Conjugate direction boosting for regression

June 2, 2004

Abstract

Boosting in the context of linear regression has gained additional attraction by the invention of least angle regression (LARS), where the connection between the lasso and forward stagewise fitting (boosting) was established. Earlier it has been found that boosting is a functional gradient optimization. Instead of the gradient method, we propose a conjugate direction method to minimize the residual sum of squares as a function of the regression coefficients. The result is a fast forward variable selection algorithm (CDBoost).

The addition of shrinkage (small step-sizes) is only possible with an adjustment in form of a restart condition. For infinitesimal shrinkage this leads to an algorithm between forward stagewise fitting (boosting) and least angle regression. In particular, it also employs variable selection.

While least angle regression seems to be restricted to linear regression, we can easily generalize our approach to arbitrary learners (fitting methods) like trees or splines, as for boosting. The learner is used to produce vectors of fitted values which can then be linearly fitted by CDBoost.

The different methods are compared on simulated and real datasets. CDBoost achieves the best predictions mainly in complicated settings with correlated covariates, where it is difficult to assign how much a covariate contributes to the response. The gain of CDBoost over boosting is especially high in low and mid noise problems.

Key words: conjugate direction optimization, high-dimensional linear regression, least angle regression, non-parametric regression.

1 Introduction

Boosting is one of the most successful machine learning ideas introduced in the last decade (Schapire 1990, Freund 1995, Freund and Schapire 1996). It combines the output of many weak (or base) learners to a strong committee. The original idea was to use the output from the weak learner to weight the data. In the next iteration, the weak learner is fitted to the weighted data and so on. Later it has been found that boosting is a functional gradient descent method (Breiman 1999, Friedman 2001). Thus, boosting exhibits a connection between statistics and numerical minimization. In numerics, there are faster and better optimization methods than gradient descent. Examples are conjugate gradient or conjugate direction optimization, which usually lead to faster convergence. In statistics, fast convergence on the training set is nice to have, but the main goal of a fitting method is to achieve good out-of-sample performance. It is not obvious whether these two aims go together or not.

Boosting, when viewed as a functional gradient descent, tries to optimize in the direction of the negative gradient of a loss function. Because each update must be a base learner,

this can only be done approximately. Because of this approximation, it is unclear whether and how a more sophisticated technique like conjugate gradient works.

In a first part, we propose a conjugate direction method for boosting (CDBoost) in the context of linear models with the squared error loss (L_2-loss). This method yields sparse estimates and employs variable selection. We compare our method to boosting, forward variable selection, lasso (Tibshirani 1996) and least angle regression (Efron, Hastie, Johnstone and Tibshirani 2004). CDBoost outperforms the other methods mainly in complicated settings, where it is not obvious how much a covariate contributes to the response.

In a second part, we generalize CDBoost to arbitrary learners like trees and smoothing splines. This generalization, while also being feasible for boosting, seems not to be easily possible for lasso and least angle regression.

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:

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

Parameter estimation is most often done by least squares, minimizing the loss function

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

Assuming $d < n$ 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 Forward variable selection

For $d > n$, the matrix $(\mathbf{X}^T\mathbf{X})$ becomes singular and the least squares solution is not unique. For $d < n$ fairly large, the least squares solution will often over-fit the data. In both cases we need another strategy.

One way is forward variable selection which is also computationally feasible if d is very large. We start with the intercept or empty model and sequentially add the covariate which most reduces the loss function L. We stop if no significant improvement can be achieved anymore. Forward variable selection is a very greedy method. Covariates which are highly correlated with variables already included in the model have a small chance to be chosen.

3 Boosting with L_2-loss

Boosting is a very general, iterative method. It uses a simple fitting method, called the weak or base learner. Boosting then combines many outputs of the simple method to a strong committee.

The general regression problem with L_2-loss is to estimate a function $F : \mathbb{R}^d \to \mathbb{R}$ minimizing the expected loss $\mathbf{E}\left[(Y - \hat{F}(X))^2/2\right]$. Boosting builds the function $\hat{F}$ step by step: in each move a base learner function estimate $h(x, \hat{\theta})$ is added. Due to the L_2-loss, boosting is especially simple and amounts to iterative fitting of residuals (Friedman 2001, Bühlmann and Yu 2003).

Boosting algorithm with L_2-loss :

Step 1 (initialization): $\hat{F}^{(0)}(.) \equiv \bar{\mathbf{y}}$ and $m = 1$.

Step 2: Compute current residuals $r_i = y_i - \hat{F}^{(m-1)}(x_i)$ $(i = 1, \ldots, n)$ and fit the base learner to them using the predictor variables. The fit is denoted by $\hat{f}^{(m)}(x) = h(x, \hat{\theta}^{(m)})$, based on $\{(x_i, r_i); i = 1, \ldots, n\}$. Update $\hat{F}^{(m)}(.) = \hat{F}^{(m-1)}(.) + \hat{f}^{(m)}(.)$.

Step 3 (iteration): 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.

3.1 Shrinkage or small step size

Boosting can often be improved by shrinkage: in each boosting step, only a small fraction ν (for example 0.1) of the fitted base learner is added. We can modify *Step 2* of the boosting algorithm to $\hat{F}^{(m)}(.) = \hat{F}^{(m-1)}(.) + \nu \cdot \hat{f}^{(m)}(.)$.

3.2 Boosting with componentwise linear least squares

We consider now boosting for the linear regression problem. Our base learner is a linear regression with only the one selected covariate which reduces the loss L most: $h(x, \hat{\theta}) = \hat{\beta}_{\hat{k}} x_{\hat{k}}$ with $\hat{\theta} = (\hat{k}, \hat{\beta}_{\hat{k}})$ the selected covariate and estimated coefficient, where $\hat{\beta}_k = \mathbf{x}_k^T \mathbf{r} / \mathbf{x}_k^T \mathbf{x}_k$, $\hat{k} = \arg\min_k \frac{1}{2}\|\mathbf{r} - \hat{\beta}_k \mathbf{x}_k\|^2$ and $\mathbf{r}$ denoting the current residuals.

