Greedy Function Approximation: A Gradient Boosting Machine

Jerome H. Friedman*
IMS 1999 Reitz Lecture

February 24,1999 (modified March 15, 2000, April 19, 2001)

Abstract

Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest–descent minimization. A general gradient–descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least–squares, least–absolute–deviation, and Huber–M loss functions for regression, and multi–class logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire 1996, and Friedman, Hastie, and Tibshirani 2000 are discussed.

1 Function estimation

In the function estimation or "predictive learning" problem, one has a system consisting of a random "output" or "response" variable y and a set of random "input" or "explanatory" variables $\mathbf{x} = \{x_1, \cdots, x_n\}$. Using a "training" sample $\{y_i, \mathbf{x}_i\}_1^N$ of known $(y, \mathbf{x})$–values, the goal is to obtain an estimate or approximation $\hat{F}(\mathbf{x})$, of the function $F^*(\mathbf{x})$ mapping $\mathbf{x}$ to y, that minimizes the expected value of some specified loss function $L(y, F(\mathbf{x}))$ over the joint distribution of all $(y, \mathbf{x})$–values

$$F^* = \arg\min_F E_{y,\mathbf{x}} L(y, F(\mathbf{x})) = \arg\min_F E_{\mathbf{x}} \left[E_y(L(y, F(\mathbf{x}))) \,|\, \mathbf{x} \right]. \tag{1}$$

Frequently employed loss functions $L(y, F)$ include squared–error $(y - F)^2$ and absolute error $|y - F|$ for $y \in R^1$ (regression), and negative binomial log–likelihood, $\log(1 + e^{-2yF})$, when $y \in \{-1, 1\}$ (classification).

A common procedure is to restrict $F(\mathbf{x})$ to be a member of a parameterized class of functions $F(\mathbf{x}; \mathbf{P})$, where $\mathbf{P} = \{P_1, P_2, \cdots\}$ is a finite set of parameters whose joint values identify individual class members. In this paper we focus on "additive" expansions of the form

$$F(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{2}$$

The (generic) function $h(\mathbf{x}; \mathbf{a})$ in (2) is usually a simple parameterized function of the input variables $\mathbf{x}$, characterized by parameters $\mathbf{a} = \{a_1, a_2, \cdots\}$. The individual terms differ in the joint

*Sequoia Hall, Stanford University, Stanford, CA 94305; jhf@stat.stanford.edu

values $\mathbf{a}_m$ chosen for these parameters. Such expansions (2) are at the heart of many function approximation methods such as neural networks (Rumelhart, Hinton, and Williams 1986), radial basis functions (Powell 1987), MARS (Friedman 1991), wavelets (Donoho 1993), and support vector machines (Vapnik 1995). Of special interest here is the case where each of the functions $h(\mathbf{x};\mathbf{a}_m)$ is a small regression tree, such as those produced by $CART^{\text{TM}}$ (Breiman, Friedman, Olshen, and Stone 1983). For a regression tree the parameters $\mathbf{a}_m$ are the splitting variables, split locations, and the terminal node means of the individual trees.

1.1 Numerical optimization

In general, choosing a parameterized model $F(\mathbf{x};\mathbf{P})$ changes the function optimization problem to one of parameter optimization

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \Phi(\mathbf{P}) \tag{3}$$

where

$$\Phi(\mathbf{P}) = E_{y,\mathbf{x}} L(y, F(\mathbf{x};\mathbf{P}))$$

and then

$$F^*(\mathbf{x}) = F(\mathbf{x};\mathbf{P}^*).$$

For most $F(\mathbf{x};\mathbf{P})$ and L, numerical optimization methods must be applied to solve (3). This often involves expressing the solution for the parameters in the form

$$\mathbf{P}^* = \sum_{m=0}^{M} \mathbf{p}_m \tag{4}$$

where $\mathbf{p}_0$ is an initial guess and $\{\mathbf{p}_m\}_1^M$ are successive increments ("steps" or "boosts"), each based on the sequence of preceding steps. The prescription for computing each step $\mathbf{p}_m$ is defined by the optimization method.

1.2 Steepest–descent

Steepest–descent is one of the simplest of the frequently used numerical minimization methods. It defines the increments $\{\mathbf{p}_m\}_1^M$ (4) as follows. First the current gradient $\mathbf{g}_m$ is computed

$$\mathbf{g}_m = \{g_{jm}\} = \left\{ \left[\frac{\partial \Phi(\mathbf{P})}{\partial P_j} \right]_{\mathbf{P}=\mathbf{P}_{m-1}} \right\}$$

where

$$\mathbf{P}_{m-1} = \sum_{i=0}^{m-1} \mathbf{p}_i.$$

The step is taken to be

$$\mathbf{p}_m = -\rho_m \mathbf{g}_m$$

where

$$\rho_m = \arg\min_{\rho} \Phi\left(\mathbf{P}_{m-1} - \rho \mathbf{g}_m\right). \tag{5}$$

The negative gradient $-\mathbf{g}_m$ is said to define the "steepest–descent" direction and (5) is called the "line search" along that direction.

2 Numerical optimization in function space

Here we take a "nonparametric" approach and apply numerical optimization in function space. That is we consider $F(\mathbf{x})$ evaluated at each point $\mathbf{x}$ to be a "parameter" and seek to minimize

$$\Phi(F) = E_{y,\mathbf{x}} L(y, F(\mathbf{x})) = E_{\mathbf{x}} \left[E_y(L(y, F(\mathbf{x}))) \,|\, \mathbf{x} \right],$$

or equivalently

$$\phi(F(\mathbf{x})) = E_y[L(y, F(\mathbf{x}))) | \, \mathbf{x}]$$

at each individual $\mathbf{x}$, directly with respect to $F(\mathbf{x})$. In function space there are an infinite number of such parameters, but in data sets (discussed below) only a finite number $\{F(\mathbf{x}_i)\}_1^N$ are involved. Following the numerical optimization paradigm we take the solution to be

$$F^*(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x})$$

where $f_0(\mathbf{x})$ is an initial guess, and $\{f_m(\mathbf{x})\}_1^M$ are incremental functions ("steps" or "boosts") defined by the optimization method.

For steepest–descent

$$f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x}) \tag{6}$$

with

$$g_m(\mathbf{x}) = \left[\frac{\partial \phi(F(\mathbf{x}))}{\partial F(\mathbf{x})} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left[\frac{\partial E_y[L(y, F(\mathbf{x})) \,|\, \mathbf{x}]}{\partial F(\mathbf{x})} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

and

$$F_{m-1}(\mathbf{x}) = \sum_{i=0}^{m-1} f_i(\mathbf{x}).$$

Assuming sufficient regularity that one can interchange differentiation and integration, this becomes

$$g_m(\mathbf{x}) = E_y \left[\frac{\partial L(y, F(\mathbf{x}))}{\partial F(\mathbf{x})} \,|\, \mathbf{x} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}. \tag{7}$$

The multiplier ρ_m in (6) is given by the line search

$$\rho_m = \arg\min_{\rho} E_{y,\mathbf{x}} L\left(y, F_{m-1}(\mathbf{x}) - \rho g_m(\mathbf{x})\right). \tag{8}$$

3 Finite data

This nonparametric approach breaks down when the joint distribution of $(y, \mathbf{x})$ is estimated by a finite data sample $\{y_i, \mathbf{x}_i\}_1^N$. In this case $E_y[\cdot \,|\, \mathbf{x}]$ cannot be estimated accurately by its data value at each $\mathbf{x}_i$, and even if it could, one would like to estimate $F^*(\mathbf{x})$ at $\mathbf{x}$ values other than the training sample points. Strength must be borrowed from nearby data points by imposing smoothness on the solution. One way to do this is to assume a parameterized form such as (2) and do parameter optimization as discussed in Section 1.1 to minimize the corresponding data based estimate of expected loss

$$\{\beta_m, \mathbf{a}_m\}_1^M = \arg\min_{\{\beta'_m, \mathbf{a}'_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta'_m h(\mathbf{x}_i; \mathbf{a}'_m) \right).$$

In situations where this is infeasible one can try a "greedy–stagewise" approach. For $m = 1, 2, \cdots, M$

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a})\right) \tag{9}$$

and then

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{10}$$

Note that this *stagewise* strategy is different from *stepwise* approaches that readjust previously entered terms when new ones are added.

In signal processing this stagewise strategy is called "matching pursuit" (Mallat and Zhang 1993) where $L(y, F)$ is squared–error loss and the $\{h(\mathbf{x}; \mathbf{a}_m)\}_1^M$ are called basis functions, usually taken from an over–complete wavelet–like dictionary. In machine learning, (9) (10) is called "boosting" where $y \in \{-1, 1\}$ and $L(y, F)$ is either an exponential loss criterion e^{-yF} (Freund and Schapire 1996, Schapire and Singer 1998) or negative binomial log–likelihood (Friedman, Hastie, and Tibshirani 2000 – FHT00). The function $h(\mathbf{x}; \mathbf{a})$ is called a "weak learner" or "base learner", and is usually a classification tree.

Suppose that for a particular loss $L(y, F)$ and/or base learner $h(\mathbf{x}; \mathbf{a})$ the solution to (9) is difficult to obtain. Given any approximator $F_{m-1}(\mathbf{x})$, the function $\beta_m h(\mathbf{x}; \mathbf{a}_m)$ (9) (10) can be viewed the best greedy step towards the data based estimate of $F^*(\mathbf{x})$ (1), under the constraint that the step "direction" $h(\mathbf{x}; \mathbf{a}_m)$ be a member of the parameterized class of functions $h(\mathbf{x}; \mathbf{a})$. It can thus be regarded as a steepest–descent step (6) under that constraint. By construction, the data based analogue of the unconstrained negative gradient (7)

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}$$

gives the best steepest–descent step direction $-\mathbf{g}_m = \{-g_m(\mathbf{x}_i)\}_1^N$ in the N–dimensional data space at $F_{m-1}(\mathbf{x})$. However, this gradient is defined only at the data points $\{\mathbf{x}_i\}_1^N$ and cannot be generalized to other $\mathbf{x}$–values. One possibility for generalization is to choose that member of the parameterized class $h(\mathbf{x}; \mathbf{a}_m)$ that produces $\mathbf{h}_m = \{h(\mathbf{x}_i; \mathbf{a}_m)\}_1^N$ most parallel to $-\mathbf{g}_m \in R^N$. This is the $h(\mathbf{x}; \mathbf{a})$ most highly correlated with $-g_m(\mathbf{x})$ over the data distribution. It can be obtained from the solution

$$\mathbf{a}_m = \arg\min_{\mathbf{a}, \beta} \sum_{i=1}^{N} [-g_m(\mathbf{x}_i) - \beta h(\mathbf{x}_i; \mathbf{a})]^2. \tag{11}$$

This constrained negative gradient $h(\mathbf{x}; \mathbf{a}_m)$ is used in place of the unconstrained one $-g_m(\mathbf{x})$ (7) in the steepest–descent strategy. Specifically, the line search (8) is performed

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\right) \tag{12}$$

and the approximation updated

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m).$$

Basically, instead of obtaining the solution under a smoothness constraint (9), the constraint is applied to the unconstrained (rough) solution by fitting $h(\mathbf{x}; \mathbf{a})$ to the "pseudo–responses" $\{\tilde{y}_i = -g_m(\mathbf{x}_i)\}_{i=1}^N$ (7). This permits the replacement of the difficult function minimization problem (9) by least–squares function minimization (11), followed by only a single parameter optimization based on the original criterion (12). Thus, for any $h(\mathbf{x}; \mathbf{a})$ for which a feasible least–squares algorithm exists for solving (11), one can use this approach to minimize any differentiable loss $L(y, F)$ in conjunction with forward stagewise additive modeling. This leads to the following (generic) algorithm using steepest–descent.

	Algorithm 1: Gradient_Boost
1	$F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L\left(y_i, \rho\right)$
2	For $m = 1$ to M do:
3	$\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4	$\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5	$\quad \rho_m = \arg\min_\rho \sum_{i=1}^N L\left(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\right)$
6	$\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7	endFor
	end Algorithm

Note that any fitting criterion that estimates conditional expectation (given $\mathbf{x}$) could in principle be used to estimate the (smoothed) negative gradient (7) at line 4 of Algorithm 1. Least–squares (11) is a natural choice owing to the superior computational properties of many least–squares algorithms.

In the special case where $y \in \{-1, 1\}$ and the loss function $L\left(y, F\right)$ depends on y and F only through their product $L\left(y, F\right) = L\left(yF\right)$, the analogy of boosting (9) (10) to steepest–descent minimization has been noted in the machine learning literature (Ratsch, Onoda, and Muller 1998, Breiman 1999). Duffy and Helmbold 1999 elegantly exploit this analogy to motivate their GeoLev and GeoArc procedures. The quantity yF is called the "margin" and the steepest–descent is performed in the space of margin values, rather than the space of function values F. The latter approach permits application to more general loss functions where the notion of margins is not apparent. Drucker 1997 employs a different strategy of casting regression into the framework of classification in the context of the AdaBoost algorithm (Freund and Schapire 1996).

4 Applications: additive modeling

In this section the gradient boosting strategy is applied to several popular loss criteria: least–squares (LS), least–absolute–deviation (LAD), Huber (M), and logistic binomial log–likelihood (L). The first serves as a "reality check", where as the others lead to new boosting algorithms.

4.1 Least–squares regression

Here $L\left(y, F\right) = (y - F)^2/2$. The pseudo–response in line 3 of Algorithm 1 is $\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i)$. Thus, line 4 simply fits the current residuals and the line search (line 5) produces the result $\rho_m = \beta_m$, where β_m is the minimizing β of line 4. Therefore, gradient boosting on squared–error loss produces the usual *stagewise* approach of iteratively fitting the current residuals:

Algorithm 2: LS_Boost
$F_0(\mathbf{x}) = \bar{y}$
For $m = 1$ to M do:
$\quad \tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$
$\quad (\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^N [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$
$\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor
end Algorithm

4.2 Least-absolute-deviation (LAD) regression

For the loss function $L\left(y, F\right) = |y - F|$, one has

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = sign(y_i - F_{m-1}(\mathbf{x}_i)). \tag{13}$$

This implies that $h(\mathbf{x};\mathbf{a})$ is fit (by least–squares) to the *sign* of the current residuals in line 4 of Algorithm 1. The line search (line 5) becomes

$$\begin{aligned}\rho_m &= \arg\min_{\rho} \sum_{i=1}^{N} |y_i - F_{m-1}(\mathbf{x}_i) - \rho h(\mathbf{x}_i;\mathbf{a}_m)| \\ &= \arg\min_{\rho} \sum_{i=1}^{N} |h(\mathbf{x}_i;\mathbf{a}_m)| \cdot \left| \frac{y_i - F_{m-1}(\mathbf{x}_i)}{h(\mathbf{x}_i;\mathbf{a}_m)} - \rho \right| \\ &= median_W \left\{ \frac{y_i - F_{m-1}(\mathbf{x}_i)}{h(\mathbf{x}_i;\mathbf{a}_m)} \right\}_1^N, \quad w_i = |h(\mathbf{x}_i;\mathbf{a}_m)|. \end{aligned} \tag{14}$$

Here $median_W\{\cdot\}$ is the weighted median with weights w_i. Inserting these results (13) (14) into Algorithm 1 yields an algorithm for least-absolute-deviation boosting, using any base learner $h(\mathbf{x};\mathbf{a})$.

