDBoost: conjugate direction boosting for linear regression

We now describe our conjugate direction method which employs variable selection. In each step, only one component of $\hat{\beta}$ should change from zero to non-zero. We achieve this by choosing a special set of conjugate directions.

The first step is identical to gradient boosting: we look for the component of the negative gradient with the highest absolute value, say component $\hat{k}^{(1)}$. The vector $\mathbf{e}_{\hat{k}^{(1)}}$ is then the best approximation to the negative gradient (among all $\mathbf{e}_k$'s) in terms of having maximal absolute inner product. We then optimize along that first direction $\mathbf{p}_1 = \mathbf{e}_{\hat{k}^{(1)}}$.

In the following steps we have to determine a direction that is conjugate to all previous directions. Again, we compute the negative gradient and its approximation by a unit-coordinate vector $\mathbf{e}_{\hat{k}^{(m)}}$. The new direction $\mathbf{p}_m$ is a linear combination of the gradient approximation and all the previous directions. It is easy to compute the coefficients of the linear combination so that the new $\mathbf{p}_m$ is conjugate to all previous directions. In the m-th iteration we have

$$\mathbf{p}_m = \sum_{j=1}^{m-1} \lambda_j \mathbf{p}_j + \mathbf{e}_{\hat{k}^{(m)}}.$$

The $\lambda's$ are determined by the $m-1$ equations:

$$\mathbf{p}_i^T \mathbf{A} \left(\sum_{j=1}^{m-1} \lambda_j \mathbf{p}_j + \mathbf{e}_{\hat{k}^{(m)}} \right) = 0, \qquad i = 1, \ldots, m-1.$$

Using the property of conjugacy, we have

$$\lambda_i \mathbf{p}_i^T \mathbf{A} \mathbf{p}_i + \mathbf{p}_i^T \mathbf{A} \mathbf{e}_{\hat{k}^{(m)}} = 0, \qquad i = 1, \ldots, m-1,$$

and

$$\lambda_i = -\frac{\mathbf{p}_i^T \mathbf{A} \mathbf{e}_{\hat{k}^{(m)}}}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \qquad i = 1, \ldots, m-1.$$

Now we can formulate the

CDBoost algorithm with L_2-loss for linear regression:

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length. Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$ and $m = 1$.

Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T (\mathbf{y} - \mathbf{X} \hat{\beta}^{(m-1)}) \tag{1}$$

and take the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j | -\nabla L_j^{(m)}|$. Define the gradient approximation by $\mathbf{e}_{\hat{k}^{(m)}}$ (unit coordinate vector).

For $i = 1, \ldots, m-1$ compute the coefficients for the linear combination (for $m = 1$ there are no $\lambda's$):

$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T \mathbf{X}^T \mathbf{X} \mathbf{e}_{\hat{k}^{(m)}}}{\mathbf{p}_i^T \mathbf{X}^T \mathbf{X} \mathbf{p}_i}.$$

Compute the new direction (for $m = 1$ the sum vanishes)

$$\mathbf{p}_m = \sum_{i=1}^{m-1} \lambda_i^{(m)} \mathbf{p}_i + \mathbf{e}_{\hat{k}^{(m)}}. \tag{2}$$

Minimize along the direction of $\mathbf{p}_m$: i.e., update

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \frac{(\mathbf{y} - \mathbf{X} \hat{\beta}^{(m-1)})^T \mathbf{X} \mathbf{p}_m}{\mathbf{p}_m^T \mathbf{X}^T \mathbf{X} \mathbf{p}_m} \mathbf{p}_m. \tag{3}$$

Step 3: Increase iteration index m by one and go back to Step 2.

The number of iterations is again estimated using a validation set or cross validation.

We can take advantage of sparse vectors when implementing the CDBoost algorithm, in particular when $d \gg n$. The vectors $\mathbf{p}_m$ and $\hat{\beta}^{(m)}$ have m non-zero entries, and the unit-coordinate vectors $\mathbf{e}_k$ have only one non-zero entry. Thus, the terms $\mathbf{X} \mathbf{e}_{\hat{k}^{(m)}}$, $\mathbf{X} \mathbf{p}_i$ ($i =$

$1, \ldots, m$) and $\mathbf{X}\hat{\beta}^{(m-1)}$ can be computed efficiently in $O(nm)$ operations, and there is no need to compute $\mathbf{X}^T\mathbf{X}$ which is of particular interest if d is (very) large.

CDBoost is not as simple as the *conjugate gradient method*, because we have to store all the previous directions. On the other hand we have an algorithm which performs variable selection because only one coefficient of $\hat{\beta}^{(m)}$ in each step changes from zero to non-zero.

4.2.1 A fast forward forward stepwise variable selection algorithm

Here, we establish an equivalence of CDBoost to the orthogonal greedy algorithm (cf. Temlyakov 2000). The latter is similar to forward stepwise variable selection as described in section 2.1.1. It also employs least squares based on the selected covariates but a new covariate in the m-th selection step is chosen as the minimizer, with respect to j, of

$$|\mathbf{x}_j^T\mathbf{r}|/\mathbf{x}_j^T\mathbf{x}_j, \tag{4}$$

where $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)}$ are the current residuals. Note that forward stepwise variable selection chooses the covariate such that the loss $L(\beta)$ is minimized when fitting least squares on the previously and currently selected covariates.

Proposition 1 *The solutions of CDBoost and the orthogonal greedy algorithm coincide in every iteration $m \leq \min(n-1, d) = rank(X)$. In particular, in every iteration CDBoost employs least squares fitting in the corresponding sub-model of selected covariates.*

Proof: (i) We first show that for every iteration m, $\hat{\beta}^{(m)}$ is the least squares solution using the selected m covariates. If we started our algorithm again with only these m covariates, we would reach the same model after m iterations and this is the least squares solution because we use a conjugate direction method. Thus, we have proved that CDBoost yields least squares solutions when fitting $\mathbf{y}$ to the selected covariates.

In particular, in every iteration CDBoost selects a covariate which has not been selected before and the saturated model is reached after $\min(n-1, d)$ iterations.

(ii) For the equivalence to the orthogonal greedy algorithm (OGA), we only need to show that CDBoost selects the same covariates as the OGA (since both algorithms employ least squares fitting on the selected covariates). But this holds by the equivalence of formula (1) for CDBoost and (4) for OGA (note that (1) is for standardized variables with $\mathbf{x}_j^T\mathbf{x}_j = 1$). $\square$

It should be pointed out that CDBoost and the orthogonal greedy method are different algorithms. As a conjugate direction method, CDBoost has natural generalizations to be used with small step sizes (see section 5) and to the non-parametric setting (see section 8).

4.2.2 Computations

We are interested in the computational complexity for computing the full path of solutions of a method: that is, all possible solutions from the empty model with no covariates to the fully saturated model from least squares (if $d \leq n-1$) or with zero residual sum of squares (if $d \geq n$).