The first iteration is the same as forward variable selection. But the further iterations are not. The main difference to forward variable selection is that we do not use ordinary least squares for the coefficients of the variables already included in the model. In particular, it is possible with boosting to choose a covariate again. This gives us the chance to adjust a coefficient which was badly estimated in an earlier step.

3.3 Gradient descent

Now we take a look at linear regression from another point of view. We can minimize the loss function

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

with the gradient method. For a given β, the negative gradient

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

gives the direction in the β-space in which L decreases most rapidly. It seems obvious to optimize in that direction by performing a one dimensional minimization, a so called line search. This yields a new β vector and we can repeat the procedure.

Now we come back to boosting which can be seen as a functional gradient descent method (Breiman 1999, Friedman 2001), where the loss L is viewed as a function of $F(x)$. Notice

that we use a slightly different approach for the linear regression problem because we work entirely in the β-space and take L as a function of β. But the main idea remains the same: we perform a line search in the direction, which is most parallel to the negative gradient and which is a base learner.

When using componentwise linear least squares as base learner (linear regression with only one selected covariate) we update only one component of $\hat{\beta}$ in each step. So we cannot take a step in the negative gradient direction, because this usually changes all coefficients. So 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 $\hat{k}$. We shall call $\mathbf{e}_{\hat{k}}$ gradient approximation and $\hat{k}$ is just the component of the negative gradient with highest absolute value. Then we perform a step in the direction $\mathbf{e}_{\hat{k}}$.

We have to be careful here: $\mathbf{e}_{\hat{k}}$ is the coordinate direction with steepest descent. But there can be another coordinate direction $\mathbf{e}_{\hat{j}}$ which leads to a smaller minimum than $\mathbf{e}_{\hat{k}}$ when performing the line searches. And that's what we really want: we want to choose the covariate which reduces L most and not the coordinate direction with steepest descent. Roughly speaking: a long flat descent is better than a short steep descent. Obviously this depends on the scales of the covariates. The solution is to standardize the covariates to have unit length ($\mathbf{x}_i^T \mathbf{x}_i = 1$). Then the "steepest descent coordinate" is also the "smallest minimum coordinate". This can be seen as follows:

For each covariate we compute the reduction of L that we can achieve by updating its coefficient. The best update for component $k \in \{1, \ldots, d\}$ is

$$\Delta\hat{\beta}_k = \frac{\mathbf{r}^T \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{x}_k} = \mathbf{r}^T \mathbf{x}_k$$

with $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\beta}$ denoting the residuals. This update reduces L by

$$\begin{aligned} \Delta L &= \frac{1}{2}\mathbf{r}^T\mathbf{r} - \frac{1}{2}\Big(\mathbf{r} - \mathbf{x}_k(\mathbf{r}^T\mathbf{x}_k)\Big)^T\Big(\mathbf{r} - \mathbf{x}_k(\mathbf{r}^T\mathbf{x}_k)\Big) \\ &= \mathbf{r}^T\mathbf{x}_k(\mathbf{r}^T\mathbf{x}_k) - \frac{1}{2}(\mathbf{r}^T\mathbf{x}_k)\mathbf{x}_k^T\mathbf{x}_k(\mathbf{r}^T\mathbf{x}_k) = \frac{1}{2}(\mathbf{r}^T\mathbf{x}_k)^2. \end{aligned}$$

On the other hand the component k of the negative gradient is $\mathbf{x}_k^T\mathbf{r}$. Thus, we see that the component with highest absolute gradient value is also the component which can reduce the loss function L most.

For convenience we center all the covariates and the response to mean 0. So we do not have to care about an intercept. This gives the following algorithm (notice again that we work with $\hat{\beta}^{(m)}$ and not with $\hat{F}^{(m)}(x) = x^T\hat{\beta}^{(m)}$):

Boosting algorithm with componentwise linear least squares:

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

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

and determine the component with highest absolute value, say component $\hat{k}^{(m)}$.
Update $\hat{\beta}$: component $\hat{k}^{(m)}$ changes to

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

and all the other components remain unchanged.

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

For $\nu = 1$, this algorithm is known as matching pursuit in signal processing (Mallat and Zhang 1993).

4 Conjugate boosting

4.1 Conjugate direction and gradient optimization

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

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

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 steps by minimizing along the individual directions in a conjugate set. A conjugate direction method takes an arbitrary set of d conjugate directions and minimizes L along them. In contrast to the gradient method we reach the minimum after d steps.

The question is now how to find a set of conjugate directions. The canonical *conjugate gradient method* does it in a very efficient way 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).

4.2 Linear conjugate direction boosting (CDBoost)

We develop now some conjugate boosting method for the linear regression problem. In our notation we have $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and $\mathbf{b}^T = \mathbf{y}^T\mathbf{X}$. If $n < d$, the matrix $\mathbf{A}$ is not positive definite. This does not bother because we actually only use sub-matrices of $\mathbf{X}$ so that the corresponding $\mathbf{A}$ is positive definite.

Similarly to boosting, we don't want to minimize along the negative gradient (first direction of the conjugate gradient method) because this would change all components of $\hat{\beta}$ to non-zero values. In fact, it can be shown that the *conjugate gradient method* is the same as *Partial Least Squares*. But here we want to be able to do some kind of variable selection, where in each step only one component of $\hat{\beta}$ should change from zero to non-zero.

This is still possible within the framework of a *conjugate direction method*. The first step is identical to gradient boosting: we look for the component of the negative gradient with highest absolute value, say component $\hat{k}^{(1)}$. The vector $\mathbf{e}_{\hat{k}^{(1)}}$ is then the best approximation to the negative gradient by a unit-coordinate vector, in terms of having maximal absolute inner product. So we optimize in that direction.