4.3 Regression trees

Here we consider the special case where each base learner is an J–terminal node regression tree (Breiman, Friedman, Olshen and Stone 1983). Each regression tree model itself has the additive form

$$h(\mathbf{x};\{b_j,R_j\}_1^J) = \sum_{j=1}^{J} b_j 1(\mathbf{x}\in R_j). \tag{15}$$

Here $\{R_j\}_1^J$ are disjoint regions that collectively cover the space of all joint values of the predictor variables $\mathbf{x}$. These regions are represented by the terminal nodes of the corresponding tree. The indicator function $1(\cdot)$ has the value one if its argument is true, and zero otherwise. The "parameters" of this base learner (15) are the coefficients $\{b_j\}_1^J$, and the quantities that define the boundaries of the regions $\{R_j\}_1^J$. These are the splitting variables, and the values of those variables, that represent the splits at the nonterminal nodes of the tree. Because the regions are disjoint, (15) is equivalent to the prediction rule: if $\mathbf{x}\in R_j$ then $h(\mathbf{x}) = b_j$.

For a regression tree, the update at line 6 of Algorithm 1 becomes

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m \sum_{j=1}^{J} b_{jm} 1(\mathbf{x}\in R_{jm}). \tag{16}$$

Here $\{R_{jm}\}_1^J$ are the regions defined by the terminal nodes of the tree at the mth iteration. They are constructed to predict the pseudo–responses $\{\tilde{y}_i\}_1^N$ (line 3) by least–squares (line 4). The $\{b_{jm}\}$ are the corresponding least–squares coefficients

$$b_{jm} = ave_{\mathbf{x}_i\in R_{jm}} \tilde{y}_i.$$

The scaling factor ρ_m is the solution to the "line search" at line 5.

The update (16) can be alternatively expressed as

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x}\in R_{jm}) \tag{17}$$

with $\gamma_{jm} = \rho_m b_{jm}$. One can view (17) as adding J separate basis functions at each step $\{1(\mathbf{x}\in R_{jm})\}_1^J$, instead of a single additive one as in (16). Thus, in this case one can further improve the quality of the fit by using the optimal coefficients for each of these separate basis functions (17). These optimal coefficients are the solution to

$$\{\gamma_{jm}\}_1^J = \arg\min_{\{\gamma_j\}_1^J} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_j 1(\mathbf{x}\in R_{jm})\right).$$

Owing to the disjoint nature of the regions produced by regression trees, this reduces to

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma). \tag{18}$$

This is just the optimal constant update in each terminal node region, based on the loss function L, given the current approximation $F_{m-1}(\mathbf{x})$.

For the case of LAD regression (18) becomes

$$\gamma_{jm} = \mathit{median}_{\mathbf{x}_i \in R_{jm}} \{y_i - F_{m-1}(\mathbf{x}_i)\},$$

which is simply the median of the current residuals in the jth terminal node at the mth iteration. At each iteration a regression tree is built to best predict the *sign* of the current residuals $y_i - F_{m-1}(\mathbf{x}_i)$, based on a least–squares criterion. Then the approximation is updated by adding the *median* of the residuals in each of the derived terminal nodes.

Algorithm 3: LAD_TreeBoost
$F_0(\mathbf{x}) = \mathit{median}\{y_i\}_1^N$
For $m = 1$ to M do:
  $\tilde{y}_i = \mathit{sign}(y_i - F_{m-1}(\mathbf{x}_i)),\ i = 1, N$
  $\{R_{jm}\}_1^J = J$–terminal node $\mathit{tree}(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
  $\gamma_{jm} = \mathit{median}_{\mathbf{x}_i \in R_{jm}} \{y_i - F_{m-1}(\mathbf{x}_i)\},\ j = 1, J$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$
endFor
end Algorithm

This algorithm is highly robust. The trees use only order information on the individual input variables x_j, and the pseudo–responses $\tilde{y}_i$ (13) have only two values, $\tilde{y}_i \in \{-1, 1\}$. The terminal node updates are based on medians. An alternative approach would be to build a tree to directly minimize the loss criterion

$$\mathit{tree}_m(\mathbf{x}) = \arg \min_{J\text{–node } \mathit{tree}} \sum_{i=1}^{N} |y_i - F_{m-1}(\mathbf{x}_i) - \mathit{tree}(\mathbf{x}_i)|$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}_i) + \mathit{tree}_m(\mathbf{x}).$$

However, Algorithm 3 is much faster since it uses least–squares to induce the trees. Squared–error loss is much more rapidly updated than mean-absolute-deviation when searching for splits during the tree building process.

4.4 M–Regression

M–regression techniques attempt resistance to long–tailed error distributions and outliers while maintaining high efficiency for normally distributed errors. We consider the Huber loss function (Huber 1964)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \le \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}. \tag{19}$$

Here the pseudo–response is

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})} = \begin{cases} y_i - F_{m-1}(\mathbf{x}_i) & |y_i - F_{m-1}(\mathbf{x}_i)| \le \delta \\ \delta \cdot \mathit{sign}(y_i - F_{m-1}(\mathbf{x}_i)) & |y_i - F_{m-1}(\mathbf{x}_i)| > \delta \end{cases}.$$

and the line search becomes

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\right) \tag{20}$$

with L given by (19). The solution to (19) (20) can be obtained by standard iterative methods (see Huber 1964).

The value of the transition point δ defines those residual values that are considered to be "outliers", subject to absolute rather than squared–error loss. An optimal value will depend on the distribution of $y - F^*(\mathbf{x})$, where F^* is the true target function (1). A common practice is to choose the value of δ to be the α–quantile of the distribution of $|y - F^*(\mathbf{x})|$, where $(1 - \alpha)$ controls the break–down point of the procedure. The "break–down" point is the fraction of observations that can be arbitrarily modified without seriously degrading the quality of the result. Since $F^*(\mathbf{x})$ is unknown one uses the current estimate $F_{m-1}(\mathbf{x})$ as an approximation at the mth iteration. The distribution of $|y - F_{m-1}(\mathbf{x})|$ is estimated by the current residuals, leading to

$$\delta_m = \text{quantile}_\alpha\{|y_i - F_{m-1}(\mathbf{x}_i)|\}_1^N.$$

With regression trees as base learners we use the strategy of Section 4.3; that is, a separate update (18) in each terminal node R_{jm}. For the Huber loss (19) the solution to (18) can be approximated by a single step of the standard iterative procedure (Huber 1964) starting at the median

$$\tilde{r}_{jm} = median_{\mathbf{x}_i \in R_{jm}}\{r_{m-1}(\mathbf{x}_i)\}.$$

where $\{r_{m-1}(\mathbf{x}_i)\}_1^N$ are the current residuals

$$r_{m-1}(\mathbf{x}_i) = y_i - F_{m-1}(\mathbf{x}_i).$$

The approximation is

$$\gamma_{jm} = \tilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{\mathbf{x}_i \in R_{jm}} sign(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm}) \cdot \min(\delta_m, abs(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm})),$$

where N_{jm} is the number of observations in the jth terminal node. This gives the following algorithm for boosting regression trees based on Huber loss (19):

Algorithm 4: M_TreeBoost
$F_0(\mathbf{x}) = median\{y_i\}_1^N$
For $m = 1$ to M do:
  $r_{m-1}(\mathbf{x}_i) = y_i - F_{m-1}(\mathbf{x}_i),\ i = 1, N$
  δ_m =quantile$_\alpha\{|r_{m-1}(\mathbf{x}_i)|\}_1^N$
  $\tilde{y}_i = \begin{cases} r_{m-1}(\mathbf{x}_i) & |r_{m-1}(\mathbf{x}_i)| \le \delta_m \\ \delta_m \cdot sign(r_{m-1}(\mathbf{x}_i)) & |r_{m-1}(\mathbf{x}_i)| > \delta_m \end{cases},\ i = 1, N$
  $\{R_{jm}\}_1^J = J$–terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
  $\tilde{r}_{jm} = median_{\mathbf{x}_i \in R_{jm}}\{r_{m-1}(\mathbf{x}_i)\},\ j = 1, J$
  $\gamma_{jm} = \tilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{\mathbf{x}_i \in R_{jm}} sign(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm}) \cdot \min(\delta_m, abs(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm})),$
    $j = 1, J$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$
endFor
end Algorithm

According to the motivations underlying robust regression, this algorithm should have properties similar to that of least–squares boosting (Algorithm 2) for normally distributed errors, and similar to that of least–absolute–deviation regression (Algorithm 3) with very long tailed distributions. For error distributions with only moderately long tails it can have performance superior to both (See Section 6.2).

4.5 Two–class logistic regression and classification

Here the loss function is negative binomial log–likelihood (FHT00)

$$L(y, F) = \log\left(1 + \exp(-2yF)\right), \quad y \in \{-1, 1\},$$

where

$$F(\mathbf{x}) = \frac{1}{2}\log\left[\frac{\Pr(y = 1 \,|\, \mathbf{x})}{\Pr(y = -1 \,|\, \mathbf{x})}\right]. \tag{21}$$

The pseudo–response is

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})} = 2y_i / (1 + \exp(2y_i F_{m-1}(\mathbf{x}_i))). \tag{22}$$

The line search becomes

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} \log\left(1 + \exp(-2y_i(F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)))\right).$$

With regression trees as base learners we again use the strategy (Section 4.3) of separate updates in each terminal node R_{jm}:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} \log\left(1 + \exp(-2y_i(F_{m-1}(\mathbf{x}_i) + \gamma))\right). \tag{23}$$

There is no closed form solution to (23). Following FHT00, we approximate it by a single Newton–Raphson step. This turns out to be

$$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i \Big/ \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i| \,(2 - |\tilde{y}_i|)$$

with $\tilde{y}_i$ given by (22). This gives the following algorithm for likelihood gradient boosting with regression trees:

Algorithm 5: L_2_TreeBoost
$F_0(\mathbf{x}) = \frac{1}{2}\log\frac{1+\bar{y}}{1-\bar{y}}$
For $m = 1$ to M do:
  $\tilde{y}_i = 2y_i / (1 + \exp(2y_i F_{m-1}(\mathbf{x}_i)))$, $i = 1, N$
  $\{R_{jm}\}_1^J = J$–terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$
  $\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i| \,(2 - |\tilde{y}_i|)$, $j = 1, J$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$
endFor
end Algorithm

The final approximation $F_M(\mathbf{x})$ is related to log–odds through (21). This can be inverted to yield probability estimates

$$p_+(\mathbf{x}) = \widehat{\Pr}(y = 1 \,|\, \mathbf{x}) = 1/\left(1 + e^{-2F_M(\mathbf{x})}\right)$$
$$p_-(\mathbf{x}) = \widehat{\Pr}(y = -1 \,|\, \mathbf{x}) = 1/\left(1 + e^{2F_M(\mathbf{x})}\right).$$

These in turn can be used for classification

$$\hat{y}(\mathbf{x}) = 2 \cdot 1[c(-1, 1)\, p_+(\mathbf{x}) > c(1, -1)\, p_-(\mathbf{x})] - 1$$

where $c(\hat{y}, y)$ is the cost associated with predicting $\hat{y}$ when the truth is y.

4.5.1 Influence trimming

The empirical loss function for the two–class logistic regression problem at the mth iteration is

$$\phi_m(\rho, \mathbf{a}) = \sum_{i=1}^{N} \log[1 + \exp(-2y_i F_{m-1}(\mathbf{x}_i)) \cdot \exp(-2y_i \rho h(\mathbf{x}_i; \mathbf{a}))]. \tag{24}$$

If $y_i F_{m-1}(\mathbf{x}_i)$ is very large, then (24) has almost no dependence on $\rho h(\mathbf{x}_i; \mathbf{a})$ for small to moderate values near zero. This implies that the ith observation $(y_i, \mathbf{x}_i)$ has almost no influence on the loss function, and therefore on its solution

$$(\rho_m, \mathbf{a}_m) = \arg\min_{\rho, \mathbf{a}} \phi_m(\rho, \mathbf{a}).$$

This suggests that all observations $(y_i, \mathbf{x}_i)$ for which $y_i F_{m-1}(\mathbf{x}_i)$ is relatively very large can be deleted from all computations of the mth iteration without having a substantial effect on the result. Thus,

$$w_i = \exp(-2y_i F_{m-1}(\mathbf{x}_i)) \tag{25}$$

can be viewed as a measure of the "influence" or weight of the ith observation on the estimate $\rho_m h(\mathbf{x}; \mathbf{a}_m)$.

More generally, from the nonparametric function space perspective of Section 2, the parameters are the observation function values $\{F(\mathbf{x}_i)\}_1^N$. The influence on an estimate to changes in a "parameter" value $F(\mathbf{x}_i)$ (holding all the other parameters fixed) can be gauged by the second derivative of the loss function with respect to that parameter. Here this second derivative at the mth iteration is $|\tilde{y}_i|\,(2 - |\tilde{y}_i|)$ with $\tilde{y}_i$ given by (22). Thus, another measure of the influence or "weight" of the ith observation on the estimate $\rho_m h(\mathbf{x}; \mathbf{a}_m)$ at the mth iteration is

$$w_i = |\tilde{y}_i|\,(2 - |\tilde{y}_i|). \tag{26}$$

Influence trimming deletes all observations with w_i–values less than $w_{l(\alpha)}$, where $l(\alpha)$ is the solution to

$$\sum_{i=1}^{l(\alpha)} w_{(i)} = \alpha \sum_{i=1}^{N} w_i. \tag{27}$$

Here $\{w_{(i)}\}_1^N$ are the weights $\{w_i\}_1^N$ arranged in ascending order. Typical values are $\alpha \in [0.05, 0.2]$. Note that influence trimming based on (25) (27) is identical to the "weight trimming" strategy employed with Real AdaBoost, whereas (26) (27) is equivalent to that used with LogitBoost, in FHT00. There it was seen that 90% to 95% of the observations were often deleted without sacrificing accuracy of the estimates, using either influence measure. This results in a corresponding reduction in computation by factors of 10 to 20.