Proposition 2 *The computational complexity of CDBoost for computing the full path of solutions is $O(nd\min(n, d))$.*

Proof: Consider the m-th iteration of CDBoost: the negative gradient can be computed at a cost of $O(nd)$, all the λ's at a cost of $O(nm)$ (reusing terms computed in earlier iterations), the new direction at a cost of $O(dm)$ and the update at a cost of $O(nd)$. Thus, the computational cost for the m-th iteration is

$$O(nd) + O(nm) + O(dm). \tag{5}$$

Because the fully saturated model is fitted after $\min(n-1, d)$ iterations (which follows from Proposition 1), we get $m \leq \min(n-1, d)$. Hence by formula (5), the cost for computing the whole path of CDBoost solutions is $O(nd \min(n, d))$. □

The result from Proposition 2 can be compared with other methods. The computation of the whole path of the LARS solutions is of the same computational complexity $O(nd \min(n, d))$ (cf. Efron et al. 2004; in case of $d > n - 1$, their statement of $O(n^3)$ is wrong and should be $O(n^2 d) = O(nd \min(n, d))$). Thus, CDBoost and LARS are comparable in terms of computational cost: while the former does least squares on the selected variables, the latter employs shrinkage from ℓ_1-penalisation.

The full path of classical forward stepwise variable selection costs $O(nd^2)$ (cf. Miller 2002), regardless whether d is smaller or greater than n. Thus, for $d < n$, forward stepwise variable selection is as fast as CDBoost or LARS; for $d \gg n$, the situation is markedly different and forward stepwise variable selection is no longer linear in the dimension d. The main point is that forward stepwise variable selection depends on the square of the number of covariates out of the model and CDBoost depends on the square of the number of covariates in the model.

The computational cost for the path of L_2Boosting until boosting iteration m is $O(ndm)$. In cases where the number of boosting iterations m is of the order of $\min(n, d)$, we have comparable computational speed to CDBoost or LARS. However, to compute the fully saturated model we would typically need infinitely many ($m = \infty$) iterations: the convergence speed depends in a complicated manner on the design matrix $\mathbf{X}$.

CDBoost and forward stepwise variable selection employ least squares fits based on the selected variables while LARS (Lasso) and L_2Boosting yield shrunken estimates. For cases where $d \gg n$, CDBoost can have a substantial computational advantage over forward stepwise variable selection.

5 Shrinkage or small step size

Noticed by Friedman (2001), the predictive accuracy of boosting (and forward stagewise fitting) can be improved by a simple shrinkage strategy: in each boosting step, only a small fraction ν (for example $\nu = 0.1$) of the optimal update is added. This can be interpreted as a small step-size along an approximate gradient. Of course, there is a computational cost when using small step sizes, but it usually pays off as a clear improvement in accuracy.

We modify the update in *Step 2* of the L_2Boosting algorithm from section 3 to

$$\hat{F}^{(m)}(x) = \hat{F}^{(m-1)}(x) + \nu \cdot \hat{\beta}_{\hat{k}^{(m)}} x_{\hat{k}^{(m)}} \quad \text{(function version)}$$

or

$$\hat{\beta}^{(m)}_{\hat{k}^{(m)}} = \hat{\beta}^{(m-1)}_{\hat{k}^{(m)}} + \nu \cdot \mathbf{x}^T_{\hat{k}^{(m)}}(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)}) \quad \text{(gradient version)}.$$

The implementation of a similar strategy with small step sizes for forward stepwise variable selection or the orthogonal greedy algorithm (see section 4.2.1) is not straightforward.

However, with CDBoost (which yields the same solutions as the orthogonal greedy algorithm, see Proposition 1) we have a natural concept of using small step sizes for the updates along the conjugate directions, as discussed next.

5.1 Conjugate direction boosting with restart

A new problem arises when we use shrinkage or small step sizes for CDBoost. Without shrinkage, each covariate can only be chosen once. This is no longer true with shrinkage because the coefficients $\hat{\beta}^{(m)}$ are no longer least squares solutions (in sub-models). Thus, it may happen that a covariate already included in the model fits the current residuals best. In this case, we cannot find a *new* direction as in (2) which is conjugate to all previous directions, because there are only m conjugate directions in an m-dimensional space. A possible remedy is discussed next.

An easy and effective solution to the problem is to restart the algorithm when a variable is chosen the second time. We take the actual $\hat{\beta}^{(m-1)}$ as the new starting $\hat{\beta}$, delete all the conjugate directions $\mathbf{p}_i$ and start again. A justification for restarting is described below in Proposition 3. To keep track of the total number of iterations, denoted by m, we formulate our algorithm by marking the starting points which are denoted by an index s.

CDBoost algorithm with shrinkage and L_2-loss for linear regression:

Step 1: Standardize $\mathbf{x}_i$ to zero mean and unit length. Standardize $\mathbf{y}$ to zero mean. Initialize $\hat{\beta}^{(0)} = \mathbf{0}$, $m = 1$ and $s = 1$.

Step 2: Compute the negative gradient

$$-\nabla L^{(m)} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})$$

and take the component with highest absolute value: $\hat{k}^{(m)} = \arg\max_j | -\nabla L_j^{(m)}|$. Define the gradient approximation by $\mathbf{e}_{\hat{k}^{(m)}}$ (unit coordinate vector).

If $\hat{k}^{(m)} \in \{\hat{k}^{(s)}, \ldots, \hat{k}^{(m-1)}\}$ restart: Set $s = m$.

- If $m = s$:
$$\mathbf{p}_m = \mathbf{e}_{\hat{k}^{(m)}}.$$
- If $m > s$: For $i = s, \ldots, m-1$ compute the coefficients for the linear combination
$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T\mathbf{X}^T\mathbf{X}\mathbf{e}_{\hat{k}^{(m)}}}{\mathbf{p}_i^T\mathbf{X}^T\mathbf{X}\mathbf{p}_i},$$
and compute the new direction
$$\mathbf{p}_m = \sum_{i=s}^{m-1} \lambda_i^{(m)}\mathbf{p}_i + \mathbf{e}_{\hat{k}^{(m)}}.$$

Minimize along the direction of $\mathbf{p}_m$, i.e., update $\hat{\beta}$:

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \nu\,\frac{(\mathbf{y} - \mathbf{X}\hat{\beta}^{(m-1)})^T\mathbf{X}\mathbf{p}_m}{\mathbf{p}_m^T\mathbf{X}^T\mathbf{X}\mathbf{p}_m}\mathbf{p}_m.$$

Step 3: Increase iteration index m by one and go back to Step 2.

As already indicated, by the inclusion of shrinkage we no longer have a proper conjugate direction method and thus, we do not reach the least squares solution after $\min(n-1, d) = \operatorname{rank}(\mathbf{X})$ iterations. With shrinkage, the goal is to generate a "good" path from the empty model to the least squares solution (which is obtained with possibly infinitely many iterations). The conjugate direction method is only used here to propose "good" directions rather than achieving fast convergence to a good submodel. In section 7 we show that shrinkage can dramatically improve the fits.