In the following steps we have to determine a direction which is conjugate to all previous directions. We compute again the negative gradient and its approximation by a unit-coordinate vector $\mathbf{e}_{\hat{k}^{(m)}}$. The new direction 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 it 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 conjugate property 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

Linear conjugate direction boosting (CDBoost) algorithm with L_2-loss:

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 $m = 1$. Set $\mathbf{A} = \mathbf{X}^T \mathbf{X}$.

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, say component $\hat{k}^{(m)}$. Define the gradient approximation by $\mathbf{e}_{\hat{k}^{(m)}}$.

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

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

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

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

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

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

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

We have to store all the previous directions. So we have lost the simplicity of the conjugate gradient method. On the other hand we have an algorithm which performs variable selection because in each step only one coefficient of $\hat{\beta}^{(m)}$ changes from zero to non-zero.

The advantage of the conjugate direction method is its speed. For a problem with d covariates it needs at most d steps to end up in the least squares solution and it can be much faster than boosting.

For d very large, the computation of $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ becomes expensive. But we don't have to compute $\mathbf{A}$ in advance. We can do it during the iteration process, where we only compute the elements of $\mathbf{A}$ which are necessary. Therefore it is possible to use CDBoost for $d \gg n$ or with an overcomplete dictionary of basis functions, since we will do at most n iterations.

4.2.1 A fast forward variable selection algorithm

Now we take a closer look at the iteration process. After $m < d$ iterations, our conjugate direction method has included m covariates in the model and the actual solution is the least squares fit for these m covariates. This can be seen as follows: 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 used a conjugate direction method. That means the conjugate direction method moves from least squares solution to least squares solution (in sub-models) and therefore, each covariate can only be chosen once. At first sight this seems to be the same as the classical forward variable selection. But there is a difference in how the next variable is selected.

The conjugate direction method selects the covariate which would most improve the fit without adjusting the coefficients of the other variables. After the selection, we update $\hat{\beta}$ by adjusting all the non-zero coefficients. On the other hand, forward variable selection looks for the best model consisting of all the old covariates and a new one. To determine the improvement when adding a variable, it computes the entire new model and compares it with the old one.

Thus we have found a much faster version of a forward variable selection algorithm with very similar performance as the classical one. Moreover, CDBoost seems to have better prediction accuracy than matching pursuit (or boosting with $\nu = 1$) for low or mid-noise problems. This may be of special interest for many signal processing problems such as imaging. However, as mentioned above, boosting often performs better with shrinkage. So it's obvious to include shrinkage for CDBoost too.

4.3 Shrinkage

Shrinkage means to not fully update $\hat{\beta}$. We only add a fraction ν of the optimal $\Delta\hat{\beta}$. But here arises a new problem: without shrinkage, each covariate can only be chosen once. This is no longer true with shrinkage. The $\hat{\beta}^{(m)}$ are no longer least squares solutions. It is possible that we achieve the best improvement with changing the coefficient of a covariate already included in the model. When a variable is chosen the second time, a normal update as in formula (1) is not possible anymore. The linear combination between the gradient approximation and all the previous directions would be the zero vector, because there are only m conjugate directions in a m-dimensional space. A possible remedy is discussed next.

4.3.1 Conjugate direction boosting with restart

An easy and effective way out of the problem discussed above is to restart the algorithm when a variable is chosen the second time. We take the actual $\hat{\beta}^{(m)}$ as the new starting $\hat{\beta}$, delete all the conjugate directions $\mathbf{p}_i$ and start again. In the algorithm given below we actually don't delete the directions $\mathbf{p}_i$, but sum only over the directions since the last restart (given by s). The final algorithm is:

Linear CDBoost algorithm with L_2-loss and shrinkage:

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 $s = 1$. Set $\mathbf{A} = \mathbf{X}^T\mathbf{X}$.

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, say component $\hat{k}^{(m)}$. Define the gradient approximation by $\mathbf{e}_{\hat{k}^{(m)}}$.

If $m > 1$ and $\hat{k}^{(m)} \in \{\hat{k}^{(s)}, \ldots, \hat{k}^{(m-1)}\}$ 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

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

  and compute the new direction

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

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

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

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

We 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 therefore is chosen first. 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 take variable 1 again followed by a step in direction $\mathbf{p}_1$. This scenario repeats until $\hat{\beta} = (1.03, 0, 0)^T$, 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 (directions $\mathbf{p}_1$ and $\mathbf{p}_2$) we moved the fraction ν to the least squares solution of covariate 1 and 2. Before 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 1 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 repeats until $\hat{\beta} = (1.96, 0.93, 0)^T$, 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

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

The fact that the restart has always been caused by covariate 1 is not a coincidence, as described by the following result (also the repeating pattern of selected pairs and triples is a more general fact).

Proposition 1 *The linear 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 only take 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

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

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.

Proposition 1 follows from the fact above. Suppose the covariate with index $\hat{k}^{(1)}$ has highest absolute gradient value and is chosen at the beginning and CDBoost yields the sequence $\hat{k}^{(1)}, \ldots, \hat{k}^{(j)}$, where all $\hat{k}^{(i)}, i = 1, \ldots, j$ are different. In these last j steps we moved the fraction ν to the least squares solution of these j covariates. It follows from the fact above 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 choose therefore either $\hat{k}^{(1)}$ (restart) or a completely new covariate (no restart necessary). $\square$

For higher d there will be less restarts, because we can choose among a greater set of covariates. Only when n and d both are large, CDBoost becomes slow. A possibility to speed up the computation is to force a restart after a certain number of iterations. This yields an algorithm between boosting and CDBoost. We don't pursue this possibility any further.

Without shrinkage we have a proper conjugate direction method and are much faster than boosting. This isn't true anymore with shrinkage.

5 Connections to LARS

Each iterative method generates a path from the intercept model to the full least squares solution ($n > d$) or to a perfect fit ($n \leq d$). The general question for the linear regression problem is which path to choose.