4.6 Multi–class logistic regression and classification

Here we develop a gradient–descent boosting algorithm for the K–class problem. The loss function is

$$L\left(\{y_k, F_k(\mathbf{x})\}_1^K\right) = -\sum_{k=1}^{K} y_k \log p_k(\mathbf{x}), \tag{28}$$

where $y_k = 1(\text{class} = k) \in \{0, 1\}$, and $p_k(\mathbf{x}) = \Pr(y_k = 1 \,|\, \mathbf{x})$. Following FHT00, we use the symmetric multiple logistic transform

$$F_k(\mathbf{x}) = \log p_k(\mathbf{x}) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(\mathbf{x}) \tag{29}$$

or equivalently

$$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) \Big/ \sum_{l=1}^{K} \exp(F_l(\mathbf{x})) \; . \tag{30}$$

Substituting (30) into (28) and taking first derivatives one has

$$\tilde{y}_{ik} = -\left[\frac{\partial L(\{y_{il}, F_l(\mathbf{x}_i)\}_{l=1}^{K})}{\partial F_k(\mathbf{x}_i)}\right]_{\{F_l(\mathbf{x})=F_{l,m-1}(\mathbf{x})\}_1^K} = y_{ik} - p_{k,m-1}(\mathbf{x}_i) \tag{31}$$

where $p_{k,m-1}(\mathbf{x})$ is derived from $F_{k,m-1}(\mathbf{x})$ through (30). Thus, K–trees are induced at each iteration m to predict the corresponding current residuals for each class on the probability scale. Each of these trees has J–terminal nodes, with corresponding regions $\{R_{jkm}\}_{j=1}^{J}$. The model updates γ_{jkm} corresponding to these regions are the solution to

$$\{\gamma_{jkm}\} = \arg\min_{\{\gamma_{jk}\}} \sum_{i=1}^{N} \sum_{k=1}^{K} \phi\left(y_{ik}, F_{k,m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_{jk} 1(\mathbf{x}_i \in R_{jm})\right)$$

where $\phi(y_k, F_k) = -y_k \log p_k$ from (28), with F_k related to p_k through (30). This has no closed form solution. Moreover, the regions corresponding to the different class trees overlap, so that the solution does not reduce to a separate calculation within each region of each tree in analogy with (18). Following FHT00, we approximate the solution with a single Newton–Raphson step, using a diagonal approximation to the Hessian. This decomposes the problem into a separate calculation for each terminal node of each tree. The result is

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}| \, (1 - |\tilde{y}_{ik}|)}. \tag{32}$$

This leads to the following algorithm for K–class logistic gradient boosting:

Algorithm 6: $\mathbf{L_K}$-TreeBoost
$F_{k0}(\mathbf{x}) = 0, \quad k = 1, K$
For $m = 1$ to M do:
  $p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) \Big/ \sum_{l=1}^{K} \exp(F_l(\mathbf{x})), \quad k = 1, K$
  For $k = 1$ to K do:
    $\tilde{y}_{ik} = y_{ik} - p_k(\mathbf{x}_i), \quad i = 1, N$
    $\{R_{jkm}\}_{j=1}^{J} = J$–terminal node $tree(\{\tilde{y}_{ik}, \mathbf{x}_i\}_1^N)$
    $\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}| \, (1 - |\tilde{y}_{ik}|)}, \quad j = 1, J$
    $F_{km}(\mathbf{x}) = F_{k,m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jkm} 1(\mathbf{x} \in R_{jkm})$
  endFor
endFor
end Algorithm

The final estimates $\{F_{kM}(\mathbf{x})\}_1^K$ can be used to obtain corresponding probability estimates $\{p_{kM}(\mathbf{x})\}_1^K$ through (30). These in turn can be used for classification

$$\hat{k}(\mathbf{x}) = \arg\min_{1 \le k \le K} \sum_{k'=1}^{K} c(k, k') \, p_{k'M}(\mathbf{x})$$

where $c(k, k')$ is the cost associated with predicting the kth class when the truth is k'. Note that for $K = 2$ Algorithm 6 is equivalent to Algorithm 5.

Algorithm 6 bears a close similarity to the K–class LogitBoost procedure of FHT00, which is based on Newton-Raphson rather than gradient descent in function space. In that algorithm K trees were induced, each using corresponding pseudo–responses

$$\tilde{y}_{ik} = \frac{K-1}{K}\frac{y_{ik} - p_k(\mathbf{x}_i)}{p_k(\mathbf{x}_i)\,(1 - p_k(\mathbf{x}_i))} \tag{33}$$

and a weight

$$w_k(\mathbf{x}_i) = p_k(\mathbf{x}_i)\,(1 - p_k(\mathbf{x}_i)) \tag{34}$$

applied to each observation $(\tilde{y}_{ik}, \mathbf{x}_i)$. The terminal node updates were

$$\gamma_{jkm} = \frac{\sum_{\mathbf{x}_i \in R_{jkm}} w_k(\mathbf{x}_i)\,\tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} w_k(\mathbf{x}_i)}$$

which is equivalent to (32). The difference between the two algorithms is the splitting criterion used to induce the trees and thereby the terminal regions $\{R_{jkm}\}_1^J$.

The least–squares improvement criterion used to evaluate potential splits of a currently terminal region R into two subregions (R_l, R_r) is

$$i^2(R_l, R_r) = \frac{w_l w_r}{w_l + w_r}(\bar{y}_l - \bar{y}_r)^2 \tag{35}$$

where $\bar{y}_l$, $\bar{y}_r$ are the left and right daughter response means respectively, and w_l, w_r are the corresponding sums of the weights. For a given split, using (31) with unit weights, or (33) with weights (34), give the same values for $\bar{y}_l$, $\bar{y}_r$. However, the weight sums w_l, w_r are different. Unit weights (L_K_TeeBoost) favor splits that are symmetric in the number of observations in each daughter node, whereas (34) (LogitBoost) favors splits for which the sums of the currently estimated response variances $var(y_{ik}) = p_k(\mathbf{x}_i)\,(1 - p_k(\mathbf{x}_i))$ are more equal.

L_K_TreeBoost has an implementation advantage in numerical stability. LogitBoost becomes numerically unstable whenever the value of (34) is close to zero for *any* observation $\mathbf{x}_i$, which happens quite frequently. This is a consequence of the difficulty that Newton–Raphson has with vanishing second derivatives. Its performance is strongly affected by the way this problem is handled (see FHT00, pg.352). L_K_TreeBoost has such difficulties only when (34) is close to zero for *all* observations in a terminal node. This happens much less frequently, and is easier to deal with when it does happen.

Influence trimming for the multi–class procedure is implemented in the same way as that for the two–class case outlined in Section 4.5.1 . Associated with each "observation" $(y_{ik}, \mathbf{x}_i)$ is an influence $w_{ik} = |\tilde{y}_{ik}|\,(1 - |\tilde{y}_{ik}|)$ which is used for deleting observations (27) when inducing the kth tree at the current iteration m.

5 Regularization

In prediction problems, fitting the training data too closely can be counterproductive. Reducing the expected loss on the training data beyond some point causes the population–expected loss to stop decreasing, and often start to increase. Regularization methods attempt to prevent such "over–fitting" by constraining the fitting procedure. For additive expansions (2) a natural regularization parameter is the number of components M. This is analogous to "stepwise" regression where the $\{h(\mathbf{x}; \mathbf{a}_m)\}_1^M$ are considered explanatory variables that are sequentially entered. Controlling the value of M regulates the degree to which expected loss on the training data can be minimized. The best value for M can be estimated by some model selection method, such as using an independent "test" set, or cross–validation.

Regularizing by controlling the number of terms in the expansion places an implicit prior belief that "sparse" approximations involving fewer terms are likely to provide better prediction.

However, it has often been found that regularization through shrinkage provides superior results to that obtained by restricting the number of components (Copas 1983). In the context of additive models (2) constructed in a forward stage–wise manner (9) (10), a simple shrinkage strategy is to replace line 6 of the generic algorithm (Algorithm 1) with

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \rho_m h(\mathbf{x}; \mathbf{a}_m), \quad 0 < \nu \leq 1, \tag{36}$$

and making the corresponding equivalent changes in all of the specific algorithms (Algorithms 2 - 6). Each update is simply scaled by the value of the "learning rate" parameter ν.

Introducing shrinkage into gradient boosting (36) in this manner provides two regularization parameters, the learning rate ν and the number of components M. Each one can control the degree-of-fit and thus affect the best value for other one. Decreasing the value of ν increases the best value for M. Ideally one should estimate optimal values for both by minimizing a model selection criterion jointly with respect to the values of the two parameters. There are also computational considerations; increasing the size of M produces a proportionate increase in computation.

We illustrate this ν-M trade–off through a simulation study. The training sample consists of 5000 observations $\{y_i, \mathbf{x}_i\}$ with

$$y_i = F^*(\mathbf{x}_i) + \varepsilon_i.$$

The target function $F^*(\mathbf{x})$, $\mathbf{x} \in R^{10}$, is randomly generated as described in Section 6.1. The noise ε was generated from a normal distribution with zero mean, and variance adjusted so that

$$E|\varepsilon| = \frac{1}{2} E_{\mathbf{x}} |F^*(\mathbf{x}) - median_{\mathbf{x}} F^*(\mathbf{x})|$$

giving a "signal–to–noise" ratio of 2/1. For this illustration the base learner $h(\mathbf{x}; \mathbf{a})$ is taken to be an 11– terminal node regression tree induced in a best–first manner (FHT00). A general discussion of tree size choice appears in Section 7.

Figure 1 shows the lack–of–fit (LOF) of LS_TreeBoost, LAD_TreeBoost, and L_2_TreeBoost as a function of number of terms (iterations) M, for several values of the shrinkage parameter $\nu \in \{1.0, 0.25, 0.125, 0.06\}$. For the first two methods, LOF is measured by the average absolute error of the estimate $\hat{F}_M(\mathbf{x})$ relative to that of the optimal constant solution

$$A(\hat{F}_M(\mathbf{x})) = \frac{E_{\mathbf{x}} |F^*(\mathbf{x}) - \hat{F}_M(\mathbf{x})|}{E_{\mathbf{x}} |F^*(\mathbf{x}) - median_{\mathbf{x}} F^*(\mathbf{x})|}. \tag{37}$$

For logistic regression the y–values were obtained by thresholding at the median of $F^*(\mathbf{x})$ over the distribution of $\mathbf{x}$–values; $F^*(\mathbf{x}_i)$ values greater than the median were assigned $y_i = 1$, those below the median were assigned $y_i = -1$. The Bayes error rate is thus zero, but the decision boundary is fairly complicated. There are two LOF measures for L_2_TreeBoost; minus twice log–likelihood ("deviance"), and the misclassification error–rate $E_{\mathbf{x}}[1(y \neq sign(\hat{F}_M(\mathbf{x})))]$. The values of all LOF measures were computed by using an independent validation data set of 10000 observations.

As seen in Fig. 1, smaller values of the shrinkage parameter ν (more shrinkage) are seen to result in better performance, although there is a diminishing return for the smallest values. For the larger values, behavior characteristic of over–fitting is observed; performance reaches an optimum at some value of M and thereafter diminishes as M increases beyond that point. This effect is much less pronounced with LAD_TreeBoost, and with the error–rate criterion of L_2_TreeBoost. For smaller values of ν there is less over–fitting, as would be expected.

Although difficult to see except for $\nu = 1$, the misclassification error–rate (lower right panel) continues to decrease well after the logistic likelihood has reached its optimum (lower left panel). Thus, degrading the likelihood by over–fitting actually *improves* misclassification error–rate. Although perhaps counter–intuitive, this is not a contradiction; likelihood and error–rate measure different aspects of fit quality. Error–rate depends only on the sign of $\hat{F}_M(\mathbf{x})$ whereas likelihood

is affected by both its sign and magnitude. Apparently, over–fitting degrades the quality of the magnitude estimate without affecting (and sometimes improving) the sign. Thus, misclassification error is much less sensitive to over–fitting.

Table 1 summarizes the simulation results for several values of ν including those shown in Fig. 1. Shown for each ν–value (row) are the iteration number at which the minimum LOF was achieved and the corresponding minimizing value (pairs of columns).

Table 1
Iteration number giving the best fit, and the best fit value, for several shrinkage parameter ν–values, with three boosting methods.

ν	LS: $A(F_M(\mathbf{x}))$		LAD: $A(F_M(\mathbf{x}))$		L_2: $-2\log(\text{like})$		L_2: error-rate	
1.0	15	0.48	19	0.57	20	0.60	436	0.111
0.5	43	0.40	19	0.44	80	0.50	371	0.106
0.25	77	0.34	84	0.38	310	0.46	967	0.099
0.125	146	0.32	307	0.35	570	0.45	580	0.098
0.06	326	0.32	509	0.35	1000	0.44	994	0.094
0.03	855	0.32	937	0.35	1000	0.45	979	0.097

The ν-M trade–off is clearly evident; smaller values of ν give rise to larger optimal M–values. They also provide higher accuracy, with a diminishing return for $\nu < 0.125$. The misclassification error rate is very flat for $M \gtrsim 200$, so that optimal M–values for it are unstable.

Although illustrated here for just one target function and base learner (11–terminal node tree), the qualitative nature of these results is fairly universal. Other target functions and tree sizes (not shown) give rise to the same behavior. This suggests that the best value for ν depends on the number of iterations M. The latter should be made as large as is computationally convenient or feasible. The value of ν should then be adjusted so that LOF achieves its minimum close to the value chosen for M. If LOF is still decreasing at the last iteration, the value of ν or the number of iterations M should be increased, preferably the latter. Given the sequential nature of the algorithm, it can easily be restarted where it finished previously, so that no computation need be repeated. LOF as a function of iteration number is most conveniently estimated using a left–out test sample.

As illustrated here decreasing the learning rate clearly improves performance, usually dramatically. The reason for this is less clear. Shrinking the model update (36) at each iteration produces a more complex effect than direct proportional shrinkage of the entire model

$$\hat{F}_\nu(\mathbf{x}) = \bar{y} + \nu \cdot (\hat{F}_M(\mathbf{x}) - \bar{y}), \tag{38}$$

where $\hat{F}_M(\mathbf{x})$ is the model induced without shrinkage. The update $\rho_m h(\mathbf{x}; \mathbf{a}_m)$ at each iteration depends on the specific sequence of updates at the previous iterations. Incremental shrinkage (36) produces very different models than global shrinkage (38). Empirical evidence (not shown) indicates that global shrinkage (38) provides at best marginal improvement over no shrinkage, far from the dramatic effect of incremental shrinkage. The mystery underlying the success of incremental shrinkage is currently under investigation.

6 Simulation studies

The performance of any function estimation method depends on the particular problem to which it is applied. Important characteristics of problems that affect performance include training sample size N, true underlying "target" function $F^*(\mathbf{x})$ (1), and the distribution of the departures, ε, of $y \mid \mathbf{x}$ from $F^*(\mathbf{x})$. For any given problem, N is always known and sometimes the distribution of ε is also know, for example when y is binary (Bernoulli). When y is a general real–valued variable the distribution of ε is seldom known. In nearly all cases, the nature of $F^*(\mathbf{x})$ is unknown.