We now give an illustrating example with 3 correlated covariates, $\beta = (3, 2, 1)^T$, low noise and $\nu = 0.005$. Covariate 1 has highest absolute gradient value and is therefore chosen first in CDBoost with shrinkage. After a tiny update in direction $\mathbf{p}_1 = (1, 0, 0)^T$, covariate 1 is still the best choice. This leads to a restart and we again take variable 1 followed by a step in direction $\mathbf{p}_1$. This scenario is repeated until $\hat{\beta}^{(43)} = (1.03, 0, 0)^T$, where covariate 2 gets higher absolute gradient value. Then, we perform an update in the direction $\mathbf{p}_2 = (-0.81, 1, 0)^T$ which is conjugate to $\mathbf{p}_1$. In the last two steps (along the directions $\mathbf{p}_1$ and $\mathbf{p}_2$) we moved the fraction ν from $\hat{\beta}^{(42)}$ to the least squares solution of covariate 1 and 2.

Prior to these two steps, covariate 1 had a higher absolute negative gradient value than covariate 2 and the same is true after these two steps (see the Proof of Proposition 3 below). Thus, we choose variable 1 which causes a restart and a step in direction $\mathbf{p}_1$ followed by the choice of variable 2 and a step in direction $\mathbf{p}_2$ followed by another restart with variable 1. This is repeated until $\hat{\beta}^{(139)} = (1.96, 0.93, 0)^T$, where variable 3 gets highest absolute negative gradient value. Then we perform an update in the direction $\mathbf{p}_3 = (-0.43, -0.48, 1)^T$ conjugate to $\mathbf{p}_1$ and $\mathbf{p}_2$. The next choice is covariate 1 which causes again a restart and we repeatedly find the sequence of variables 1, 2, 3, restart until we reach (theoretically after ∞ steps) the least squares solution $\hat{\beta} = (3.03, 1.94, 1.02)^T$.

In summary: the selected covariate indices are

$$\underbrace{1,1,1,1,\ldots,1}_{\text{42 times}},\underbrace{1,2,1,2,1,2,1,2,\ldots,1,2}_{\text{95 times the pair (1,2)}},\underbrace{1,2,3,1,2,3,1,2,3,1,2,3,\ldots}_{\infty\text{ times the triple (1,2,3)}}.$$

It is no coincidence that the restart was always caused by covariate 1, as described by the following result (the repeating pattern of selected pairs and triples is also a more general fact).

Proposition 3 *The CDBoost algorithm has the following property: It is always the first chosen covariate $\mathbf{x}_{\hat{k}(1)}$ which causes a restart.*

Proof: Suppose we have an arbitrary $\hat{\beta}^{(m)}$ and j conjugate directions involving j covariates. Now we restrict everything (L, ∇L, $\hat{\beta}$) to these j covariates (denoted with $\cdot|_j$). We can minimize $L|_j$ (find the least squares solution for these j covariates) by j individual minimizations along the j conjugate directions. For each of the j minimizations we take only ν of the optimal step length and therefore we only move the fraction ν from $\hat{\beta}^{(m)}$ to the least squares solution of the j covariates. The new negative gradient after these j steps is

$$\begin{aligned} -\nabla L^{(m+j)}|_j &= \mathbf{X}|_j^T\Big(\mathbf{y} - \mathbf{X}|_j\big(\hat{\beta}^{(m)}|_j + \nu(\hat{\beta}^{(OLS)}|_j - \hat{\beta}^{(m)}|_j)\big)\Big) \\ &= \mathbf{X}|_j^T\Big(\nu\mathbf{y} - \nu\mathbf{X}|_j\hat{\beta}^{(OLS)}|_j + (1-\nu)\mathbf{y} - (1-\nu)\mathbf{X}|_j\hat{\beta}^{(m)}|_j\Big) \\ &= (1-\nu)(-\nabla L^{(m)}|_j) \end{aligned} \tag{6}$$

because the negative gradient at the least squares solution is $\mathbf{0}$. We see that the involved components of the negative gradient are scaled by the factor $1-\nu$ and therefore it is the same component which has highest absolute value before and after the j updates.

Suppose covariate $\hat{k}^{(1)}$ has highest absolute gradient value and CDBoost yields the following sequence of covariate indices at the beginning: $\hat{k}^{(1)}, \ldots, \hat{k}^{(j)}$, where all $\hat{k}^{(i)}, i=1,\ldots,j$ are different. It follows from (6) that the variable with index $\hat{k}^{(1)}$ has higher absolute gradient value than $\hat{k}^{(2)}, \ldots, \hat{k}^{(j)}$. In the next iteration we therefore choose either $\hat{k}^{(1)}$ (restart) or a completely new covariate (no restart necessary). □

5.2 Orthonormal design

Proposition 4 *Assume an orthonormal design* $\mathbf{X}$ *with* $d \leq n-1$*, satisfying* $\mathbf{X}^T\mathbf{X} = I$*. Then, for every iteration* m *there exists a threshold* λ *(depending on* m*) such that CDBoost equals:*
(i) the soft-threshold estimator if ν *is infinitesimally small;*
(ii) the hard-threshold estimator if $\nu = 1$*.*

Proof: First note that CDBoost and boosting coincide due to orthonormality of the design. Therefore: statement (i) follows from Bühlmann and Yu 2005, and statement (ii) is a direct consequence of Proposition 1 and the orthonormality of the design. □

For the special case with an orthonormal design matrix, CDBoost, boosting, both with infinitesimal shrinkage, and all versions of LARS yield the soft-threshold estimator; the latter follows from results in Efron et al. 2004. CDBoost has the interesting interpretation that changing ν from infinitesimally small to 1 results in moving from the soft- to the hard-threshold estimator; a larger ν points more towards the hard-threshold estimator. The same is true for boosting. However, beyond the setting of orthonormal designs, CDBoost is flexible enough to cover a whole range from shrinkage estimation/variable selection to least squares estimation/variable selection, while boosting in general (even with $\nu = 1$) does not yield least squares on selected variables.

6 Connections to LARS

Efron et al. (2004) recently proposed least angle regression (LARS). A modification of LARS yields all Lasso solutions and another modification yields forward stagewise fitting (boosting with infinitesimal shrinkage). When we use boosting with a small shrinkage parameter ν, we almost reproduce the forward stagewise fitting path. The bigger we take ν, the more we depart from this path.

For the rest of this section we assume infinitesimal shrinkage for boosting and CDBoost. LARS and boosting yield the same path, when all the β-coefficients move monotonically away from zero as the LARS or boosting iterations increase. In this case CDBoost also leads to that path. For more general cases, turnarounds of coefficients (for example decreasing after increasing) are possible with LARS. Forward stagewise fitting behaves differently in such a case: it drops that variable from the active set of selected covariates and leaves its coefficient unchanged in the next step. The CDBoost algorithm works in between LARS and boosting/forward stagewise. Two examples with 3 covariates demonstrate this behavior. The following tables show the traces (endpoints of linear pieces) of the $\hat{\beta}$-coefficients from zero to the least squares solutions.

Example 1: LARS/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2	0	0
5	3	0
2	10	6

Boosting/Stagewise

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2	0	0
5	3	0
5	6	3
2	10	6

Example 2: LARS

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7	0	0
13	5	0
19	3	5

Boosting/Stagewise/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7	0	0
13	5	0
15	5	2
19	3	5

LARS always finds the least squares solution after 3 fairly large steps (in principle we can also evaluate intermediate models between two steps). Forward stagewise fitting drops covariate 1 in example 1 and covariate 2 in example 2 after step 2, the corresponding coefficient remains and it needs one more step to reach the least squares solution (note the turnaround of coefficient $\hat{\beta}_1$ in example 1 and $\hat{\beta}_2$ in example 2).