Efron et al. (2004) developed recently a new efficient algorithm called LAR(S) (least angle regression). It is a cautious version of forward variable selection, where the selected covariates build the so called active set. They call $\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta^{(m)})$ the vector of current correlations and we call it the negative gradient. A slight modification of LAR leads to the lasso 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 chapter we assume infinitesimal shrinkage for boosting and CDBoost. LAR and boosting yield the same path, when all the β-coefficients move monotonically away from zero. In this case also CDBoost leads to that path. For more general cases, turnarounds of coefficients (for example decreasing after increasing) are possible with LAR. Forward stagewise fitting behaves different in such a case: it drops that variable from the active set and leaves its coefficient unchanged in the next step. The CDBoost algorithm performs intermediate, depending on the turning coefficient. Two examples with 3 covariates demonstrate this behavior. The following tables show the traces (endpoints of linear pieces) of the β-coefficients from zero to the least squares solutions.

Example 1:

LAR/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2.0	0.0	0.0
4.9	2.9	0.0
1.8	10.2	6.0

Boosting/Stagewise

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
2.0	0.0	0.0
4.9	2.9	0.0
4.9	6.1	3.2
1.8	10.1	6.0

Example 2:

LAR

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7.4	0.0	0.0
12.8	5.4	0.0
18.5	2.8	4.6

Boosting/Stagewise/CDBoost

$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$
7.4	0.0	0.0
12.8	5.4	0.0
15.2	5.4	2.4
18.5	2.8	4.6

LAR finds the least squares solution always after 3 fairly large steps (in principle we can also evaluate intermediate models between two steps). For forward stagewise fitting immediate turnarounds of coefficients are not possible. Therefore covariate 1 in example 1 and covariate 2 in example 2 is dropped after step 2, its coefficient rests and forward stagewise fitting needs one more step.

CDBoost behaves like LAR 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 LAR. 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:

$1, 1, \ldots, 1,\ 1, 2, 1, 2, \ldots, 1, 2,\ 1, 3, 1, 3, \ldots, 1, 3,\ 1, 3, 2, 1, 3, 2, \ldots$.

6 Simulations with linear regression

The performance of the CDBoost method is assessed first by a simulation study. The advantage of a simulation is that we know the true model and therefore can easily evaluate the performance of the different methods. The CDBoost method is compared to boosting, forward variable selection and the three LARS methods (least angle regression, lasso and forward stagewise fitting).

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 model. The training error, which is the loss L on the training data, decreases with each boosting iteration. When there are more covariates than observations, the training error will finally reach zero giving a perfect fit. The validation set is used to stop the iteration process. We use the model fitted to the training set to predict on the validation set. The validation error is then the prediction error on the validation set. The number of iterations for the final model is chosen to minimize the validation error. Finally we have the test set to measure the prediction error on the test set:

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

where $F(x) = x^T\beta$ is the true underlying regression function. Both the validation and the test set error decrease during the first iterations and later increase due to over-fitting.

Boosting and CDBoost both have a second tuning parameter, the shrinkage factor ν, which is typically less crucial than the number of iterations. To compare CDBoost to boosting we can fix ν and compare the methods. To compare CDBoost and boosting to forward variable selection and the three LARS methods, we have to choose a ν. We decided to validate over ν using a small set of different shrinkage factors. For each ν, the sequence of models was computed and the final model is the one with smallest validation error. Because $\nu = 1$ usually leads to bad models, we excluded this model sequence from the validation.

In practice, a validation over ν is hardly ever done and $\nu = 0.1$ is often reasonable. But here it is fairer than to choose the best ν after we tried different values. Furthermore these computations can easily be done in parallel if needed in practice.

Forward variable selection and the LARS methods use large steps and yield only a small set of models. We also used the validation set to choose the final model. For the LARS methods one could not only evaluate the models at the end of a step, but also intermediate ones. We don't give detailed results for that.

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

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

6.1.1 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 y 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. $\sim \mathcal{N}_d(0, I)$ 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 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. We set the first five components of β to values around five, more precisely, $\beta_1, \ldots, \beta_5$ i.i.d. $\sim \mathcal{N}(5, 1)$. The other five components are zero.

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

$$stnr = \frac{\text{Var}\left(F(X)\right)}{\text{Var}\left(\epsilon\right)}.$$

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

To compare the methods we take various signal-to-noise ratios and simulate 100 datasets from each setting. The performance is the mean over the 100 test errors resulting from the 100 simulated datasets.

We report on the test error in a standardized form. We divide it by the test error of the best constant (location) model. Each fitting method should be better than the best constant model and therefore achieve a value below 1. This standardized test error makes it possible to compare different settings. For convenience we still refer to it as test error.

6.1.2 Results for model 1

Table 1 shows the comparison of boosting and CDBoost. Given the same shrinkage, CDBoost performs most of the time better than boosting. The percentage gain increases with higher *stnr*.

The effect of shrinkage is substantial in this example. Without shrinkage both methods perform badly. Because the covariates with influence are often positively correlated and have a positive influence on y, the coefficient of the first selected variable is estimated too high. Although this coefficient is (CDBoost) or can be (boosting) adjusted in later steps, we often get a bad model. So it is not surprising that shrinkage improves the fits drastically. The choice of the ν is not so crucial, the important thing is the usage of some shrinkage.

		$stnr = 9$		$stnr = 4$		$stnr = 1$	
		test error	gain	test error	gain	test error	gain
boo	$\nu = 1$	0.0133		0.0291		0.0956	
	$\nu = 0.3$	0.0120		**0.0244**		0.0763	
	$\nu = 0.1$	0.0120		0.0244		0.0750	
	$\nu = 0.03$	**0.0120**		0.0244		**0.0746**	
	$\nu = 0.01$	0.0120		0.0244		0.0749	
cdb	$\nu = 1$	0.0120	10.1	0.0281	3.6	0.0985	−3.1
	$\nu = 0.3$	**0.0108**	9.8	**0.0224**	7.9	**0.0739**	3.2
	$\nu = 0.1$	0.0109	9.5	0.0227	7.0	0.0742	1.0
	$\nu = 0.03$	0.0109	8.8	0.0228	6.9	0.0753	−1.0
	$\nu = 0.01$	0.0109	9.3	0.0228	6.6	0.0747	0.2

Table 1: Comparison of boosting (boo) and conjugate direction boosting (cdb) for various signal-to-noise ratios and ν for simulation model 1. Given is the (standardized) mean of the test error over 100 simulations. The best value for each method and signal-to-noise ratio is in bold face. The gain of CDBoost over boosting with the same ν is given in %.