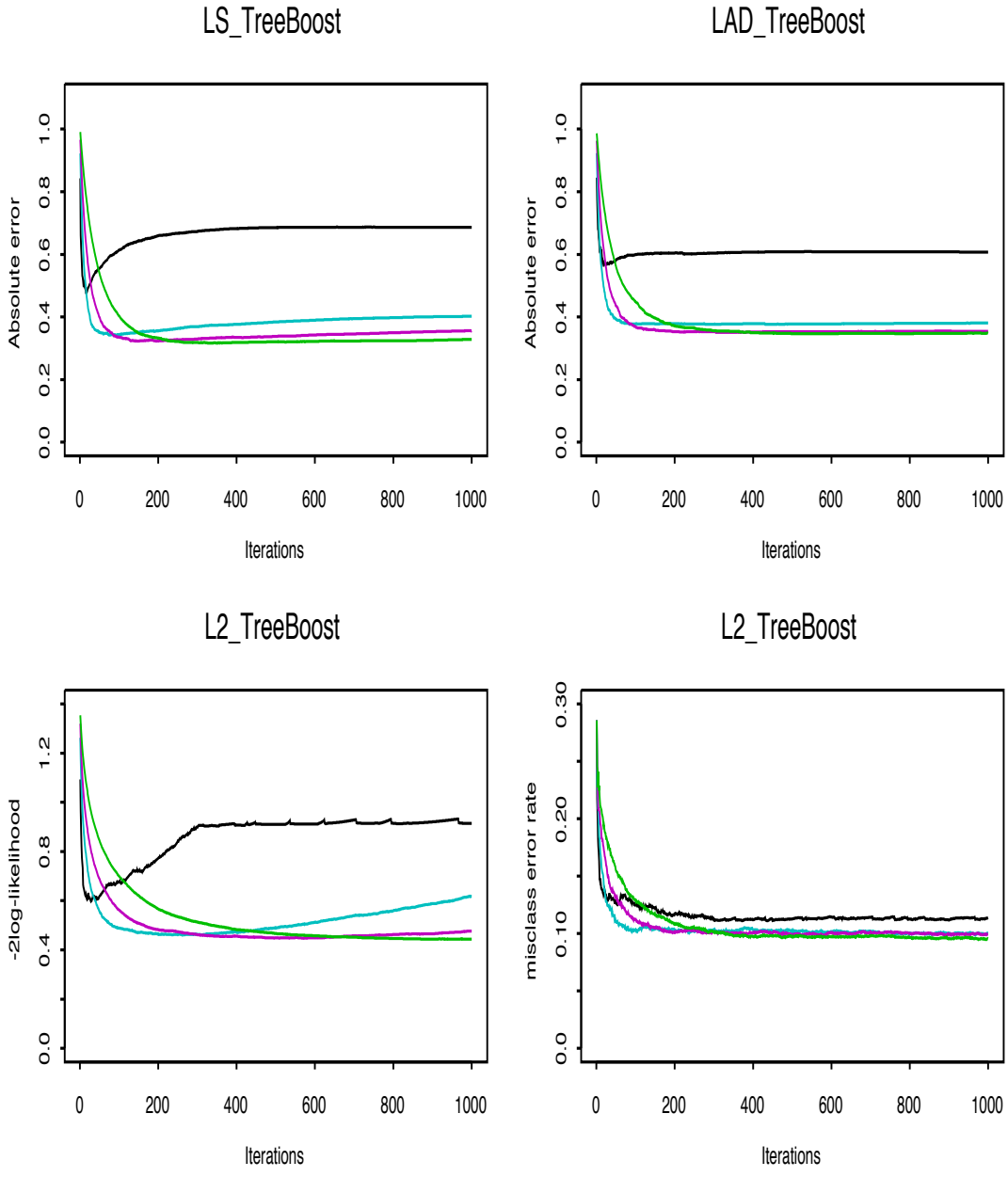

Figure 1: Performance of three gradient boosting algorithms as a function of number of iterations M. The four curves correspond to shrinkage parameter values of $\nu \in \{1.0, 0.25, 0.125, 0.06\}$, and are in that order (top to bottom) at the extreme right of each plot.

In order to gauge the value of any estimation method it is necessary to accurately evaluate its performance over many different situations. This is most conveniently accomplished through Monte Carlo simulation where data can be generated according to a wide variety of prescriptions, and resulting performance accurately calculated. In this section several such studies are presented in an attempt to understand the properties of the various Gradient_TreeBoost procedures developed in the previous sections. Although such a study is far more thorough than evaluating the methods on just a few selected examples, real or simulated, the results of even a large study can only be regarded as suggestive.

6.1 Random function generator

One of the most important characteristics of any problem affecting performance is the true underlying target function $F^*(\mathbf{x})$ (1). Every method has particular targets for which it is most appropriate and others for which it is not. Since the nature of the target function can vary greatly over different problems, and is seldom known, we compare the merits of regression tree gradient boosting algorithms on a variety of different randomly generated targets. Each one takes the form

$$F^*(\mathbf{x}) = \sum_{l=1}^{20} a_l g_l(\mathbf{z}_l). \tag{39}$$

The coefficients $\{a_l\}_1^{20}$ are randomly generated from a uniform distribution $a_l \backsim U[-1, 1]$. Each $g_l(\mathbf{z}_l)$ is a function of a randomly selected subset, of size n_l, of the n–input variables $\mathbf{x}$. Specifically,

$$\mathbf{z}_l = \{x_{P_l(j)}\}_{j=1}^{n_l}$$

where each P_l is a separate random permutation of the integers $\{1, 2, \cdots, n\}$. The size of each subset n_l is itself taken to be random, $n_l = \lfloor 1.5 + r \rfloor$, with r being drawn from an exponential distribution with mean $\lambda = 2$. Thus, the expected number of input variables for each $g_l(\mathbf{z}_l)$ is between three and four. However, most often there will be fewer than that, and somewhat less often, more. This reflects a bias against strong very high order interaction effects. However, for any realized $F^*(\mathbf{x})$ there is a good chance that at least a few of the 20 functions $g_l(\mathbf{z}_l)$ will involve higher order interactions. In any case, $F^*(\mathbf{x})$ will be a function of all, or nearly all, of the input variables.

Each $g_l(\mathbf{z}_l)$ is an n_l–dimensional Gaussian function

$$g_l(\mathbf{z}_l) = \exp(-\frac{1}{2}((\mathbf{z}_l - \mu_l)^T \mathbf{V}_l (\mathbf{z}_l - \mu_l)) \tag{40}$$

where each of the mean vectors $\{\mu_l\}_1^{20}$ is randomly generated from the same distribution as that of the input variables $\mathbf{x}$. The $n_l \times n_l$ covariance matrix $\mathbf{V}_l$ is also randomly generated. Specifically,

$$\mathbf{V}_l = \mathbf{U}_l \mathbf{D}_l \mathbf{U}_l^T$$

where $\mathbf{U}_l$ is a random orthonormal matrix (uniform on Haar measure) and $\mathbf{D}_l = diag\{d_{1l} \cdots d_{n_l l}\}$. The square–roots of the eigenvalues are randomly generated from a uniform distribution $\sqrt{d_{jl}} \backsim U[a, b]$, where the limits a, b depend on the distribution of the input variables $\mathbf{x}$.

For all of the studies presented here the number of input variables was taken to be $n = 10$, and their joint distribution was taken to be standard normal $\mathbf{x} \backsim N(0, \mathbf{I})$. The eigenvalue limits were $a = 0.1$ and $b = 2.0$. Although the tails of the normal distribution are often shorter than that of data encountered in practice, they are still more realistic than uniformly distributed inputs often used in simulation studies. Also, regression trees are immune to the effects of long tailed input variable distributions, so shorter tails gives a relative advantage to competitors in the comparisons.

In the simulation studies below, 100 target functions $F^*(\mathbf{x})$ were randomly generated according to the above prescription (39) (40). Performance is evaluated in terms of the distribution of approximation inaccuracy (relative approximation error (37), or misclassification risk) over these different targets. This approach allows a wide variety of quite different target functions to be generated in terms of the shapes of their contours in the ten–dimensional input space. Although lower order interactions are favored, these functions are not especially well suited to additive regression trees. Decision trees produce tensor product basis functions, and the components $g_l(\mathbf{z}_l)$ of the targets $F^*(\mathbf{x})$ are not tensor product functions. Using the techniques described in Section 8, visualizations of the dependencies of the first randomly generated function on some of its more important arguments are shown in Section 8.3.

Although there are only ten input variables, each target is a function of all of them. In many data mining applications there are many more than ten inputs. However, the relevant dimensionalities are the *intrinsic* dimensionality of the input space, and the number of inputs that actually influence the output response variable y. In problems with many input variables there are usually high degrees of collinearity among many of them, and the number of roughly independent variables (approximate intrinsic dimensionality) is much smaller. Also, target functions often strongly depend only on a small subset of all of the inputs.

6.2 Error distribution

In this section, LS_TreeBoost, LAD_TreeBoost, and M_TreeBoost are compared in terms of their performance over the 100 target functions for two different error distributions. Best–first regression trees with 11 terminal nodes were used with all algorithms. The break–down parameter for the M_TreeBoost was set to its default value $\alpha = 0.9$. The learning rate parameter (36) was set to $\nu = 0.1$ for all TreeBoost procedures in all of the simulation studies.

One hundred data sets $\{y_i, \mathbf{x}_i\}_1^N$ were generated according to

$$y_i = F^*(\mathbf{x}_i) + \varepsilon_i$$

where $F^*(\mathbf{x})$ represents each of the 100 target functions randomly generated as described in Section 6.1. For the first study, the errors ε_i were generated from a normal distribution with zero mean, and variance adjusted so that

$$E|\varepsilon| = E_{\mathbf{x}}|F^*(\mathbf{x}) - median_{\mathbf{x}}F^*(\mathbf{x})| \tag{41}$$

giving a 1/1 signal–to–noise ratio. For the second study the errors were generated from a "slash" distribution, $\varepsilon_i = s \cdot (u/v)$, where $u \backsim N(0,1)$ and $v \backsim U[0,1]$. The scale factor s is adjusted to give a 1/1 signal–to–noise ratio (41). The slash distribution has very thick tails and is often used as an extreme to test robustness. The training sample size was taken to be $N = 7500$, with 5000 used for training, and 2500 left out as a test sample to estimate the optimal number of components M. For each of the 100 trials an additional validation sample of 5000 observations was generated (without error) to evaluate the approximation inaccuracy (37) for that trial.

The left panels of Fig. 2 show boxplots of the distribution of approximation inaccuracy (37) over the 100 targets for the two error distributions for each of the three methods. The shaded area of each boxplot shows the interquartile range of the distribution with the enclosed white bar being the median. The outer hinges represent the points closest to (plus/minus)1.5 interquartile range units from the (upper/lower) quartiles. The isolated bars represent individual points outside this range (outliers).

These plots allow the comparison of the overall distributions, but give no information concerning relative performance for individual target functions. The right two panels of Fig. 2 attempt to provide such a summary. They show distributions of error *ratios*, rather than the errors themselves. For each target function and method, the error for the method on that target is divided by the smallest error obtained on that target, over all of the methods (here three) being compared. Thus, for each of the 100 trials, the best method receives a value of 1.0 and the others receive a larger value. If a particular method was best (smallest error) for all 100 target

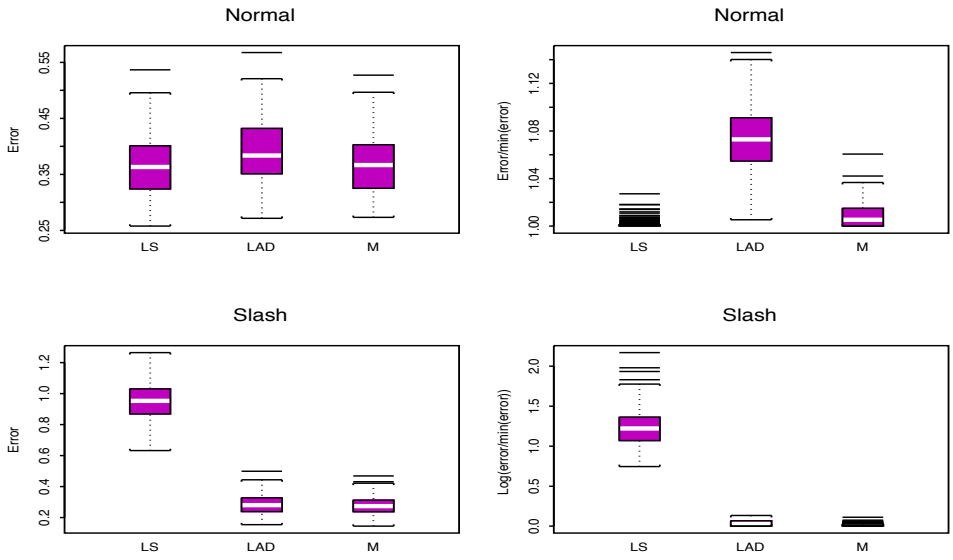

Figure 2: Distribution of absolute approximation error (left panels) and error relative to the best (right panels) for LS_TreeBoost, LAD_TreeBoost and M_TreeBoost for normal and slash error distributions. LS_TreeBoost performs best with the normal error distribution. LAD_TreeBoost and M_TreeBoost both perform well with slash errors. M_TreeBoost is very close to the best for both error distributions. Note the use of logarithmic scale in the lower right panel.

functions, its resulting distribution (boxplot) would be a point mass at the value 1.0. Note that the logarithm of this ratio is plotted in the lower right panel.

From the left panels of Fig. 2 one sees that the 100 targets represent a fairly wide spectrum of difficulty for all three methods; approximation errors vary by over a factor of two. For normally distributed errors LS_TreeBoost is the superior performer, as might be expected. It had the smallest error in 73 of the trials, with M_TreeBoost best the other 27 times. On average LS_TreeBoost was 0.2% worse than the best, M_TreeBoost 0.9% worse, and LAD_TreeBoost was 7.4% worse that the best.

With slash–distributed errors things are reversed. On average the approximation error for LS_TreeBoost was 0.95, thereby explaining only 5% target variation. On individual trials however, it could be much better or much worse. The performance of both LAD_TreeBoost and M_TreeBoost was much better and comparable to each other. LAD_TreeBoost was best 32 times and M_TreeBoost 68 times. On average LAD_TreeBoost was 4.1% worse than the best, M_TreeBoost 1.0% worse, and LS_TreeBoost was 364.6% worse that the best, over the 100 targets.

The results suggest that of these three, M_TreeBoost is the method of choice. In both the extreme cases of very well behaved (normal) and very badly behaved (slash) errors, its performance was very close to that of the best. By comparison, LAD_TreeBoost suffered somewhat with normal errors, and LS_TreeBoost was disastrous with slash errors.

6.3 LS_TreeBoost versus MARS

All Gradient_TreeBoost algorithms produce piecewise–constant approximations. Although the number of such pieces is generally much larger than that produced by a single tree, this aspect of the approximating function $\hat{F}_M(\mathbf{x})$ might be expected to represent a disadvantage with respect

to methods that provide continuous approximations, especially when the true underlying target $F^*(\mathbf{x})$ (1) is continuous and fairly smooth. All of the randomly generated target functions (39) (40) are continuous and very smooth. In this section we investigate the extent of the piecewise–constant disadvantage by comparing the accuracy of Gradient_TreeBoost with that of MARS (Friedman 1991) over these 100 targets. Like TreeBoost, MARS produces a tensor product based approximation. However it uses continuous functions as the product factors, thereby producing a continuous approximation. It also uses a more involved (*stepwise*) strategy to induce the tensor products.

Since MARS is based on least–squares fitting, we compare it to LS_TreeBoost using normally distributed errors, again with a 1/1 signal–to–noise ratio (41). The experimental setup is the same as that in Section 6.2. It is interesting to note that here the performance of MARS was considerably enhanced by using the 2500 observation test set for model selection, rather than its default GCV criterion (Friedman 1991).

The top–left panel of Fig. 3 compares the distribution of MARS average–absolute approximation errors, over the 100 randomly generated target functions (39) (40), to that of LS_TreeBoost from Fig. 2. The MARS distribution is seen to be much broader, varying by almost a factor of three. There were many targets for which MARS did considerably better than LS_TreeBoost, and many for which it was substantially worse. This further illustrates the fact that the nature of the target function strongly influences the relative performance of different methods. The top–right panel of Fig. 3 shows the distribution of errors, relative to the best for each target. The two methods exhibit similar performance based on average–absolute error. There were a number of targets where each one substantially outperformed the other.