CDBoost behaves like LARS in the first example and like forward stagewise fitting in the second. Remember that CDBoost takes a lot of tiny steps to produce one LARS-step. At the beginning it always takes covariate 1 until the first LARS-step is reproduced, then covariate 1 and 2 in an alternating fashion. In example 1 it is the first chosen variable which turns around and CDBoost behaves like LARS. This is because covariate 1 is always chosen after a restart and its coefficient is adjusted in each step. In example 2 it is not the first chosen variable which turns around and CDBoost behaves like forward stagewise fitting. The selected variable indices with CDBoost in example 2 are: $1, 1, \ldots, 1,\ 1, 2, 1, 2, \ldots, 1, 2,\ 1, 3, 1, 3, \ldots, 1, 3,\ 1, 3, 2, 1, 3, 2, \ldots$.

7 Simulations with linear regression

Now, we compare CDBoost with boosting, forward stepwise variable selection and the three LARS methods (LARS, Lasso and forward stagewise fitting) for simulated linear regression models.

In our simulation study, the data is split into three parts: a training set, a validation set, both of equal size n and a test set of size 1000. The training set is used to fit the models and the validation set for stopping the iterations in CDBoost, boosting or LARS (the number of iterations is chosen to minimize the validation error). Finally we use the test set to measure the prediction error on the test set:

$$\frac{1}{|\text{testset}|} \sum_{x_i \in \text{testset}} (\hat{F}(x_i) - F(x_i))^2,$$

where $F(x) = x^T\beta$ is the true underlying regression function.

CDBoost (and boosting) has a second tuning parameter, the shrinkage factor ν, which governs the degree between ℓ_0- and ℓ_1-penalisation. To compare CDBoost to boosting we can fix ν and compare the methods (although the interpretation of a larger ν is not the same for the two methods). To compare CDBoost and boosting to forward stepwise variable selection and the three LARS methods, we validate over a small set of different shrinkage factors (0.7, 0.5, 0.3, 0.1, 0.03, 0.01).

In the next two subsections we give some results of simulations where the model is always chosen to be linear with normally distributed errors.

7.1 Model 1: $\mathbf{n = 100, d = 10, d_{eff} = 5}$

7.1.1 Setup for model 1

The size of the training set is $n = 100$ and the number of covariates is $d = 10$. The number of covariates with effective influence on the response is only $d_{\text{eff}} = 5$.

We draw the covariates from a multivariate normal distribution. This is done in two steps by first producing x_i i.i.d. $\sim \mathcal{N}_d(0, \mathbf{I})$ and then multiplying $\mathbf{X}$ with the $d \times d$ matrix $\mathbf{B}$ (blank entries correspond to zero):

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & 1 & 1 & & \\ & & & & 1 & 1 & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 & 1 & 1 \\ & & & & & & 1 & 1 & 1 & 1 \\ & & & & & & & 1 & 1 & 1 \\ & & & & & & & & 1 & 1 \\ & & & & & & & & & 1 \end{pmatrix}.$$

Finally we arbitrarily permute the columns of $\mathbf{X}$. This leads to a correlation matrix with approximately each third entry equal to zero. The other two thirds of the correlations range from 0.2 to 0.9.

The next step is to choose the true β-vector: the first five components of β are $\beta_1, \ldots, \beta_5$ i.i.d. $\sim \mathcal{N}(5, 1)$, and the other five components are zero.

The last issue is to specify the variance for the normal distributed errors ϵ. We choose it via the more meaningful *signal-to-noise ratio* which we define as

$$stnr = \frac{\mathrm{Var}\,(F(x))}{\mathrm{Var}\,(\epsilon)}.$$

We select the desired signal-to-noise ratio and determine then the corresponding $\mathrm{Var}\,(\epsilon)$. A signal-to-noise ratio of 1 corresponds to a "population" R^2 equal to $\mathrm{Var}\,(F(x))\,/\mathrm{Var}\,(y) = 0.5$.

We simulate 100 datasets from each setting. Because the true β-coefficients are different for each simulation we cover a greater range of models. The overall performance is the mean over the 100 test errors. Paired sample Wilcoxon tests are used to quantify significance when comparing CDBoost to the other methods.

We report on the test error in a standardized form. We divide it by the test error of the best constant (location) model and multiply it by 100. Each method should be better than the best constant model and therefore achieve a value below 100. This standardized test error makes it possible to compare different settings. For convenience we still refer to it as test error.

7.1.2 Results for model 1

		$stnr = 9$			$stnr = 4$			$stnr = 1$		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	1.20			2.81			9.85		
	$\nu = 0.7$	1.10			2.34			7.47		
	$\nu = 0.5$	**1.08**			**2.22**			**7.32**		
	$\nu = 0.3$	1.08			2.24			7.39		
	$\nu = 0.1$	1.09			2.27			7.42		
	$\nu = 0.03$	1.09			2.28			7.53		
	$\nu = 0.01$	1.09			2.28			7.47		
Boo	$\nu = 1$	1.33	11.2	9e−05	2.91	3.7	1e−01	9.56	−3.0	5e−01
	$\nu = 0.7$	1.28	15.8	3e−05	2.71	15.7	2e−06	8.24	10.4	4e−04
	$\nu = 0.5$	1.20	11.8	4e−05	2.60	17.0	2e−05	7.71	5.3	2e−02
	$\nu = 0.3$	1.20	10.9	6e−03	**2.44**	8.6	2e−03	7.63	3.3	1e−01
	$\nu = 0.1$	1.20	10.5	2e−03	2.44	7.5	5e−04	7.50	1.0	8e−01
	$\nu = 0.03$	**1.20**	9.7	3e−02	2.44	7.4	9e−03	**7.46**	−0.9	6e−01
	$\nu = 0.01$	1.20	10.2	5e−03	2.44	7.1	5e−03	7.49	0.2	7e−01

Table 1: Comparison of conjugate direction boosting (CDB) and boosting (Boo) for various signal-to-noise ratios (*stnr*) and shrinkage factor ν for simulation model 1. Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and each *stnr* is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests.

Table 1 shows the comparison of CDBoost and boosting. CDBoost performs better and the difference is significant for the higher *stnr*'s. Both methods do substantially better with shrinkage. While boosting needs small ν's for best performance, CDBoost gives good results for the whole range of shrinkage factors.

	$stnr = 9$			$stnr = 4$			$stnr = 1$		
	test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	1.00			2.19			7.16		
Boo	1.16	16.3	3e−04	2.40	9.5	1e−02	7.35	2.7	5e−02
FVS	1.18	17.6	4e−04	2.93	33.6	3e−07	9.83	37.3	3e−11
LAR	1.13	12.4	1e−04	2.45	11.7	5e−05	8.20	14.6	1e−08
Las	1.10	9.6	4e−04	2.38	8.5	9e−04	7.97	11.4	1e−07
For	1.21	20.6	7e−08	2.52	15.3	4e−07	7.93	10.7	9e−07

Table 2: Comparison of conjugate direction boosting (CDB), boosting (Boo), forward stepwise variable selection (FVS) and the three LARS methods (LARS, Lasso, forward stagewise fitting) for simulation model 1. Given is the (standardized) mean of the test error (test e) over 100 simulations. CDBoost and boosting are validated over the shrinkage factor ν with candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. The loss compared to CDBoost is given in % and the p-values are from paired sample Wilcoxon tests.