In terms of computational efficiency (not explicitly shown) CDBoost needs much less iterations than boosting without shrinkage. This is no longer true with shrinkage where CDBoost looses the fast convergence of the conjugate methods. For high *stnr* CDBoost has been found to need less iterations, while it was the other way around for small *stnr*.

	$stnr = 9$		$stnr = 4$		$stnr = 1$	
	test error	gain	test error	gain	test error	gain
boo	0.0116		0.0239		0.0748	
cdb	0.0106	8.8	0.0223	6.6	0.0735	1.8
fvs	0.0118	−1.3	0.0293	−22.5	0.0983	−31.3
lar	0.0113	3.2	0.0245	−2.4	0.0820	−9.6
las	0.0110	5.6	0.0238	0.5	0.0797	−6.5
for	0.0121	−3.9	0.0252	−5.8	0.0793	−5.9

Table 2: Comparison of boosting (boo), conjugate direction boosting (cdb), forward variable selection (fvs) and the three LARS methods (LAR, lasso, forward stagewise fitting) for simulation model 1. Given is the (standardized) mean of the test error over 100 simulations. Boosting and CDBoost are validated over the shrinkage factor ν taking the values $0.3, 0.1, 0.03, 0.01$. The gain over boosting is given in %.

Now we compare boosting and CDBoost (validated over $\nu = 0.3, 0.1, 0.03, 0.01$) to forward variable selection and the three LARS methods LAR, lasso and forward stagewise fitting. The results are given in table 2 and figure 1. Boosting and CDBoost give most of the time slightly better results when we validate over ν compared to a fixed ν.

CDBoost performs best. The gain compared to boosting increases for higher *stnr*. LAR and lasso perform also better for high *stnr*. Forward stagewise fitting is always worse than boosting. The LARS methods achieve better results in this example when we also evaluated intermediate models, but still worse than CDBoost.

Forward variable selection leads to the worst models. It would be the perfect method, if

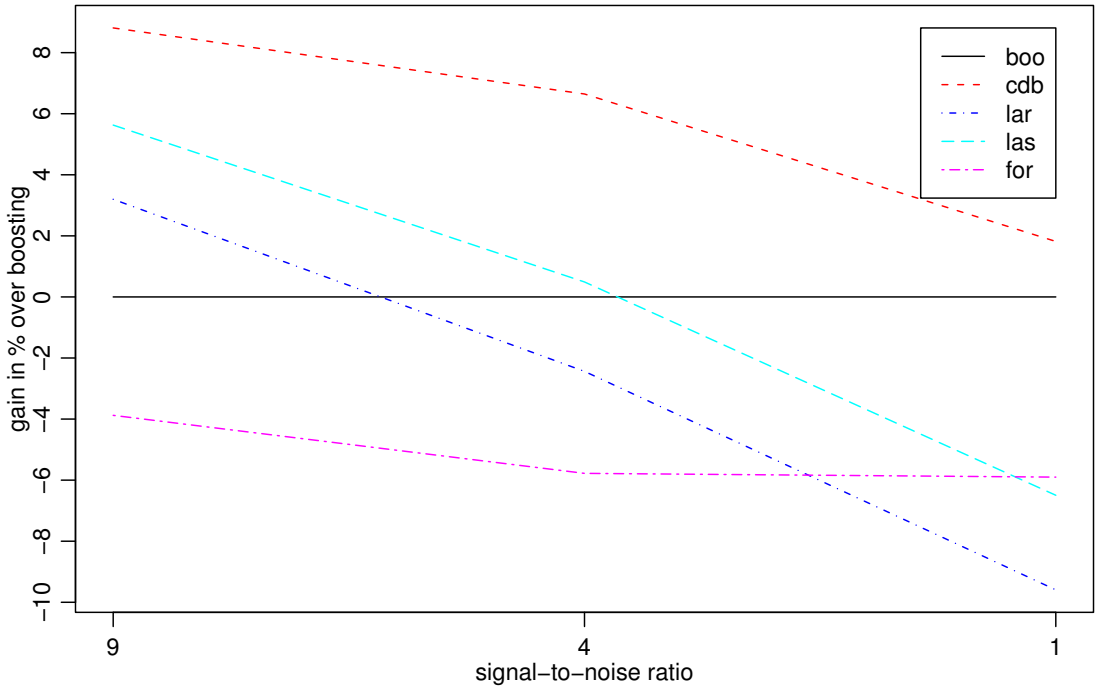

Figure 1: Percentage gain of the different methods (without forward variable selection) over boosting for signal-to-noise ratios equal to $9, 4, 1$ for simulation model 1.

it selected only the five covariates with influence. But the results show that it is far from doing that, even for $stnr = 9$. Its performance is similar to CDBoost without shrinkage, as claimed in section 4.2.1.

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

6.2.1 Setup for model 2

Now we 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 6.1.1 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 also use $stnr = 16$.