The bottom two panels of Fig. 3 show corresponding plots based on root–mean–squared error. This gives proportionally more weight to larger errors in assessing lack–of–performance. For LS_TreeBoost the two error measures have close to the same values for all of the 100 targets. However with MARS, root–mean–squared error is typically 30% higher than average–absolute error. This indicates that MARS predictions tend to be either very close to, or far from, the target. The errors from LS_TreeBoost are more evenly distributed. It tends to have fewer very large errors or very small errors. The latter may be a consequence of the piecewise–constant nature of the approximation which makes it difficult to get arbitrarily close to very smoothly varying targets with approximations of finite size. As Fig. 3 illustrates, relative performance can be quite sensitive to the criterion used to measure it.

These results indicate that the piecewise–constant aspect of TreeBoost approximations is not a serious disadvantage. In the rather pristine environment of normal errors, and normal input variable distributions, it is competitive with MARS. The advantage of the piecewise–constant approach is robustness; specifically, it provides immunity to the adverse effects of wide tails and outliers in the distribution of the input variables $\mathbf{x}$. Methods that produce continuous approximations, such as MARS, can be extremely sensitive to such problems. Also, as shown in Section 6.2, M_TreeBoost (Algorithm 4) is nearly as accurate as LS_TreeBoost for normal errors while, in addition, being highly resistant to *output* y–outliers. Therefore in data mining applications where the cleanliness of the data is not assured and $\mathbf{x}$ and/or y outliers may be present, the relatively high accuracy, consistent performance, and robustness of M_TreeBoost may represent a substantial advantage.

6.4 L_K_TreeBoost versus K–class LogitBoost and AdaBoost.MH

In this section the performance of L_K_TreeBoost is compared to that of K–class LogitBoost (FHT00) and AdaBoost.MH (Schapire and Singer 1998) over the 100 randomly generated targets (Section 6.1). Here $K = 5$ classes are generated by thresholding each target at its 0.2, 0.4, 0.6, and 0.8 quantiles over the distribution of input $\mathbf{x}$–values. There are $N = 7500$ training observations for each trial (1500 per class) divided into 5000 for training and 2500 for model selection (number of iterations, M). An independently generated validation sample of 5000 observations was used to estimate the error–rate for each target. The Bayes error rate is zero for all targets, but the induced decision boundaries can become quite complicated, depending on the nature of each

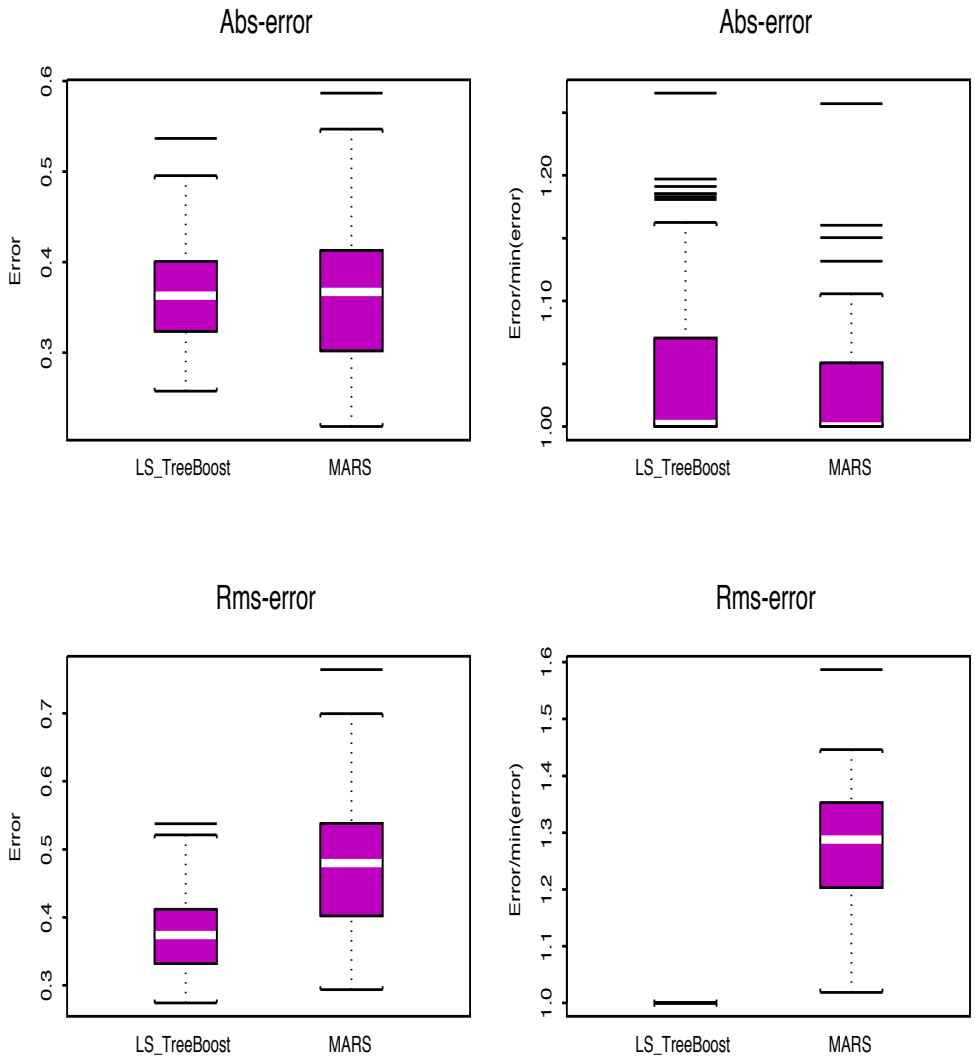

Figure 3: Distribution of approximation error (left panels) and error relative to the best (right panels) for LS_TreeBoost and MARS. The top panels are based on average–absolute–error, whereas the bottom ones use root–mean–squared error. For absolute error the MARS distribution is wider indicating more frequent better and worse performance than LS_TreeBoost. MARS performance as measured by root–mean–squared error is much worse, indicating that it tends to more frequently make both larger and smaller errors than LS_TreeBoost.

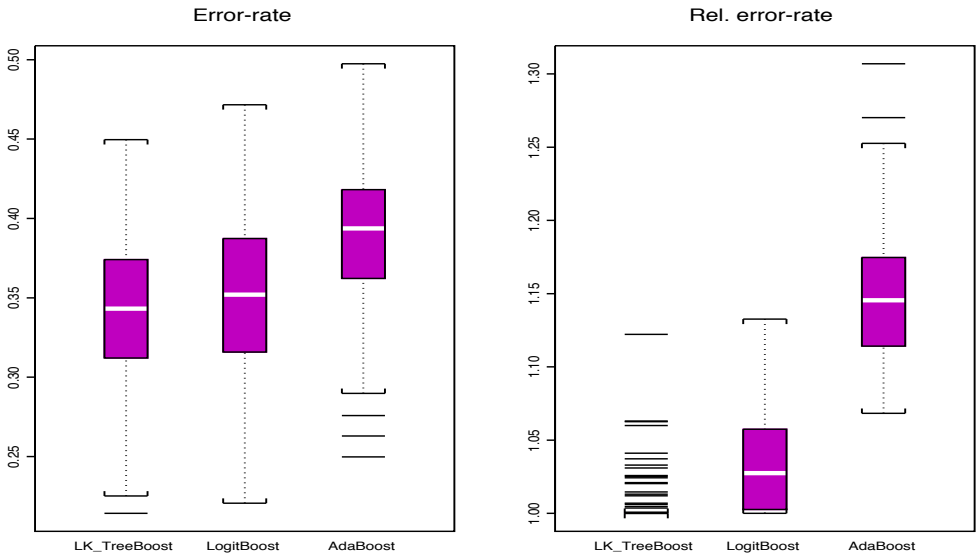

Figure 4: Distribution of error–rate on a five–class problem (left panel) and error–rate relative to the best (right panel) for L_K-TreeBoost, LogitBoost, and AdaBoost.MH. L_K-TreeBoost exhibits superior performance.

individual target function $F^*(\mathbf{x})$. Regression trees with 11 terminal nodes were used for each method.

Figure 4 shows the distribution of error–rate (left panel), and its ratio to the smallest (right panel), over the 100 target functions, for each of the three methods. The error–rate of all three methods is seen to vary substantially over these targets. L_K-TreeBoost is seen to be the generally superior performer. It had the smallest error for 78 of the trials and on average its error–rate was 0.6% higher than the best for each trial. LogitBoost was best on 21 of the targets and there was one tie. Its error–rate was 3.5% higher than the best on average. AdaBoost.MH was never the best performer, and on average it was 15% worse than the best.

Figure 5 shows a corresponding comparison, with the LogitBoost and AdaBoost.MH procedures modified to incorporate incremental shrinkage (36), with the shrinkage parameter set to the same (default) value $\nu = 0.1$ used with L_K-TreeBoost. Here one sees a somewhat different picture. Both LogitBoost and AdaBoost.MH benefit substantially from shrinkage. The performance of all three procedures is now nearly the same, with LogitBoost perhaps having a slight advantage. On average its error–rate was 0.5% worse that the best; the corresponding values for L_K-TreeBoost and AdaBoost.MH were 2.3% and 3.9%, respectively. These results suggest that the relative performance of these methods is more dependent on their aggressiveness, as parameterized by learning rate, than on their structural differences. LogitBoost has an additional internal shrinkage associated with stabilizing its pseudo–response (33) when the denominator is close to zero (FHT00, pg.18). This may account for its slight superiority in this comparison. In fact, when increased shrinkage is applied to L_K-TreeBoost ($\nu = 0.05$) its performance improves, becoming identical to that of LogitBoost shown in Fig. 5. It is likely that when the shrinkage parameter is carefully tuned for each of the three methods, there would be little performance differential between them.

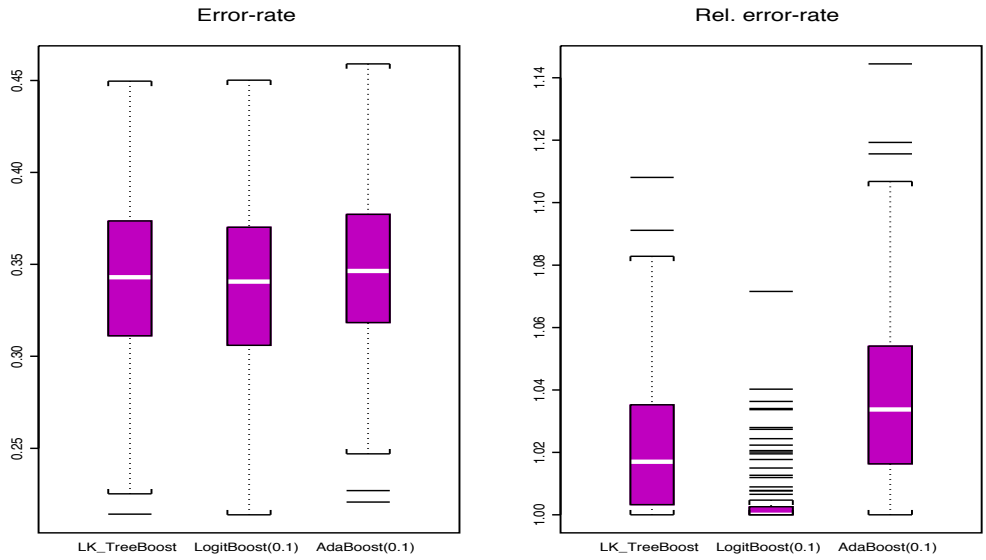

Figure 5: Distribution of error–rate on a five–class problem (left panel), and error–rate relative to the best (right panel), for L_K-TreeBoost, and with proportional shrinkage applied to LogitBoost and RealAdaBoost. Here the performance of all thee methods is similar.

7 Tree boosting

The GradientBoost procedure (Algorithm 1) has two primary meta–parameters, the number of iterations M and the learning rate parameter ν (36). These are discussed in Section 5. In addition to these, there are the meta–parameters associated with the procedure used to estimate the base learner $h(\mathbf{x};\mathbf{a})$. The primary focus of this paper has been on the use of best–first induced regression trees with a fixed number of terminal nodes, J. Thus, J is the primary meta–parameter of this base learner. The best choice for its value depends most strongly on the nature of the target function, namely the highest order of the dominant interactions among the variables.

Consider an "ANOVA" expansion of a function

$$F(\mathbf{x}) = \sum_j f_j(x_j) + \sum_{j,k} f_{jk}(x_j, x_k) + \sum_{j,k,l} f_{jkl}(x_j, x_k, x_l) + \cdots. \tag{42}$$

The first sum is called the "main effects" component of $F(\mathbf{x})$. It consists of a sum of functions that each depend on only one input variable. The particular functions $\{f_j(x_j)\}_1^N$ are those that provide the closest approximation to $F(\mathbf{x})$ under this additive constraint. This is sometimes referred to as an "additive" model because the contributions of each x_j, $f_j(x_j)$, add to the contributions of the others. This is a different and more restrictive definition of "additive" than (2). The second sum consists of functions of pairs of input variables. They are called the two–variable "interaction effects". They are chosen so that along with the main effects they provide the closest approximation to $F(\mathbf{x})$ under the limitation of no more than two–variable interactions. The third sum represents three–variable interaction effects, and so on.

The highest interaction order possible is limited by the number of input variables n. However, especially for large n, many target functions $F^*(\mathbf{x})$ encountered in practice can be closely approximated by ANOVA decompositions of much lower order. Only the first few terms in (42) are required to capture the dominant variation in $F^*(\mathbf{x})$. In fact, considerable success is

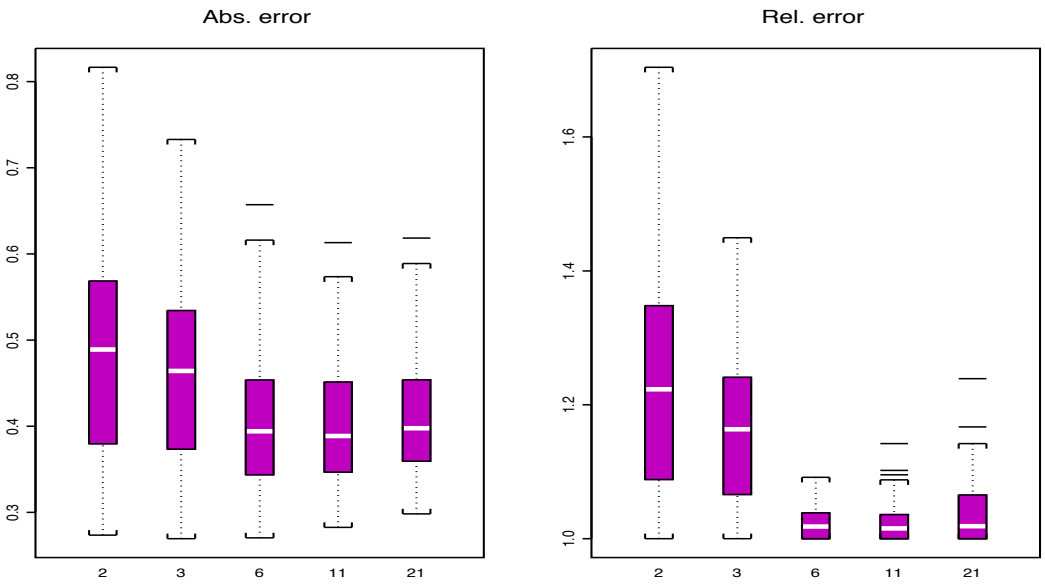

Figure 6: Distribution of absolute approximation error (left panel) and error relative to the best (right panel) for LS_TreeBoost with different sized trees, as measured by number of terminal nodes J. The distribution using the smallest trees $J \in \{2, 3\}$ is wider, indicating more frequent better and worse performance than with the larger trees, all of which have similar performance.

often achieved with the additive component alone (Hastie and Tibshirani 1990). Purely additive approximations are also produced by the "naive"–Bayes method (Warner, Toronto, Veasey and Stephenson 1961), which is often highly successful in classification. These considerations motivated the bias toward lower–order interactions in the randomly generated target functions (Section 6.1) used for the simulation studies.