In table 2 and figure 1 we compare CDBoost to boosting, forward stepwise variable selection and the three LARS methods. CDBoost and boosting are validated over the shrinkage

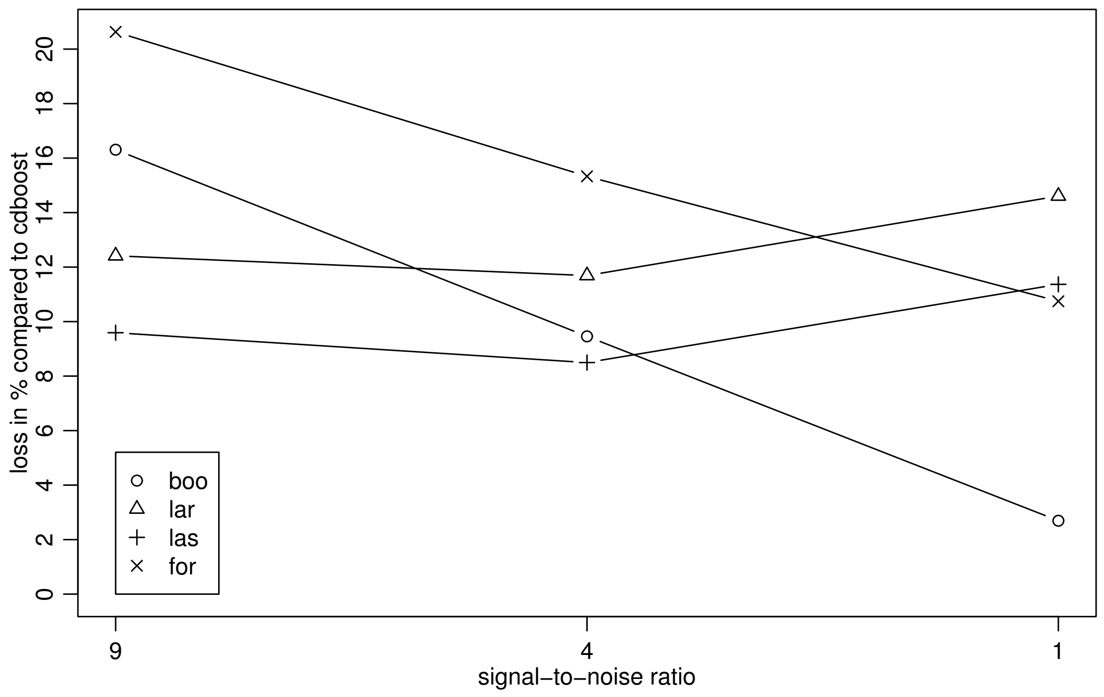

Figure 1: Percentage loss of the different methods (without forward stepwise variable selection) compared to CDBoost for simulation model 1.

factor ν with candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01, which gives better results compared to a fixed ν. This shows that in some of the 100 simulations a solution close to forward stepwise variable selection is appropriate and in other cases a solution closer to LARS is adequate. CDBoost (and also boosting) does a good job in choosing a right ν and is therefore very flexible.

CDBoost performs clearly and significantly best and forward stepwise variable selection leads to the worst models. The gain of CDBoost compared to boosting and forward stagewise increases with higher *stnr*'s and its gain compared to LARS and the Lasso is fairly constant. Forward stagewise fitting is always worse than boosting so infinitesimal shrinkage is not worthwhile in this example.

7.2 Model 2: $\mathbf{n = 50, d = 2000, d_{eff} = 10}$

7.2.1 Setup for model 2

We now reduce the size of the training set to $n = 50$ and increase the number of covariates to $d = 2000$, whereas $d_{\text{eff}} = 10$ variables have an effective influence on y. The covariates are constructed as in section 7.1.1 but with

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & & & & & & \\ 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 \\ & & & & & & 1 & 1 \end{pmatrix}$$

and without permuting the columns of $\mathbf{X}$. This leads to a correlation matrix with 2/3 in the secondary diagonals and 1/3 in the tertiary diagonals.

The ten non-zero coefficients are $\beta_{31}, \ldots, \beta_{35}, \beta_{66}, \ldots, \beta_{70}$ i.i.d. $\sim \mathcal{N}(5, 1)$. So there are two blocks of 5 correlated covariates with real influence on y. Because this setup is much harder than model 1 we use higher *stnr*'s.

7.2.2 Results for model 2

		$stnr = 16$			$stnr = 9$			$stnr = 4$		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	23.7			26.4			36.5		
	$\nu = 0.7$	15.9			21.7			**33.0**		
	$\nu = 0.5$	15.3			**21.5**			34.2		
	$\nu = 0.3$	**15.3**			21.7			34.5		
	$\nu = 0.1$	15.8			22.4			36.2		
	$\nu = 0.03$	16.0			22.8			36.9		
	$\nu = 0.01$	16.1			23.0			37.2		
Boo	$\nu = 1$	28.0	17.8	2e−07	30.4	15.1	5e−07	38.5	5.7	5e−03
	$\nu = 0.7$	20.4	28.2	3e−08	25.8	18.9	2e−05	**34.7**	5.2	2e−01
	$\nu = 0.5$	19.0	24.6	4e−11	24.1	11.9	3e−05	36.3	6.0	6e−03
	$\nu = 0.3$	18.0	18.2	2e−15	24.5	13.1	1e−12	37.2	7.8	1e−08
	$\nu = 0.1$	16.6	5.0	1e−10	**22.9**	2.3	3e−06	36.6	1.1	3e−02
	$\nu = 0.03$	16.4	2.5	8e−07	23.0	0.7	2e−02	37.0	0.2	2e−01
	$\nu = 0.01$	**16.4**	1.8	9e−05	22.9	−0.0	1e−00	37.1	−0.1	4e−01

Table 3: Comparison of conjugate direction boosting (CDB) and boosting (Boo) for various signal-to-noise ratios (*stnr*) and shrinkage factor ν for simulation model 2. Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and each *stnr* is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests.

Table 3 shows the comparison of CDBoost and boosting. The results are similar to that of model 1. CDBoost performs better than boosting, especially for high *stnr*. In this model, ν should not be too small for CDBoost.