6.2.2 Results for model 2

		$stnr = 16$		$stnr = 9$		$stnr = 4$		$stnr = 1$	
		test error	gain	test error	gain	test error	gain	test error	gain
boo	$\nu = 1$	0.2799		0.3038		0.3853		0.9046	
	$\nu = 0.3$	0.1803		0.2452		0.3720		0.7398	
	$\nu = 0.1$	0.1660		**0.2294**		**0.3662**		**0.7345**	
	$\nu = 0.03$	0.1644		0.2296		0.3703		0.7409	
	$\nu = 0.01$	**0.1643**		0.2295		0.3711		0.7433	
cdb	$\nu = 1$	0.2375	15.1	0.2638	13.2	0.3646	5.4	0.9074	−0.3
	$\nu = 0.3$	**0.1525**	15.4	**0.2168**	11.6	**0.3450**	7.3	**0.7158**	3.2
	$\nu = 0.1$	0.1580	4.8	0.2243	2.2	0.3622	1.1	0.7312	0.5
	$\nu = 0.03$	0.1603	2.5	0.2281	0.7	0.3695	0.2	0.7397	0.2
	$\nu = 0.01$	0.1613	1.8	0.2295	−0.0	0.3715	−0.1	0.7433	0.0

Table 3: Comparison of boosting (boo) and conjugate direction boosting (cdb) for various signal-to-noise ratios and ν for simulation model 2. Given is the (standardized) mean of the test error over 100 simulations. The best value for each method and signal-to-noise ratio is in bold face. The gain of CDBoost over boosting with the same ν is given in %.

Table 3 shows the comparison of boosting and CDBoost. The results are similar to the one of model 1. CDBoost performs better than boosting, especially for high *stnr* and $\nu = 0.3$, which seems to be a good choice for CDBoost. In this model, ν shouldn't be too small.

	$stnr = 16$		$stnr = 9$		$stnr = 4$		$stnr = 1$	
	test error	gain	test error	gain	test error	gain	test error	gain
boo	0.1633		0.2260		0.3584		0.7305	
cdb	0.1515	7.2	0.2164	4.3	0.3452	3.7	0.7127	2.4
lar	0.1575	3.5	0.2286	−1.1	0.3758	−4.9	0.7657	−4.8
las	0.1566	4.1	0.2268	−0.3	0.3720	−3.8	0.7678	−5.1
for	0.1643	−0.6	0.2300	−1.8	0.3731	−4.1	0.7536	−3.2

Table 4: Comparison of boosting (boo), conjugate direction boosting (cdb) and the three LARS methods (LAR, lasso, forward stagewise fitting) for simulation model 2. Given is the mean of the test error over 100 simulations. Boosting and CDBoost are validated over the shrinkage factor ν taking the values $0.3, 0.1, 0.03, 0.01$. The gain over boosting is given in %.

In table 4 and figure 2 we compare boosting and CDBoost (validated over $\nu = 0.3, 0.1, 0.03, 0.01$) to the LARS methods. The results are again analogous to model 1.

CDBoost performs best and the gain compared to boosting increases for higher *stnr*. Lasso is a bit better than LAR and they both need a high signal-to-noise ratio to be better than boosting. Forward stagewise fitting is again always worse than boosting. In this example the LARS methods were not better when evaluating also intermediate models. So they are clearly worse than CDBoost.

We do without forward variable selection in this example but remind that CDBoost with $\nu = 1$ can be thought of a fast forward variable selection, as described in section 4.2.1.

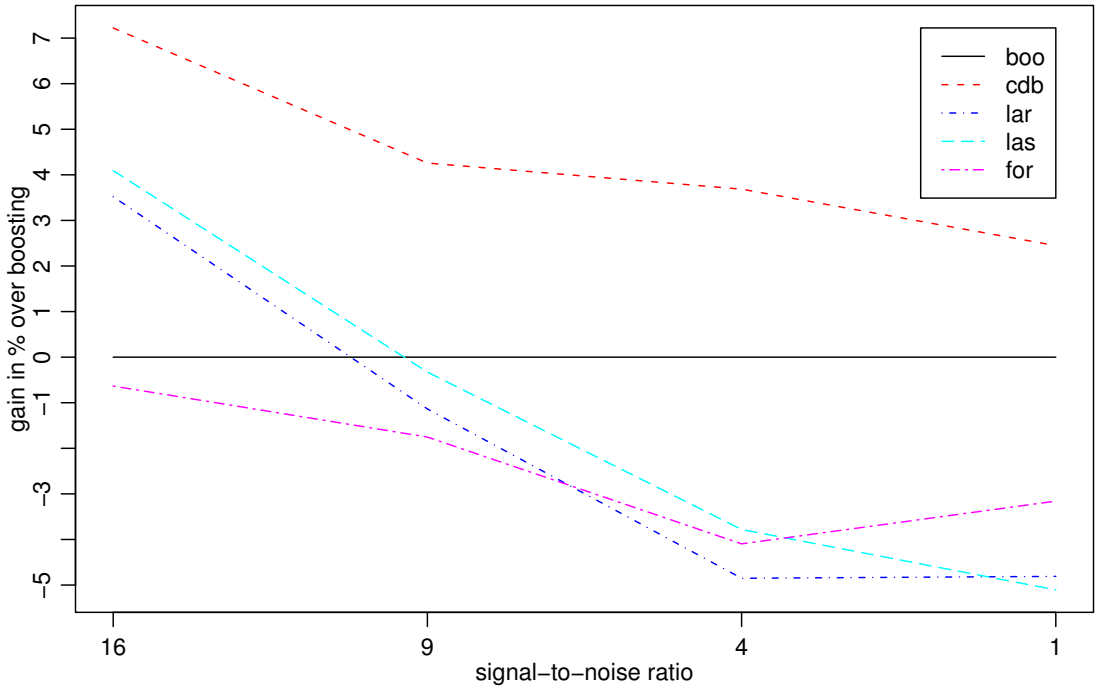

Figure 2: Percentage gain of the different methods over boosting for signal-to-noise ratios equal to $16, 9, 4, 1$ for simulation model 2.

7 Generalization to arbitrary learners

So far we only studied linear regression. Now we generalize our approach to any learner, for example trees or componentwise smoothing splines (Bühlmann and Yu 2003, see also section 8). In principle, we could use linear expansions in tree-type or B-spline basis functions, resulting in huge, highly overcomplete dictionaries. But we rather 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 predicted values (which can be viewed as an estimated basis function) and exactly that vector is taken as the next column of $\tilde{\mathbf{X}}$. Thus, $\tilde{\mathbf{X}}$ grows in each step by one column. The matrix $\tilde{\mathbf{X}}$ is then used for the linear CDBoost. The following algorithm makes the idea more 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 $\tilde{\mathbf{X}} = \text{null}$, $\hat{\beta}^{(0)} = \text{null}$ and $m = 1$.