The goal of function estimation is to produce an approximation $\hat{F}(\mathbf{x})$ that closely matches the target $F^*(\mathbf{x})$. This usually requires that the dominant interaction order of $\hat{F}(\mathbf{x})$ be similar to that of $F^*(\mathbf{x})$. In boosting regression trees, the interaction order can be controlled by limiting the size of the individual trees induced at each iteration. A tree with J terminal nodes produces a function with interaction order at most $\min(J-1, n)$. The boosting process is additive, so the interaction order of the entire approximation can be no larger than the largest among its individual components. Therefore, with any of the TreeBoost procedures, the best tree size J is governed by the effective interaction order of the target $F^*(\mathbf{x})$. This is usually unknown so that J becomes a meta–parameter of the procedure to be estimated using a model selection criterion such as cross–validation or on a left–out subsample not used in training. However, as discussed above, it is unlikely that large trees would ever be necessary or desirable.

Figure 6 illustrates the effect of tree size on approximation accuracy for the 100 randomly generated functions (Section 6.1) used in the simulation studies. The experimental setup is the same as that used in Section 6.2. Shown is the distribution of absolute errors (37) (left panel), and errors relative to the lowest for each target (right panel), for $J \in \{2, 3, 6, 11, 21\}$. The first value $J = 2$ produces additive main effects components only; $J = 3$ produces additive and two–variable interaction terms, and so on. A J terminal node tree can produce interaction levels up to a maximum of $\min(J-1, n)$, with typical values being less than that, especially when $J - 1 \lesssim n$.

As seen in Fig. 6 the smallest trees $J \in \{2, 3\}$ produce lower accuracy on average, but their

distributions are considerably wider that the others. This means that they produce more very accurate, and even more very inaccurate, approximations. The smaller trees, being restricted to low order interactions, are better able to take advantage of targets that happen to be of low interaction level. However, they do quite badly when trying to approximate the high order interaction targets. The larger trees $J \in \{6, 11, 21\}$ are more consistent. They sacrifice some accuracy on low order interaction targets, but do much better on the higher order functions. There is little performance difference among the larger trees, with perhaps some slight deterioration for $J = 21$. The $J = 2$ trees produced the most accurate approximation 8 times; the corresponding numbers for $J \in \{3, 6, 11, 21\}$ were 2, 30, 31, 29 respectively. On average the $J = 2$ trees had errors 23.2% larger than the lowest for each target, while the others had corresponding values of 16.4%, 2.4%, 2.2%, and 3.7% respectively. Higher accuracy should be obtained when the best tree size J is individually estimated for each target. In practice this can be accomplished by evaluating the use of different tree sizes with an independent test data set, as illustrated in Section 9.

8 Interpretation

In many applications it is useful to be able to interpret the derived approximation $\hat{F}(\mathbf{x})$. This involves gaining an understanding of those particular input variables that are most influential in contributing to its variation, and the nature of the dependence of $\hat{F}(\mathbf{x})$ on those influential inputs. To the extent that $\hat{F}(\mathbf{x})$ at least qualitatively reflects the nature of the target function $F^*(\mathbf{x})$ (1), such tools can provide information concerning the underlying relationship between the inputs $\mathbf{x}$ and the output variable y. In this section, several tools are presented for interpreting TreeBoost approximations. Although they can be used for interpreting single decision trees, they tend to be more effective in the context of boosting (especially small) trees. These interpretative tools are illustrated on real data examples in Section 9.

8.1 Relative importance of input variables

Among the most useful descriptions of an approximation $\hat{F}(\mathbf{x})$ are the relative influences I_j, of the individual inputs x_j, on the variation of $\hat{F}(\mathbf{x})$ over the joint input variable distribution. One such measure is

$$I_j = \left(E_{\mathbf{x}} \left[\frac{\partial \hat{F}(\mathbf{x})}{\partial x_j} \right]^2 \cdot var_{\mathbf{x}}[x_j] \right)^{1/2} . \tag{43}$$

For piecewise constant approximations produced by decision trees, (43) does not strictly exist and it must be approximated by a surrogate measure that reflects its properties. Breiman *et al* 1983 proposed

$$\hat{I}_j^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 1(v_t = j) \tag{44}$$

where the summation is over the non–terminal nodes t of the J–terminal node tree T, v_t is the splitting variable associated with node t, and $\hat{\imath}_t^2$ is the corresponding empirical improvement in squared–error (35) as a result of the split. The right–hand–side of (44) is associated with *squared*–influence so that its units correspond to those of (43). Breiman *et al* 1983 used (44) directly as a measure of influence, rather than squared–influence. For a collection of decision trees $\{T_m\}_1^M$, obtained through boosting, (44) can be generalized by its average over all of the trees

$$\hat{I}_j^2 = \frac{1}{M} \sum_{m=1}^{M} \hat{I}_j^2(T_m) \tag{45}$$

in the sequence.

The motivation for (44) (45) is based purely on heuristic arguments. As a partial justification we show that it produces expected results when applied in the simplest context. Consider a linear target function

$$F^*(\mathbf{x}) = a_0 + \sum_{j=1}^{n} a_j x_j \tag{46}$$

where the covariance matrix of the inputs is a multiple of the identity

$$E_{\mathbf{x}}\left[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T\right] = c\mathbf{I}_n.$$

In this case the influence measure (43) produces

$$J_j = |a_j|. \tag{47}$$

Table 2 shows the results of a small simulation study similar to those in Section 6, but with $F^*(\mathbf{x})$ taken to be linear (46) with coefficients

$$a_j = (-1)^j j, \tag{48}$$

and a signal to noise ratio of 1/1 (41). Shown are the mean and standard deviation of the values of (44) (45) over ten random samples, all with $F^*(\mathbf{x})$ given by (46) (48). The influence of the estimated most influential variable x_{j^*} is arbitrarily assigned the value $I_{j^*} = 100$, and the estimated values of the others scaled accordingly. The estimated importance ranking of the input variables was correct on every one of the ten trials. As can be seen in Table 2, the estimated relative influence values are consistent with those given by (47) and (48).

Table 2

Estimated mean and standard deviation of input variable relative influence for a linear target function.

Var.	Mean	Std.
10	100.0	0.0
9	90.3	4.3
8	80.0	4.1
7	69.8	3.9
6	62.1	2.3
5	51.7	2.0
4	40.3	4.2
3	31.3	2.9
2	22.2	2.8
1	13.0	3.2

In Breiman *et al* 1983, the influence measure (44) is augmented by a strategy involving surrogate splits intended to uncover the masking of influential variables by others highly associated with them. This strategy is most helpful with *single* decision trees where the opportunity for variables to participate in splitting is limited by the size J of the tree in (44). In the context of boosting however, the number of splitting opportunities is vastly increased (45), and surrogate unmasking is correspondingly less essential.

In K–class logistic regression and classification (Section 4.6) there are K (logistic) regression functions $\{F_{kM}(\mathbf{x})\}_{k=1}^{K}$, each described by a sequence of M trees. In this case (45) generalizes

to

$$\hat{I}_{jk}^2 = \frac{1}{M}\sum_{m=1}^{M} \hat{I}_j^2(T_{km}) \tag{49}$$

where T_{km} is the tree induced for the kth class at iteration m. The quantity $\hat{I}_{jk}$ can be interpreted as the relevance of predictor variable x_j in separating class k from the other classes. The overall relevance of x_j can be obtained by averaging over all classes

$$\hat{I}_j = \frac{1}{K}\sum_{k=1}^{K} \hat{I}_{jk}.$$

However, the individual $\hat{I}_{jk}$ themselves can be quite useful. It is often the case that different subsets of variables are highly relevant to different subsets of classes. This more detailed knowledge can lead to insights not obtainable by examining only overall relevance.

8.2 Partial dependence plots

Visualization is one of the most powerful interpretational tools. Graphical renderings of the value of $\hat{F}(\mathbf{x})$ as a function of its arguments provides a comprehensive summary of its dependence on the joint values of the input variables. Unfortunately, such visualization is limited to low dimensional arguments. Functions of a single real–valued variable x, $\hat{F}(x)$, can be plotted as a graph of the values of $\hat{F}(x)$ against each corresponding value of x. Functions of a single categorical variable can be represented by a bar–plot, each bar representing one of its values, and the bar height the value of the function. Functions of two real–valued variables can be pictured using contour or perspective mesh plots. Functions of a categorical variable and another variable (real or categorical) are best summarized by a sequence of ("trellis") plots, each one showing the dependence of $\hat{F}(\mathbf{x})$ on the second variable, conditioned on the respective values of the first variable (Becker and Cleveland 1996).

Viewing functions of higher dimensional arguments is more difficult. It is therefore useful to be able to view the partial dependence of the approximation $\hat{F}(\mathbf{x})$ on selected small subsets of the input variables. Although a collection of such plots can seldom provide a comprehensive depiction of the approximation, it can often produce helpful clues, especially when $\hat{F}(\mathbf{x})$ is dominated by low order interactions (Section 7).

Let $\mathbf{z}_l$ be a chosen "target" subset, of size l, of the input variables $\mathbf{x}$

$$\mathbf{z}_l = \{z_1, \cdots, z_l\} \subset \{x_1, \cdots, x_n\},$$

and $\mathbf{z}_{\setminus l}$ be the complement subset

$$\mathbf{z}_{\setminus l} \cup \mathbf{z}_l = \mathbf{x}.$$

The approximation $\hat{F}(\mathbf{x})$ in principle depends on variables in both subsets

$$\hat{F}(\mathbf{x}) = \hat{F}(\mathbf{z}_l\,,\,\mathbf{z}_{\setminus l}).$$

If one conditions on specific values for the variables in $\mathbf{z}_{\setminus l}$, then $\hat{F}(\mathbf{x})$ can be considered as a function only of the variables in the chosen subset $\mathbf{z}_l$

$$\hat{F}_{\mathbf{z}_{\setminus l}}(\mathbf{z}_l) = \hat{F}(\mathbf{z}_l \,|\, \mathbf{z}_{\setminus l}). \tag{50}$$

In general, the functional form of $\hat{F}_{\mathbf{z}_{\setminus l}}(\mathbf{z}_l)$ will depend on the particular values chosen for $\mathbf{z}_{\setminus l}$. If however, this dependence is not too strong then the average function

$$\bar{F}_l(\mathbf{z}_l) = E_{\mathbf{z}_{\setminus l}}[\hat{F}(\mathbf{x})] = \int \hat{F}(\mathbf{z}_l\,,\,\mathbf{z}_{\setminus l})\, p_{\setminus l}(\mathbf{z}_{\setminus l})\, d\mathbf{z}_{\setminus l} \tag{51}$$

can represent a useful summary of the "partial dependence" of $\hat{F}(\mathbf{x})$ on the chosen variable subset $\mathbf{z}_l$. Here $p_{\setminus l}(\mathbf{z}_{\setminus l})$ is the marginal probability density of $\mathbf{z}_{\setminus l}$

$$p_{\setminus l}(\mathbf{z}_{\setminus l}) = \int p(\mathbf{x})\, d\mathbf{z}_l, \tag{52}$$

where $p(\mathbf{x})$ is the joint density of all of the inputs $\mathbf{x}$. This complement marginal density (52) can be estimated from the training data, so that (51) becomes

$$\bar{F}_l(\mathbf{z}_l) = \frac{1}{N}\sum_{i=1}^{N} \hat{F}(\mathbf{z}_l\,,\, \mathbf{z}_{i,\setminus l}). \tag{53}$$

In the special cases where the dependence of $\hat{F}(\mathbf{x})$ on $\mathbf{z}_l$ is additive

$$\hat{F}(\mathbf{x}) = \hat{F}_l(\mathbf{z}_l) + \hat{F}_{\setminus l}(\mathbf{z}_{\setminus l}), \tag{54}$$

or multiplicative

$$\hat{F}(\mathbf{x}) = \hat{F}_l(\mathbf{z}_l) \cdot \hat{F}_{\setminus l}(\mathbf{z}_{\setminus l}), \tag{55}$$

the *form* of $\hat{F}_{\mathbf{z}_{\setminus l}}(\mathbf{z}_l)$ (50) does not depend on the joint values of the compliment variables $\mathbf{z}_{\setminus l}$. Then $\bar{F}_l(\mathbf{z}_l)$ (51) provides a complete description of the nature of the variation of $\hat{F}(\mathbf{x})$ on the chosen input variable subset $\mathbf{z}_l$.

An alternative way of summarizing the dependence of $\hat{F}(\mathbf{x})$ on a subset $\mathbf{z}_l$ is to directly model $\hat{F}(\mathbf{x})$ as a function of $\mathbf{z}_l$ on the training data

$$\tilde{F}_l(\mathbf{z}_l) = E_{\mathbf{x}}[\hat{F}(\mathbf{x}) \,|\, \mathbf{z}_l] = \int \hat{F}(\mathbf{x})\, p(\mathbf{z}_{\setminus l} \,|\, \mathbf{z}_l)\, d\mathbf{z}_{\setminus l}. \tag{56}$$

However, averaging over the conditional density in (56), rather than the marginal density in (51), causes $\tilde{F}_l(\mathbf{z}_l)$ to reflect not only the dependence of $\hat{F}(\mathbf{x})$ on the selected variable subset $\mathbf{z}_l$, but in addition, apparent dependencies induced solely by the associations between them and the complement variables $\mathbf{z}_{\setminus l}$. For example, if the contribution of $\mathbf{z}_l$ happens to be additive (54) or multiplicative (55), $\tilde{F}_l(\mathbf{z}_l)$ (56) would not evaluate to the corresponding term or factor $\hat{F}_l(\mathbf{z}_l)$, unless the joint density $p(\mathbf{x})$ happened to be the product

$$p(\mathbf{x}) = p_l(\mathbf{z}_l) \cdot p_{\setminus l}(\mathbf{z}_{\setminus l}). \tag{57}$$

Partial dependence functions (51) can be used to help interpret models produced by any "black box" prediction method, such as neural networks, support vector machines, nearest–neighbors, radial basis functions, etc. When there are a large number of predictor variables, it is very useful to have an measure of relevance (Section 8.1) to reduce the potentially large number variables, and variable combinations, to be considered. Also, a pass over the data (53) is required to evaluate each $\bar{F}_l(\mathbf{z}_l)$ for each set of joint values $\mathbf{z}_l$ of its argument. This can be time consuming for large data sets, although subsampling could help somewhat.

For regression trees based on single–variable splits, however, the partial dependence of $\hat{F}(\mathbf{x})$ on a specified target variable subset $\mathbf{z}_l$ (51) is straightforward to evaluate given only the tree, *without* reference to the data itself (53). For a specific set of values for the variables $\mathbf{z}_l$, a weighted traversal of the tree is performed. At the root of the tree, a weight value of one is assigned. For each non–terminal node visited, if its split variable is in the target subset $\mathbf{z}_l$, the appropriate left or right daughter node is visited and the weight is not modified. If the node's split variable is a member of the complement subset $\mathbf{z}_{\setminus l}$, then both daughters are visited and the current weight is multiplied by the fraction of training observations that went left or right respectively at that node.