In table 4 and figure 2 we compare CDBoost to boosting, forward stepwise variable selection and the three LARS methods. CDBoost and boosting are validated over the shrinkage factor ν with candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. Again, results are similar to model 1 with CDBoost performing significantly best.

	stnr = 16			*stnr* = 9			*stnr* = 4		
	test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	13.7			19.6			30.7		
Boo	15.1	10.1	1e−06	20.7	5.5	2e−02	32.1	4.6	7e−03
FVS	20.1	46.3	2e−07	24.1	22.7	7e−05	34.9	13.7	2e−02
LAR	15.8	14.6	3e−08	22.9	16.3	1e−11	37.6	22.4	1e−13
Las	15.7	14.0	1e−07	22.7	15.4	7e−11	37.2	21.2	1e−13
For	16.4	19.6	4e−13	23.0	17.1	3e−13	37.3	21.5	4e−14

Table 4: Comparison of conjugate direction boosting (CDB), boosting (Boo), forward stepwise variable selection (FVS) and the three LARS methods (LARS, Lasso, forward stagewise fitting) for simulation model 2. Given is the mean of the (standardized) test error (test e) over 100 simulations. CDBoost and boosting are validated over the shrinkage factor ν with candidate values 0.7, 0.5, 0.3, 0.1, 0.03, 0.01. The loss compared to CDBoost is given in % and the p-values are from paired sample Wilcoxon tests.

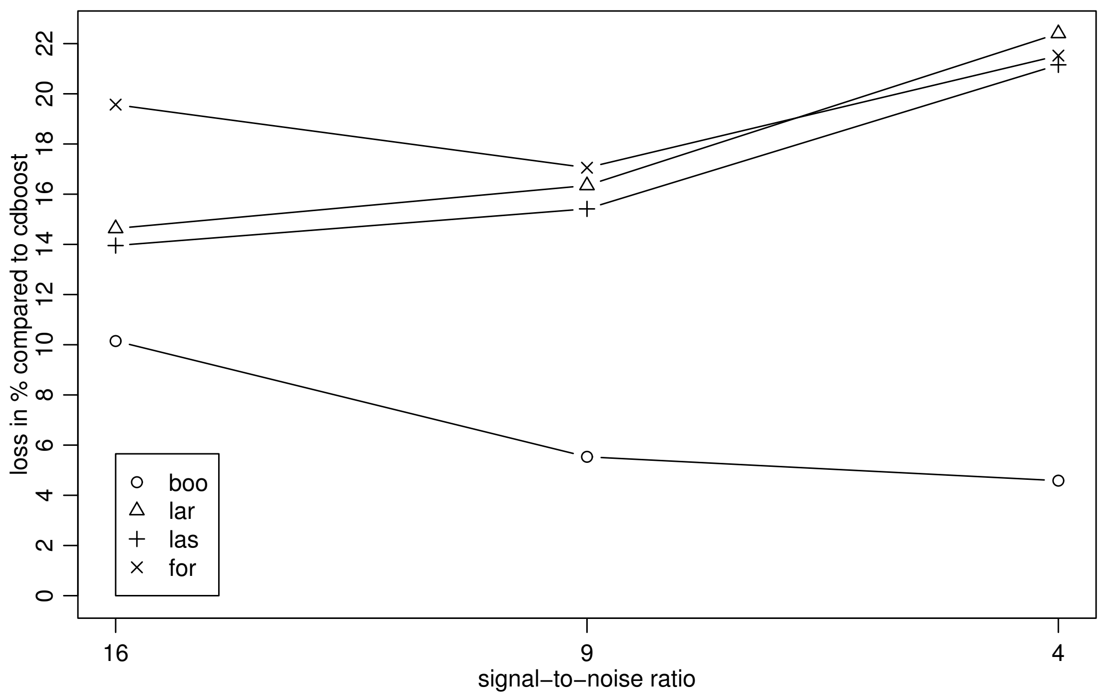

Figure 2: Percentage loss of the different methods (without forward stepwise variable selection) compared to CDBoost for simulation model 2.

8 Generalization to arbitrary learners

We now generalize our approach to any base learner, for example trees or componentwise smoothing splines (Bühlmann and Yu 2003, see also section 9) for non-parametric regression. In principle, we could use linear expansions in tree-type or B-spline basis functions, resulting in a huge design matrix. Instead, we develop an iterative, computationally more efficient approach. The idea is to construct a matrix $\tilde{\mathbf{X}}$ during the iteration process, rather than using a fixed design matrix $\mathbf{X}$ given in advance. In each iteration the learner produces a vector of fitted values for the training observations (which can be viewed as an estimated basis function) and that vector is taken as a new column of $\tilde{\mathbf{X}}$. Thus, $\tilde{\mathbf{X}}$ grows in each step by one column, and this matrix $\tilde{\mathbf{X}}$ is then used in our CDBoost method. The following algorithm makes the idea precise:

General CDBoost algorithm with L_2-loss:

Step 1: Standardize $\mathbf{y}$ to zero mean. The standardization of the $\mathbf{x}_i$ is not necessary. Initialize $\tilde{\mathbf{X}}$ = null, $\hat{\beta}^{(0)}$ = null and $m = 1$.

Step 2: Compute the current residuals $\mathbf{r} = \mathbf{y} - \tilde{\mathbf{X}}\hat{\beta}^{(m-1)}$ and fit the base learner to them using the predictor matrix $\mathbf{X}$. The fit is denoted by $\hat{f}^{(m)}(x)$ and the vector of fitted values by $\tilde{\mathbf{x}}_m \in \mathbb{R}^n$.

Increase $\tilde{\mathbf{X}}$ by one column by adding $\tilde{\mathbf{x}}_m$. Now $\tilde{\mathbf{X}}$ is of dimension $n \times m$. Extend $\hat{\beta}^{(m-1)}$ and all direction vectors $\mathbf{p}_i, i < m$, by a zero so that they all have length m.

For $i = 1, \ldots, m-1$ compute the coefficients for the linear combination of $\mathbf{p}_m \in \mathbb{R}^m$ (for $m = 1$ there are no $\lambda's$):

$$\lambda_i^{(m)} = -\frac{\mathbf{p}_i^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{e}_m}{\mathbf{p}_i^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{p}_i}.$$

Compute the new direction (for $m = 1$ the sum vanishes)

$$\mathbf{p}_m = \sum_{i=1}^{m-1} \lambda_i^{(m)} \mathbf{p}_i + \mathbf{e}_m.$$

Minimize in the direction of $\mathbf{p}_m$, that means update $\hat{\beta}$:

$$\hat{\beta}^{(m)} = \hat{\beta}^{(m-1)} + \frac{(\mathbf{y} - \tilde{\mathbf{X}}\hat{\beta}^{(m-1)})^T \tilde{\mathbf{X}} \mathbf{p}_m}{\mathbf{p}_m^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{p}_m} \mathbf{p}_m.$$

Step 3: Increase iteration index m by one and go back to Step 2.

The resulting function estimator in iteration m is then

$$\hat{F}^{(m)}(x) = \bar{y} + \sum_{i=1}^{m} \hat{\beta}_i^{(m)} \hat{f}^{(i)}(x)$$

and can be evaluated at any x.

The main difference to the linear CDBoost is that we do not compute the negative gradient, but fit the learner to the current residuals. The predicted values give $\tilde{\mathbf{x}}_m$. Loosely speaking, the "gradient approximation" corresponding to (1) is $\mathbf{e}_m \in \mathbb{R}^m$, because $\tilde{\mathbf{x}}_m$ is already an approximate functional gradient (fitting the residuals best); note the correspondence with the function and gradient version of L_2Boosting in section 3. Having a gradient approximation, we continue as in the case of linear regression: we compute $\mathbf{p}_m \in \mathbb{R}^m$ and optimize in that direction. This means that we adjust the coefficients of the fitted learners already included in the model.