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

Increase $\tilde{\mathbf{X}}$ by one column by adding $\hat{f}^{(m)}$. 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. Compute $\mathbf{A} = \tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$.

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 $\lambda's$):

$$\lambda_i = -\frac{\mathbf{p}_i^T \mathbf{A} \mathbf{e}_m}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i}.$$

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

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

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

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

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

The resulting function estimator in iteration m is then

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

Since $\hat{f}^{(i)} = \hat{f}^{(i)}(.)$ can be evaluated at any x, the resulting combined function estimator $\hat{F}$ inherits this property.

The main difference to the linear CDBoost is that we don't compute the negative gradient, but we fit the learner to the current residuals. The predicted values give $\tilde{\mathbf{x}}_m$. Loosely speaking the "gradient approximation" is $\mathbf{e}_m \in \mathbb{R}^m$, because we "chose" $\tilde{\mathbf{x}}_m$. After that we can continue as in the linear case: we compute $\mathbf{p}_m \in \mathbb{R}^m$ and optimize in that direction. That means, we adjust the coefficients of the fitted learners already included in the model.

The addition of shrinkage is implemented as follows. In the linear case we restart when a variable is chosen a second time. For arbitrary learners this would happen when we receive the identical fitted values as in an earlier step. It can be numerically hard to decide whether we have the same fitted values as before. So we propose to restart when the absolute correlation between $\tilde{\mathbf{x}}_m$ and a $\tilde{\mathbf{x}}_i$, $i < m$ exceeds a certain threshold. This threshold shouldn't 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.

8 Simulations with trees and splines

We compare boosting and CDBoost for a simulated model. We choose $n = 100$, $d = 10$, $d_{\text{eff}} = 5$ and the covariates as in section 6.1. The true function F is

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

where φ is the density of the standard normal distribution. $y = F(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. For every model simulation each of the five components is additionally randomly scaled with a factor uniformly distributed on $[0.7, 1.4]$.

Table 5 shows the result using componentwise cubic smoothing splines with 3 degrees of freedom as learner. This means we fit in each iteration a smoothing spline for each predictor individually and take the one which reduces residual sum of squares most. Boosting

		$stnr = 16$		$stnr = 9$		$stnr = 4$		$stnr = 1$	
		test error	gain	test error	gain	test error	gain	test error	gain
boo	$\nu = 1$	0.0466		0.0621		0.0990		0.2350	
	$\nu = 0.3$	**0.0425**		**0.0562**		**0.0859**		0.2013	
	$\nu = 0.1$	0.0428		0.0564		0.0868		0.1996	
	$\nu = 0.03$	0.0428		0.0565		0.0871		**0.1995**	
	validated	0.0423		0.0558		0.0851		0.1994	
cdb	$\nu = 1$	0.0549	−17.7	0.0738	−18.9	0.1115	−12.7	0.2654	−12.9
	$\nu = 0.3$	**0.0403**	5.1	**0.0528**	6.0	0.0827	3.8	0.2010	0.2
	$\nu = 0.1$	0.0415	3.1	0.0536	5.0	**0.0822**	5.3	**0.1912**	4.2
	$\nu = 0.03$	0.0426	0.5	0.0559	1.2	0.0860	1.2	0.1991	0.2
	validated	0.0392	7.2	0.0510	8.7	0.0798	6.3	0.1870	6.2

Table 5: Comparison of boosting (boo) and conjugate direction boosting (cdb) with componentwise cubic smoothing splines for various signal-to-noise ratios and ν (and validated over $\nu = 0.3,\ 0.1,\ 0.03$). Given is the (standardized) mean of the test error over 100 simulations. The best value for each method and signal-to-noise ratio is in bold face. The gain of CDBoost over boosting with the same ν is given in %.

and CDBoost both perform better with shrinkage. While boosting is not sensitive to changes of ν, CDBoost becomes worse when ν is too small. It performs best with $\nu = 0.3$ or $\nu = 0.1$ and is better than boosting. It is especially CDBoost which benefits from the validation over ν taking the values 0.3, 0.1, 0.03 and it is then clearly better than boosting.

		$stnr = 16$		$stnr = 9$		$stnr = 4$		$stnr = 1$	
		test error	gain	test error	gain	test error	gain	test error	gain
boo	$\nu = 1$	0.1645		0.2005		0.2764		0.5316	
	$\nu = 0.3$	0.0966		0.1126		0.1479		0.2711	
	$\nu = 0.1$	0.0884		0.1045		0.1400		0.2644	
	$\nu = 0.03$	0.0881		0.1041		**0.1397**		**0.2637**	
	$\nu = 0.01$	**0.0881**		**0.1038**		0.1399		0.2637	
	validated	0.0882		0.1042		0.1414		0.2677	
cdb	$\nu = 1$	0.1479	10.1	0.1769	11.8	0.2553	7.6	0.4897	7.9
	$\nu = 0.3$	0.0871	9.8	**0.1023**	9.2	0.1417	4.1	0.2652	2.2
	$\nu = 0.1$	**0.0869**	1.7	0.1023	2.2	**0.1408**	−0.6	0.2630	0.5
	$\nu = 0.03$	0.0871	1.2	0.1034	0.7	0.1412	−1.1	**0.2626**	0.4
	$\nu = 0.01$	0.0873	0.9	0.1034	0.4	0.1412	−0.9	0.2636	0.0
	validated	0.0863	2.1	0.1014	2.7	0.1413	0.1	0.2634	1.6

Table 6: Comparison of boosting (boo) and conjugate direction boosting (cdb) with stumps for various signal-to-noise ratios and ν (and validated over $\nu = 0.3,\ 0.1,\ 0.03,\ 0.01$). Given is the (standardized) mean of the test error over 100 simulations. The best value for each method and signal-to-noise ratio is in bold face. The gain of CDBoost to boosting with the same ν is given in %.