Each terminal node visited during the traversal is assigned the current value of the weight. When the tree traversal is complete, the value of $\bar{F}_l(\mathbf{z}_l)$ is the corresponding weighted average

of the $\hat{F}(\mathbf{x})$ values over those terminal nodes visited during the tree traversal. For a collection of M regression trees, obtained through boosting, the results for the individual trees are simply averaged.

For purposes of interpretation through graphical displays, input variable subsets of low cardinality ($l \leq 2$) are most useful. The most informative such subsets would likely be comprised of the input variables deemed to be among the most influential (44) (45) in contributing to the variation of $\hat{F}(\mathbf{x})$. Illustrations are provided in Sections 8.3 and 9.

The closer the dependence of $\hat{F}(\mathbf{x})$ on the subset $\mathbf{z}_l$ is to being additive (54) or multiplicative (55), the more completely the partial dependence function $\bar{F}_l(\mathbf{z}_l)$ (51) captures the nature of the influence of the variables in $\mathbf{z}_l$ on the derived approximation $\hat{F}(\mathbf{x})$. Therefore, subsets $\mathbf{z}_l$ that group together those influential inputs that have complex (nonfactorable (55)) interactions between them will provide the most reveling partial dependence plots. As a diagnostic, both $\bar{F}_l(\mathbf{z}_l)$ and $\bar{F}_l(\mathbf{z}_{\setminus l})$ can be separately computed for candidate subsets. The value of the multiple correlation over the training data between $\hat{F}(\mathbf{x})$ and $\{\bar{F}_l(\mathbf{z}_l), \bar{F}_{\setminus l}(\mathbf{z}_{\setminus l})\}$ and/or $\bar{F}_l(\mathbf{z}_l) \cdot \bar{F}_{\setminus l}(\mathbf{z}_{\setminus l})$ can be used to gauge the degree of additivity and/or factorability of $\hat{F}(\mathbf{x})$ with respect to a chosen subset $\mathbf{z}_l$. As an additional diagnostic, $\hat{F}_{\mathbf{z}_{\setminus l}}(\mathbf{z}_l)$ (50) can be computed for a small number of $\mathbf{z}_{\setminus l}$–values randomly selected from the training data. The resulting functions of $\mathbf{z}_l$ can be compared to $\bar{F}_l(\mathbf{z}_l)$ to judge the variability of the partial dependence of $\hat{F}(\mathbf{x})$ on $\mathbf{z}_l$, with respect to changing values of $\mathbf{z}_{\setminus l}$.

In K–class logistic regression and classification (Section 4.6) there are K (logistic) regression functions $\{F_k(\mathbf{x})\}_{k=1}^{K}$. Each is logarithmically related to $p_k(\mathbf{x}) = \Pr(y = k \,|\, \mathbf{x})$ through (29). Larger values of $F_k(\mathbf{x})$ imply higher probability of observing class k at $\mathbf{x}$. Partial dependence plots of each $F_k(\mathbf{x})$ on variable subsets $\mathbf{z}_l$ most relevant to that class (49) provide information on how the input variables influence the respective individual class probabilities.

8.3 Randomly generated function

In this section the interpretational tools described in the preceding two sections are applied to the first (of the 100) randomly generated functions (Section 6.1) used for the Monte Carlo studies of Section 6.

Figure 7 shows the estimated relative importance (44) (45) of the 10 input predictor variables. Some are seen to be more influential than others, but no small subset appears to dominate. This is consistent with the mechanism used to generate these functions.

Figure 8 displays single variable ($l = 1$) partial dependence plots (53) on the six most influential variables. The hash marks at the base of each plot represent the deciles of the corresponding predictor variable distribution. The piecewise–constant nature of the approximation is evident. Unlike most approximation methods, there is no explicit smoothness constraint imposed upon TreeBoost models. Arbitrarily sharp discontinuities can be accommodated. The generally smooth trends exhibited in these plots suggest that a smooth approximation best describes this target. This is again consistent with the way these functions were generated.

Figure 9 displays two–variable ($l = 2$) partial dependence plots on some of the more influential variables. Interaction effects of varying degrees are indicated among these variable pairs. This is in accordance with the way in which these target functions were actually generated (39) (40).

Given the general complexity of these generated targets as a function of their arguments, it is unlikely that one would ever be able to uncover their complete detailed functional form through a series of such partial dependence plots. The goal is to obtain an understandable description of some of the important aspects of the functional relationship. In this example the target function was generated from a known prescription, so that at least qualitatively we can verify that this is the case here.

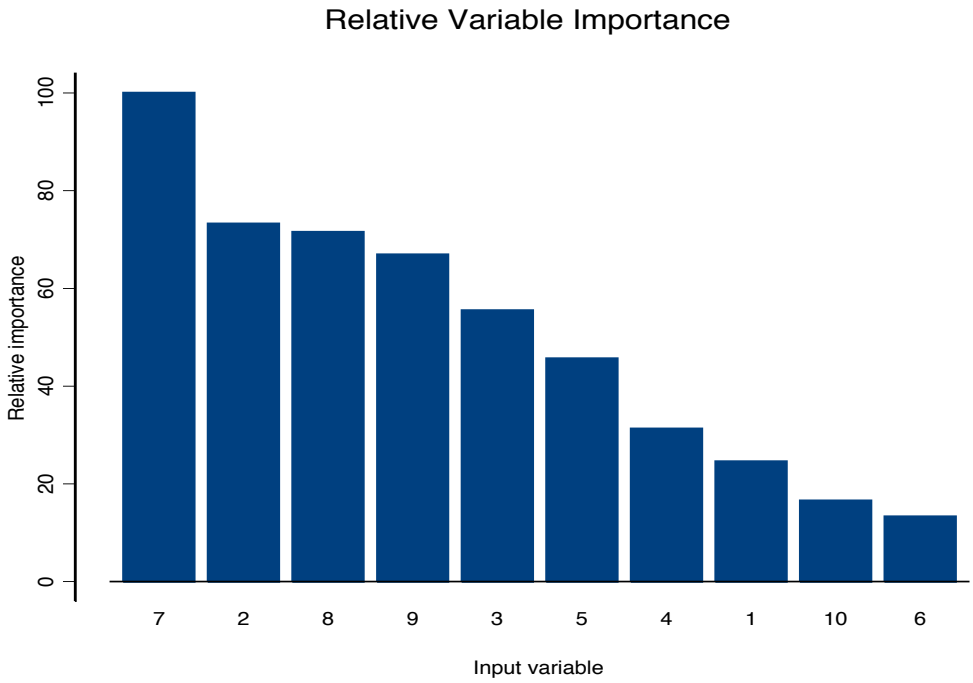

Figure 7: Relative importance of the input predictor variables for the first randomly generated function used in the Monte Carlo studies.

9 Real data

In this section the TreeBoost regression algorithms are illustrated on two moderate sized data sets. The results in Section 6.4 suggest that the properties of the classification algorithm L_K-TreeBoost are very similar to those of LogitBoost, which was extensively applied to data in FHT00. The first (scientific) data set consists of chemical concentration measurements on rock samples, and the second (demographic) is sample survey questionnaire data. Both data sets were partitioned into a learning sample consisting of two–thirds of the data, with the remaining data being used as a test sample for choosing the model size (number of iterations M). The shrinkage parameter (36) was set to $\nu = 0.1$.

9.1 Garnet data

This data set consists of a sample of $N = 13317$ garnets collected from around the world (Griffin *et al* 1997). A garnet is a complex Ca–Mg–Fe–Cr silicate that commonly occurs as a minor phase in rocks making up the Earth's mantle. The variables associated with each garnet are the concentrations of various chemicals and the tectonic plate setting where the rock was collected

$$(TiO_2,\ Cr_2O_3,\ FeO,\ MnO,\ MgO,\ CaO,\ Zn,Ga,\ Sr,Y,\ Zr,\ tec).$$

The first eleven variables representing concentrations are real–valued. The last variable (tec) takes on three categorical values: "ancient stable shields", "Proterozoic shield areas", and "young orogenic belts". There are no missing values in these data, but the distribution of many of the variables tend to be highly skewed toward larger values, with many outliers.

The purpose of this exercise is to estimate the concentration of Titanium (TiO_2) as a function of the joint concentrations of the other chemicals and the tectonic plate index.

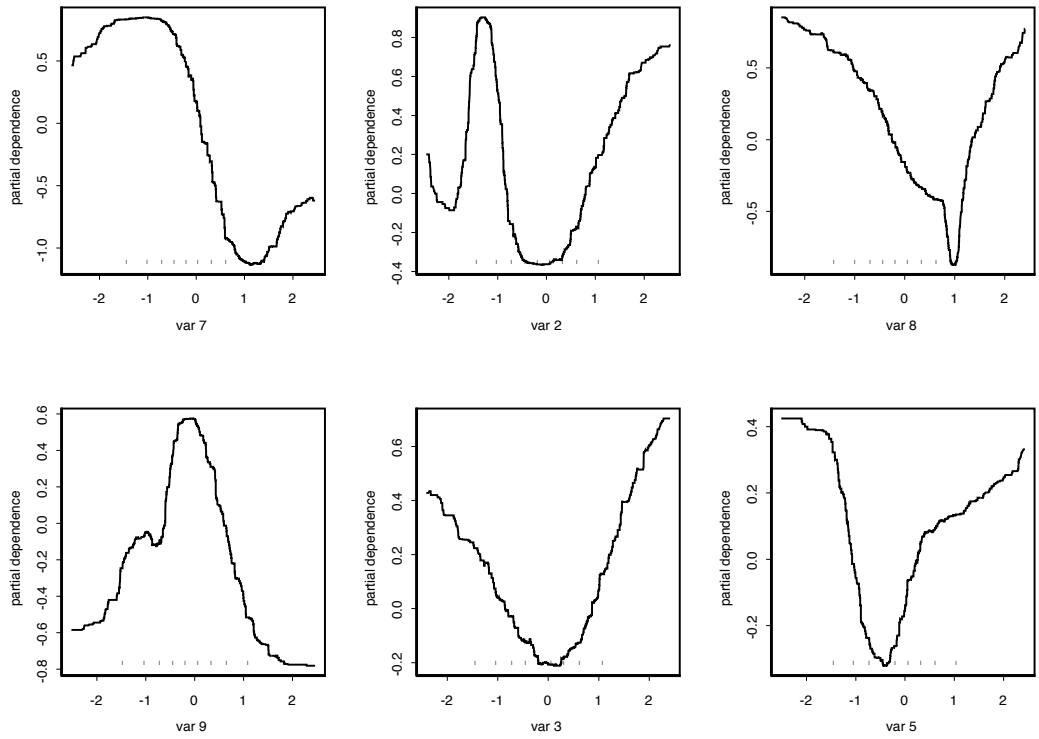

Figure 8: Single variable partial dependence plots for the six most influential predictor variables for the first randomly generated function used in the simulation studies.

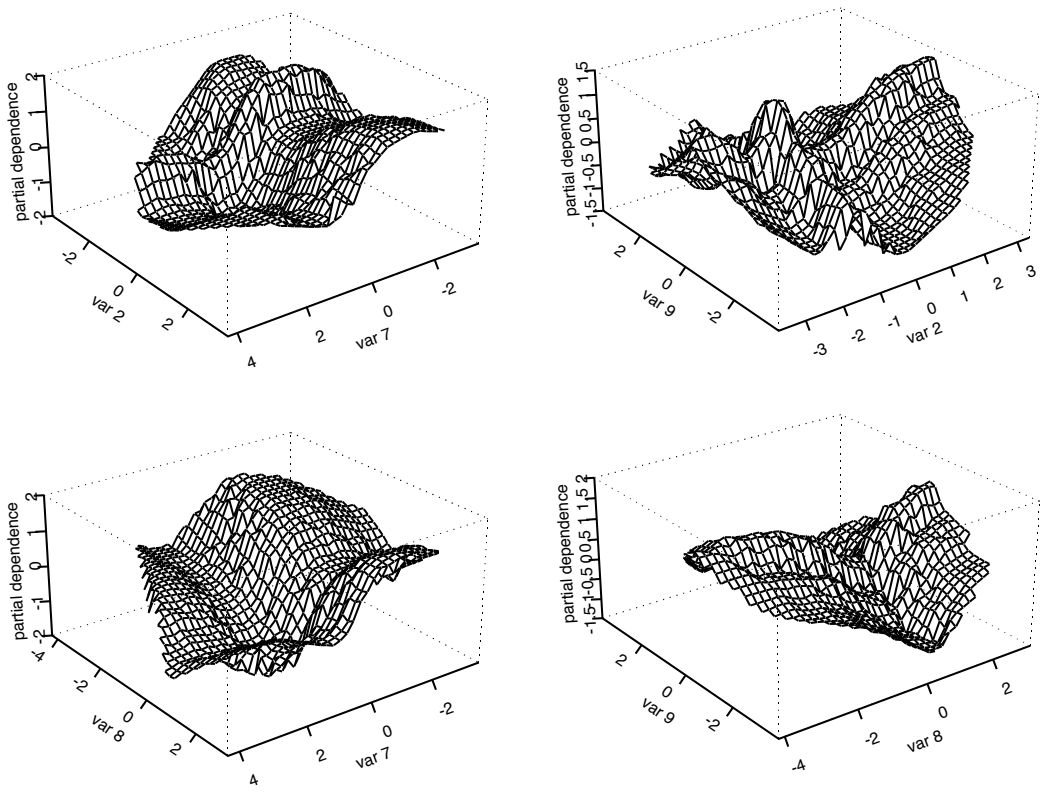

Figure 9: Two–variable partial dependence plots on a few of the important predictor variables for the first randomly generated function used in the simulation studies.

Table 3

Average–absolute error of LS_TreeBoost, LAD_TreeBoost, and M_TreeBoost on the garnet data for varying numbers of terminal nodes in the individual trees.

Term. nodes	LS	LAD	M
2	0.58	0.57	0.57
3	0.48	0.47	0.46
4	0.49	0.45	0.45
6	0.48	0.44	0.43
11	0.47	0.44	0.43
21	0.46	0.43	0.43

Table 3 shows the average–absolute error in predicting the output y–variable, relative to the optimal constant prediction,

$$A(y, \hat{F}(\mathbf{x})) = \frac{E_{y,\mathbf{x}} \left| y - \hat{F}(\mathbf{x}) \right|}{E_y \left| y - median(y) \right|}, \tag{58}$$

based on the test sample, for LS_TreeBoost, LAD_TreeBoost, and M_TreeBoost for several values of the size (number of terminal nodes) J of the constituent trees. Note that this prediction error measure (58) includes the additive irreducible error associated with the (unknown) underlying target function $F^*(\mathbf{x})$ (1). This irreducible error adds same amount to all entries in Table 3. Thus, differences in those entries reflect a proportionally greater improvement in approximation error (37) on the target function itself.

For all three methods the additive ($J = 2$) approximation is distinctly inferior to that using larger trees, indicating the presence of interaction effects (Section 7) among the input variables. Six terminal node trees are seen to be adequate and using only three terminal node trees is seen to provide accuracy within 10% of the best. The errors of LAD_TreeBoost and M_TreeBoost are smaller than those of LS_TreeBoost and similar to each other, with perhaps M_TreeBoost having a slight edge. These results are consistent with those obtained in the simulation studies as shown in Fig. 2 and Fig. 6.