The addition of shrinkage is implemented as follows: we recall that in the linear regression case we restart when a variable is chosen a second time. For arbitrary learners this would happen when obtaining the identical fitted values as in an earlier step. It can be difficult to determine numerically whether we have the same fitted values as before. We therefore propose restarting when the absolute correlation between $\tilde{\mathbf{x}}_m$ and a $\tilde{\mathbf{x}}_i,\ i < m$ exceeds a certain threshold. This threshold should not be too low, because we only want to restart when it is really necessary. We found that 0.999 works fine for sample sizes between 50 and 500.

9 Simulations with trees and splines

Using a simulated model we compare the (general) CDBoost with boosting. We choose $n = 100,\ d = 10,\ d_{\text{eff}} = 5$ and the covariates as in section 7.1. The true function F is

$$F(x) = c_1 x_1 + c_2 x_2 + 9 c_3\, \varphi(x_3) + 1.4 c_4\, \sin(2x_4) + 2.7 c_5 / (1 + \exp(-3x_5))$$

where φ is the density of the standard normal distribution and $c_1, \ldots, c_5 \sim$ log-Uniform$[-0.35, 0.35]$. The model for the response is $y = F(x) + \epsilon$ with $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.

		stnr = 9			*stnr* = 4			*stnr* = 1		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	7.38			11.15			26.5		
	$\nu = 0.7$	5.74			9.05			21.5		
	$\nu = 0.5$	5.57			8.73			20.5		
	$\nu = 0.3$	**5.28**			8.27			20.1		
	$\nu = 0.1$	5.36			**8.22**			**19.1**		
	$\nu = 0.03$	5.59			8.60			19.9		
	validated	5.01			7.93			19.0		
Boo	$\nu = 1$	6.21	−15.9	7e−09	9.90	−11.3	5e−04	23.5	−11.5	3e−05
	$\nu = 0.7$	5.90	2.7	4e−01	9.23	2.0	7e−01	21.7	1.0	1e−00
	$\nu = 0.5$	**5.61**	0.6	7e−01	8.74	0.1	6e−01	20.5	−0.1	9e−01
	$\nu = 0.3$	5.62	6.4	2e−02	**8.59**	3.9	2e−02	20.1	0.2	9e−01
	$\nu = 0.1$	5.64	5.3	1e−04	8.68	5.6	4e−06	20.0	4.4	1e−02
	$\nu = 0.03$	5.65	1.2	1e−02	8.71	1.2	2e−04	**20.0**	0.2	7e−02
	validated	5.49	9.5	2e−04	8.48	7.0	2e−04	20.2	6.3	2e−02

Table 5: Comparison of CDBoost (CDB) and boosting (Boo) with componentwise cubic smoothing splines for various signal-to-noise ratios (*stnr*) and shrinkage factor ν (and validated over $\nu = 0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03$). Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and *stnr* is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests.

Table 5 shows the results using componentwise cubic smoothing splines with 3 degrees of freedom as base learner. This means that in each iteration we fit a smoothing spline with 3 degrees of freedom for each predictor individually, and then select the one which reduces the residual sum of squares most. CDBoost and Boosting both perform better with shrinkage. It is especially CDBoost which benefits from the validation over ν with the candidate values $0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03$, and it is then clearly better than boosting.

		$stnr = 9$			$stnr = 4$			$stnr = 1$		
		test e	loss	p-value	test e	loss	p-value	test e	loss	p-value
CDB	$\nu = 1$	17.7			25.5			49.0		
	$\nu = 0.7$	11.6			15.8			30.9		
	$\nu = 0.5$	10.7			14.5			28.0		
	$\nu = 0.3$	**10.2**			14.2			26.5		
	$\nu = 0.1$	10.2			**14.1**			26.3		
	$\nu = 0.03$	10.3			14.1			**26.3**		
	$\nu = 0.01$	10.3			14.1			26.4		
	validated	10.3			14.2			26.9		
Boo	$\nu = 1$	20.0	13.3	5e−09	27.6	8.3	1e−04	53.2	8.6	1e−04
	$\nu = 0.7$	15.8	36.1	0e+00	21.7	36.8	0e+00	39.4	27.5	8e−13
	$\nu = 0.5$	13.4	25.8	2e−16	17.6	21.5	2e−16	31.5	12.7	6e−08
	$\nu = 0.3$	11.3	10.1	2e−11	14.8	4.3	7e−05	27.1	2.2	5e−02
	$\nu = 0.1$	10.5	2.2	2e−02	14.0	−0.6	7e−01	26.4	0.5	6e−01
	$\nu = 0.03$	10.4	0.7	7e−01	**14.0**	−1.1	5e−01	**26.4**	0.4	6e−02
	$\nu = 0.01$	**10.4**	0.4	1e−00	14.0	−0.9	9e−01	26.4	0.0	3e−01
	validated	10.6	2.9	3e−02	14.4	1.4	3e−01	27.7	2.8	2e−01

Table 6: Comparison of conjugate direction boosting (CDB) and boosting (Boo) with stumps for various signal-to-noise ratios (*stnr*) and shrinkage factor ν (and validated over $\nu = 0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03,\ 0.01$). Given is the (standardized) mean of the test error (test e) over 100 simulations. The best value for each method and*stnr* is in bold face. The loss of boosting compared to CDBoost with the same ν is given in % and the p-values are from paired sample Wilcoxon tests.

Table 6 shows the results using stumps (trees with two terminal nodes) as base learner. Boosting, once more, needs small ν for best performance and also CDBoost gives better results for smaller ν. Both methods perform almost equally well and the validation over ν does not pay off, because the shrinkage factors 0.7 and 0.5 lead to bad models.

Because the true underlying model is continuous, it is no surprise that both methods are clearly worse with stumps than with splines as base learner.

10 Real data examples

Finally, we compare CDBoost and boosting using two real datasets.

10.1 Los Angeles ozone data

The Los Angeles ozone dataset (Breiman and Friedman 1985) consists of $n = 330$ complete observations of $d = 8$ explanatory variables. We randomly split it into a training set of size 220 and a test set of 110. We use five-fold cross-validation to estimate the number of iterations/steps. The splitting is repeated 100 times and the test errors are averaged.

We only fit main effects models: linear models with all covariates ($d = 8$) and linear models with all covariates and the squared covariates ($d = 16$). In addition, we apply componentwise smoothing splines with 3 degrees of freedom and stumps as learners. The results are contained in table 7 (abbreviations of methods as in table 2).