Table 6 shows the result using stumps (trees with two terminal nodes) as learners. CDBoost performs only slightly better than boosting. Boosting needs small ν for best performance, while CDBoost gives again best results for $\nu = 0.3$ or $\nu = 0.1$. Because the true

underlying model is continuous, it is no surprise that both methods are clearly worse with stumps than with splines.

9 Real data examples

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

9.1 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 small training set of size 100 and a test set of 230. For larger training sets, there is hardly any difference between the methods. 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 fit only main effects models: linear models with all covariates ($d = 8$) and linear models with all covariates and the squared covariates ($d = 16$). Additionally we apply componentwise smoothing splines with 3 degrees of freedom and stumps as learners. The results are contained in Table 7. As a comparison we also use MARS as it is implemented in R (interaction degree 1, no shrinkage).

	Shrinkage					
	1	0.3	0.1	0.03	0.01	validated
boo linear	22.50	21.82	**21.81**	21.82	21.83	21.82
cdb linear	22.31	21.92	21.86	**21.84**	21.85	21.90
fvs linear	22.24					
lar linear	21.91					
las linear	22.05					
for linear	22.14					
boo quadratic	20.97	19.70	**19.70**	19.72	19.71	19.71
cdb quadratic	20.88	19.81	19.80	19.80	**19.79**	19.80
fvs quadratic	21.10					
lar quadratic	19.89					
las quadratic	19.90					
for quadratic	19.90					
boo splines	20.23	19.26	**19.22**	19.23	19.23	19.24
cdb splines	20.98	19.57	19.27	19.25	**19.25**	19.41
boo stumps	26.84	21.40	21.03	21.01	**20.99**	21.11
cdb stumps	26.50	21.26	21.06	21.05	**21.04**	21.22
mars	22.47					

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

There are differences between the learners but there are almost no differences between the methods. Only forward variable selection (fvs) and MARS fall behind. In contrast to the simulated models, CDBoost doesn't get worse when ν becomes smaller. But most of the

time, $\nu = 0.1$ is small enough to give good results and the validation over ν does not pay off.

9.2 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. It is actually a binary classification problem, but we treat it as a regression problem with outcome 0 and 1 and L_2-loss. We use the fitted values to classify the samples with cut-point one half and compute the misclassification rate. 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.

	Shrinkage					
	1	0.3	0.1	0.03	0.01	validated
	L_2-loss					
boo linear	0.1001	0.0459	**0.0430**	0.0431	0.0432	0.0441
cdb linear	0.0985	0.0418	0.0413	0.0413	**0.0412**	0.0413
lar linear	0.0470					
las linear	0.0448					
	Misclassification rate					
boo linear	9.2%	5.2%	5.5%	5.3%	**5.2%**	5.3%
cdb linear	9.0%	4.6%	4.3%	4.3%	**4.1%**	4.5%
lar linear	5.3%					
las linear	5.2%					

Table 8: Comparison of the different methods using the leukemia dataset. Given is the mean of the test error (L_2-loss and misclassification rate) over 100 random splits. The best values for boosting and CDBoost are in bold face. Boosting and CDBoost are also validated over the shrinkage factor ν taking the values $0.3, 0.1, 0.03, 0.01$.

In this example CDBoost performs better than boosting, lasso and is much better than LAR. Shrinkage improves the fits dramatically and the choice of ν is again not crucial.

10 Discussion

We propose a new conjugate direction boosting method (CDBoost) and show that it can outperform boosting, LAR and 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 main idea is to use a conjugate direction method instead of a simple gradient descent as in boosting. This leads to a fast forward variable selection method which is of interest for very large datasets (n and d large). However, it is well known that forward variable selection is too greedy, taking too long steps. The inclusion of shrinkage (small step sizes) is possible with a restart condition which substantially increases the performance, as is the case also for boosting (compare also with Friedman 2001).

The three LARS methods all describe a different path from the intercept model to the full least squares fit. CDBoost with infinitesimal shrinkage leads to a path between LAR

and forward stagewise fitting. The first chosen covariate plays a special role and is, using the LARS language, always in the active set. Other variables can be dropped during the iteration process.

When we use a real (non-infinitesimal) shrinkage factor, we depart from the idealised (infinitesimal shrinkage) conjugate direction boosting path. The departure is proportional to the shrinkage factor ν. In some cases, e.g. in our simulation model 2 in section 6.2, the shrinkage factor ν shouldn't be to small for CDBoost. While the LARS methods use in some sense infinitesimal shrinkage, they are too cautious in this example. A computational advantage of using a larger ν is that we don't need too many iterations.

For linear models, the performance of the methods shows an interesting dependence on the signal-to-noise ratio (*stnr*): compared to boosting, the gain of CDBoost increases with higher *stnr*. LAR and lasso are worse than boosting for low *stnr* and better for high *stnr*. Forward variable selection is much worse than boosting for low *stnr* and almost equal for high *stnr*. Only forward stagewise fitting behaves qualitatively like boosting. A possible explanation is that forward stagewise fitting (and boosting) is in some sense not as flexible as LAR and forward variable selection because it may drop variables from the active set and therefore does not adjust all the coefficients in each step. But it often seems worthwhile to adjust all the coefficients of the terms already included in the model, especially in low noise situations. Although also CDBoost and lasso can drop variables from the active set, they seem to be closer to LAR than to boosting.

In the given real data examples, the choice of ν is not crucial. CDBoost gives nearly identical results for ν between 0.01 and 0.1. While for the ozone data all methods (except forward variable selection) perform almost equal, CDBoost performs best for the more complicated leukemia data.

As a drawback, CDBoost is often computationally more expensive and also 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.

References

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.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004). Least angle regression, *To appear in Annals of Statistics*.

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.

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

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

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