	Shrinkage factor ν							
	1	0.7	0.5	0.3	0.1	0.03	0.01	validated
CDB linear	21.1	21.0	**21.0**	21.0	21.0	21.0	21.0	21.0
Boo linear	21.1	21.0	**20.9**	20.9	21.0	21.0	21.0	21.0
FVS linear	21.1							
LAR linear	21.0							
Las linear	21.0							
For linear	21.1							
CDB quadratic	18.9	18.6	**18.5**	18.5	18.5	18.5	18.5	18.5
Boo quadratic	18.9	18.5	**18.4**	18.4	18.5	18.5	18.5	18.5
FVS quadratic	19.0							
LAR quadratic	18.6							
Las quadratic	18.6							
For quadratic	18.5							
CDB splines	18.7	18.6	18.1	18.0	17.9	**17.9**	17.9	18.2
Boo splines	18.3	18.1	17.9	17.9	**17.9**	17.9	17.9	17.9
CDB stumps	22.6	19.9	19.2	18.9	18.9	**18.8**	18.8	19.2
Boo stumps	22.5	21.6	20.5	19.3	18.9	**18.9**	18.9	19.1

Table 7: Comparison of different methods for the ozone dataset. Given is the mean of the test error over 100 random splits. The best values for CDBoost and boosting are in bold face. CDBoost and boosting are also validated over the shrinkage factor ν with candidate values $0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03,\ 0.01$.

There are differences between the learners but almost none between the methods. In contrast to the simulated models, CDBoost does not deteriorate when ν becomes very small and the validation over ν does not pay off. This is because we work with only one dataset and therefore there should be a single ν which is adequate for all random splits. In addition, the validation would give better results when we exclude the shrinkage factor 0.7.

10.2 Leukemia data

The Leukemia dataset (Golub, Slonim, Tamayo, Huard, Gassenbeek, Mesirov, Coller, Loh, Downing, Caligiuri, Bloomfield, and Lander 1999) is from a microarray experiment with 72 samples and 3571 genes. Although it is a binary classification problem, we treat it as a regression problem with outcome 0 and 1 and L_2-loss (see also Zou and Hastie 2005 in their analysis of microarray data). We use the fitted values to classify the samples with cut-point 1/2 and compute the misclassification rate. Again, we average again over 100 random splits with 50 training cases and 22 test cases. The number of iterations is estimated with five-fold cross-validation. Table 8 shows the results.

In this example, CDBoost performs better than the other methods. The validation over ν does again not pay off (the results would again be better when we exclude the shrinkage factors 0.7 and 0.5 from the validation).

	Shrinkage factor ν								
	1	0.7	0.5	0.3	0.1	0.03	0.01	0.003	validated
CDB	8.91%	5.05%	4.41%	4.64%	4.32%	4.27%	**4.14%**	4.32%	4.55%
Boo	9.23%	7.82%	6.59%	5.23%	5.45%	5.27%	**5.18%**	5.23%	5.73%
FVS	8.50%								
LAR	5.32%								
Las	5.23%								
For	5.23%								

Table 8: Comparison of different methods for the leukemia dataset. Given is the mean of the test error (misclassification rate) over 100 random splits. The best values for CDBoost and boosting are in bold face. CDBoost and boosting are also validated over the shrinkage factor ν with candidate values $0.7,\ 0.5,\ 0.3,\ 0.1,\ 0.03,\ 0.01,\ 0.003$.

11 Discussion

We propose a conjugate direction boosting method (CDBoost) for linear and more general regression with potentially very many covariates. For linear regression, our CDBoost is a special version of forward stepwise variable selection (see Proposition 1): in high-dimensional settings with very many covariates, it is computationally much faster than traditional forward stepwise variable selection (see Proposition 2).

The concept of conjugate directions allows the inclusion of shrinkage or small step sizes, analogously to boosting. Shrinkage can yield substantial or even dramatic improvements of predictive performance; the same is true for boosting. But the role of weak or no shrinkage has a more pronounced interpretation than in boosting, as discussed in section 5.2. We also discuss in section 6 that CDBoost with infinitesimally small shrinkage yields solutions which are very similar to boosting, Lasso and other versions of least angle regression (LARS). Thus, varying the step size makes CDBoost surprisingly flexible ranging from the Lasso, LARS (ℓ_1-penalty methods) and boosting to forward stepwise variable selection (some sort of ℓ_0-penalisation). In some cases, e.g. in our simulation model 2 in section 7.2, the shrinkage factor ν should not be too small for CDBoost, indicating that we should move a little more towards least squares fitting in sub-models.

Our CDBoost method has the potential to outperform boosting, LARS and the Lasso. It works especially well in complicated settings with correlated covariates where it is not obvious how much a covariate contributes to the response. The gain of using CDBoost is also more often pronounced in sparse cases with high signal to noise ratio and few effective covariates (because CDBoost is flexible enough to cover the whole range from LARS and boosting to forward stepwise variable selection; and for cases with large signal to noise ratios and few effective covariates, a solution closer to forward stepwise variable selection is expected to be good).

CDBoost, boosting and the three LARS methods perform almost equally well for real datasets with only a few covariates. It can, however, be difficult to see clear differences in real data examples because the irreducible error ϵ is also contained in the test error. For the high-dimensional leukemia data though, CDBoost performs better than the other methods.

Finally, the concept of conjugate directions in CDBoost allows for a direct generalization to non-parametric fitting methods. Such an extension to the non-parametric setting is also possible with boosting but not with LARS. One drawback is that CDBoost is often computationally more expensive and less generic than boosting. For example, it is unclear how to modify CDBoost for classification using reweighted least squares since we would need to combine reweighting with conjugacy.

Acknowledgements: We thank two referees for their constructive comments which clearly improved the paper.

References

Breiman, L. (1998). Arcing classifier (Pkg: p801-849), *The Annals of Statistics* **26**(3): 801–824.

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

Breiman, L. and Friedman, J. H. (1985). Estimating optimal transformations for multiple regression and correlation (C/R: p598-619), *Journal of the American Statistical Association* **80**: 580–598.

Bühlmann, P. and Yu, B. (2003). Boosting with the L2-loss: Regression and classification, *Journal of the American Statistical Association* **98**: 324–339.

Bühlmann, P. and Yu, B. (2005). Boosting, model selection, lasso and nonnegative garrote, *Technical Report 127*, Seminar für Statistik ETH Zürich.

Duffy, N. and Helmbold, D. (2000). Leaveraging for regression, *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory.*

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

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

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.

Golub, T., Slonim, D., Tamayo, P., Huard, C., Gassenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J., Caligiuri, M., Bloomfield, C., and Lander, E. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science* **286**: 531–537.

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

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging, *in* S. Mendelson and A. Smola (eds), *Advanced Lectures on Machine Learning*, Springer, pp. 119–184.

Miller, A. (2002). *Subset selection in regression*, Chapman & Hall.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, Springer Series in Operations Research, Springer-Verlag, NY.

Phatak, A. and de Hoog, F. (2002). Exploiting the connection between pls, lanczos methods and conjugate gradients: alternative proofs of some properties of pls, *Journal of Chemometrics* **16**: 361–367.

Rätsch, G., Demiriz, A. and Bennett, K. (2002). Sparse regression ensembles in infinite and finite hypothesis spaces, *Machine Learning* **48**: 193–221.

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

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

Temlyakov, V. (2000). Weak greedy algorithms, *Adv. Comp. Math.* **12**: 213–227.

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

Zemel, R. and Pitassi, T. (2001). A gradient-based boosting algorithm for regression problems, *NIPS-13: Advances in Neural Information Processing Systems, 13*, MIT Press, Cambridge, MA.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net, *Journal of the Royal Statistical Society, Series B, Statistical Methodology* **67**: 301–320.