Figure 10 shows the relative importance (44) (45) of the 11 input variables in predicting TiO_2 concentration based on the M_TreeBoost approximation using six terminal node trees. Results are very similar for the other models in Table 3 with similar errors. Ga and Zr are seen to be the most influential with MnO being somewhat less important. The top three panels of Fig. 11 show the partial dependence (51) of the approximation $\hat{F}(\mathbf{x})$ on these three most influential variables. The bottom three panels show the partial dependence of $\hat{F}(\mathbf{x})$ on the three pairings of these variables. A strong interaction effect between Ga and Zr is clearly evident. $\hat{F}(\mathbf{x})$ has very little dependence on either variable when the other takes on its smallest values. As the value of one of them is increased, the dependence of $\hat{F}(\mathbf{x})$ on the other is correspondingly amplified. A somewhat smaller interaction effect is seen between MnO and Zr.

9.2 Demographic data

This data set consists of $N = 9409$ questionnaires filled out by shopping mall customers in the San Francisco Bay Area (Impact Resources, Inc, Columbus OH 1987). Here we use answers to the first 14 questions, relating to demographics, for illustration. These questions are listed in Table 4. The data are seen to consist of a mixture of real and categorical variables, each with a small numbers of distinct values. There are many missing values.

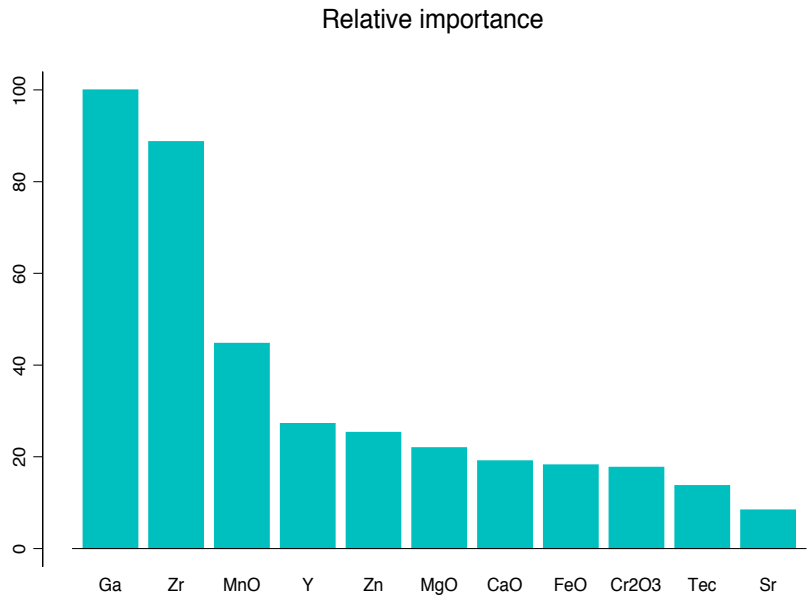

Figure 10: Relative influence of the eleven input variables on the target variation for the garnet data. Ga and Zr are much more influential that the others.

Table 4

Variables for the demographic data.

Var	Demographic	#values	Type
1	sex	2	cat
2	martial status	5	cat
3	age	7	real
4	education	6	real
5	occupation	9	cat
6	income	9	real
7	years in BA	5	real
8	dual incomes	2	cat
9	number in household	9	real
10	number in household <18	9	real
11	householder status	3	cat
12	type of home	5	cat
13	ethnic classification	8	cat
14	language in home	3	cat

We illustrate TreeBoost on these data by modeling income as a function of the other 13 variables. Table 5 shows the average–absolute error in predicting income, relative to the best constant predictor (58), for the three regression TreeBoost algorithms.

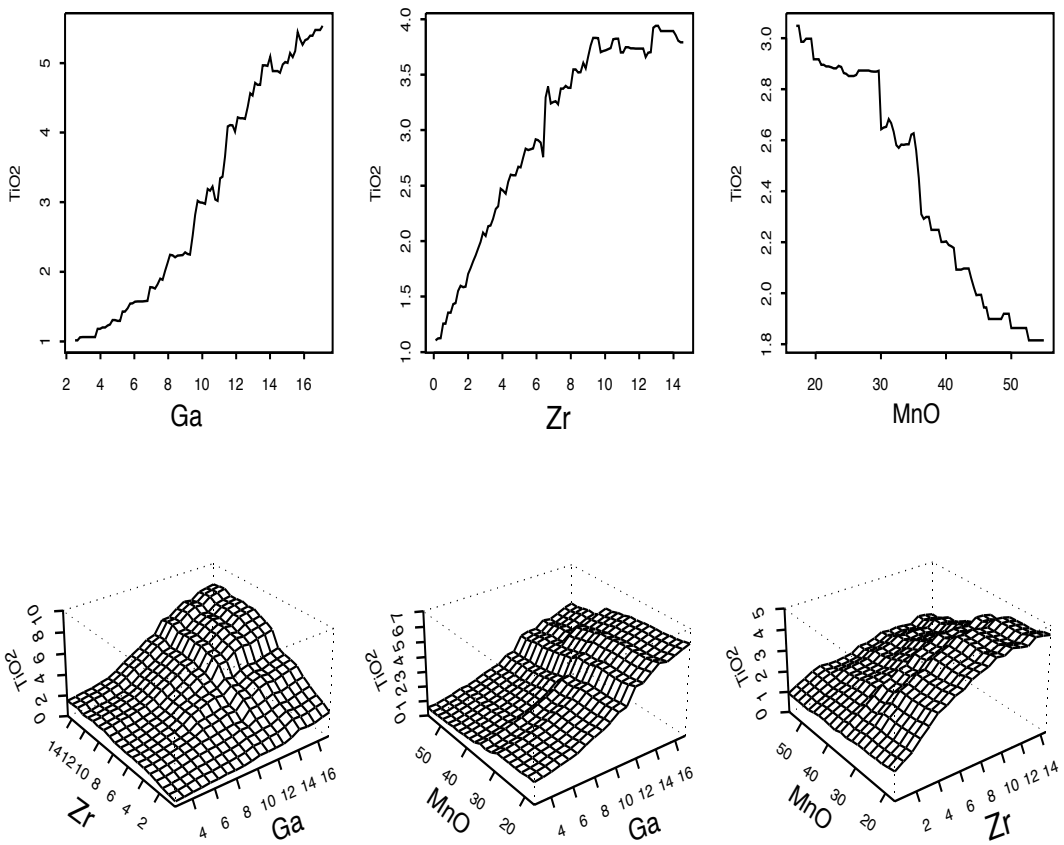

Figure 11: Partial dependence plots for the three most influential input variables in the garnet data. Note the different vertical scales for each plot. There is a strong interaction effect between Zr and Ga, and a somewhat weaker one between Zr and MnO.

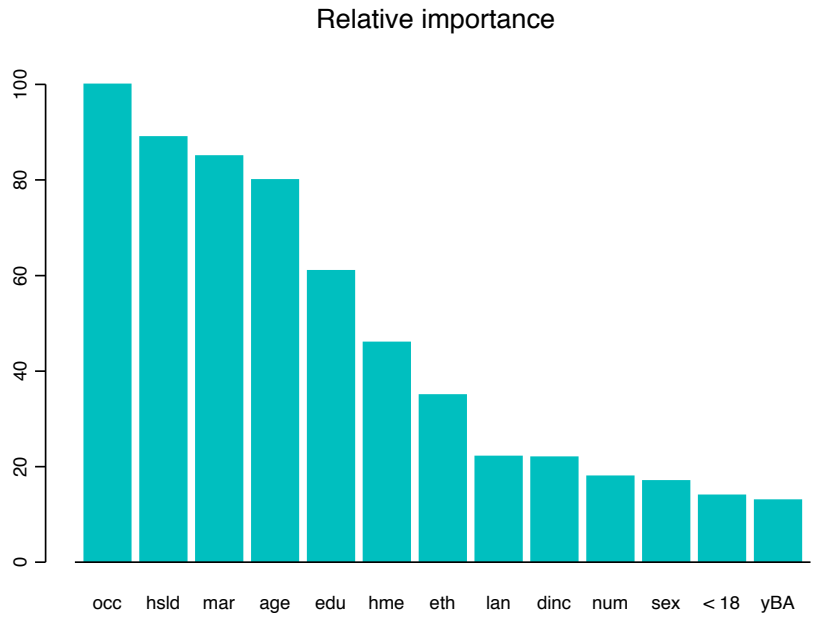

Figure 12: Relative influence of the 13 input variables on the target variation for the demographic data. No small group of variables dominate.

Table 5

Average–absolute error of LS_TreeBoost, LAD_TreeBoost, and M_TreeBoost on the demographic data for varying numbers of terminal nodes in the individual trees.

Term. nodes	LS	LAD	M
2	0.60	0.63	0.61
3	0.60	0.62	0.59
4	0.59	0.59	0.59
6	0.59	0.58	0.59
11	0.59	0.57	0.58
21	0.59	0.58	0.58

There is little difference in performance among the three methods. Owing to the highly discrete nature of these data, there are no outliers or long tailed distributions among the real valued inputs or the output y. There is also very little reduction in error as the constituent tree size J is increased, indicating lack of interactions among the input variables; an approximation additive in the individual input variables ($J = 2$) seems to be adequate.

Figure 12 shows the relative importance of the input variables in predicting income, based on the ($J = 2$) LS_TreeBoost approximation. There is no small subset of them that dominates. Figure 13 shows partial dependence plots on the six most influential variables. Those for the categorical variables are represented as bar–plots, and all plots are centered to have zero mean over the data. Since the approximation consists of main effects only (first sum in (42)), these plots completely describe the corresponding contributions $f_j(x_j)$ of each of these inputs.

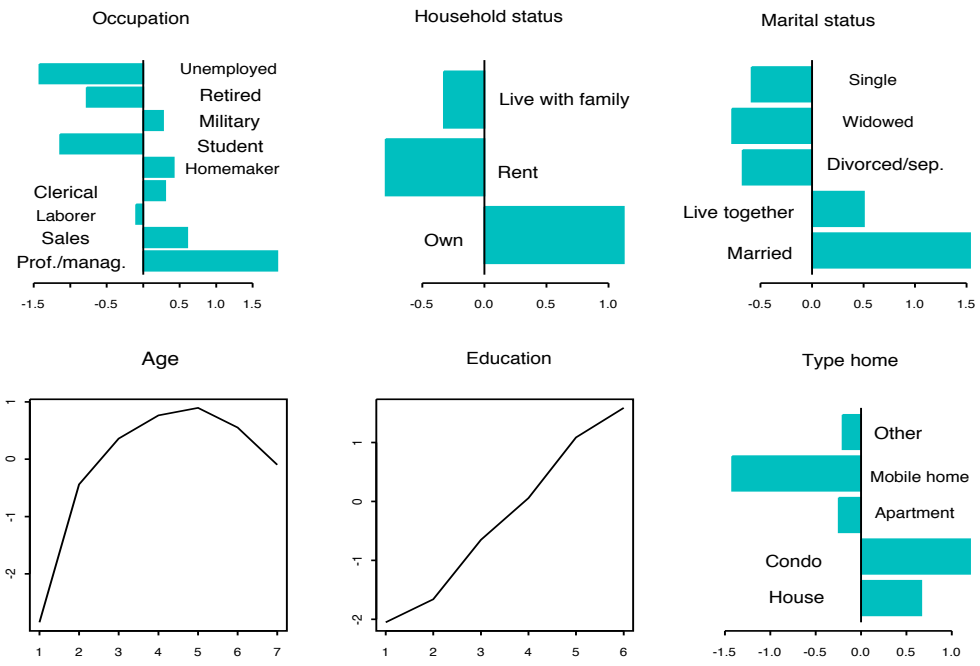

Figure 13: Partial dependence plots for the six most influential input variables in the demographic data. Note the different vertical scales for each plot. The abscissa values for age and education are codes representing consecutive equal intervals. The dependence of income on age is nonmonotonic reaching a maximum at the value 5, representing the interval 45 - 54 years old.

There do not appear to be any surprising results in Fig. 13. The dependencies for the most part confirm prior suspicions and suggest that the approximation is intuitively reasonable.

10 Data Mining

As "off–the–shelf" tools for predictive data mining, the TreeBoost procedures have some attractive properties. They inherit the favorable characteristics of trees while mitigating many of the unfavorable ones. Among the most favorable is robustness. All TreeBoost procedures are invariant under all (strictly) monotone transformations of the individual input variables. For example, using x_j, $\log x_j$, e^{x_j}, or x_j^a as the jth input variable yields the same result. Thus, the need for considering input variable transformations is eliminated. As a consequence of this invariance, sensitivity to long tailed distributions and outliers is also eliminated. In addition, LAD_TreeBoost is completely robust against outliers in the *output* variable y as well. M_TreeBoost also enjoys a fair measure of robustness against output outliers.

Another advantage of decision tree induction is internal feature selection. Trees tend to be quite robust against the addition of irrelevant input variables. Also, tree based models handle missing values in a unified and elegant manner (Breiman *et al* 1983). There is no need to consider external imputation schemes. TreeBoost clearly inherits these properties as well.

The principal disadvantage of single tree models is inaccuracy. This is a consequence of the coarse nature of their piecewise–constant approximations - especially for smaller trees, instability - especially for larger trees, and the fact that they involve predominately high order interactions. All of these are mitigated by boosting. TreeBoost procedures produce piecewise–constant approximations, but the granularity is much finer. TreeBoost enhances stability by using small

trees, and by the effect of averaging over many of them. The interaction level of TreeBoost approximations is effectively controlled by limiting the size of the individual constituent trees.

Among the purported biggest advantages of single tree models is interpretability, whereas boosted trees are thought to lack this feature. Small trees can be easily interpreted, but due to instability such interpretations should be treated with caution. The interpretability of larger trees is questionable (Ripley 1996). TreeBoost approximations can be interpreted using partial dependence plots in conjunction with the input variable relative importance measure, as illustrated in Sections 8.3 and 9. While not providing a complete description, they at least offer some insight into the nature of the input–output relationship. Although these tools can be used with any approximation method, the special characteristics of tree based models allow their rapid calculation. Partial dependence plots can also be used with single regression trees, but as noted above, more caution is required owing to greater instability.

After sorting on the input variables, the computation of the regression TreeBoost procedures (LS_, LAD_, and M_TreeBoost) scales linearly with the number of observations N, the number of input variables n, and the number of iterations M. It scales roughly as the logarithm of the size of the constituent trees J. In addition, the classification algorithm L_K_TreeBoost scales linearly with the number of classes K; but it scales highly *sub*–linearly with the number of iterations M, if influence trimming (Section 4.5.1) is employed. As a point of reference, applying M_TreeBoost to the garnet data of Section 9.1 ($N = 13317$, $n = 11$, $J = 6$, $M = 500$) required 20 seconds on a 933Mh Pentium III computer.

As seen in Section 5, many boosting iterations ($M \simeq 500$) can be required to obtain optimal TreeBoost approximations, based on small values of the shrinkage parameter ν (36). This is somewhat mitigated by the very small size of the trees induced at each iteration. However, as illustrated in Fig. 1, improvement tends to be very rapid initially and then levels off to slower increments. Thus, nearly optimal approximations can be achieved quite early ($M \simeq 100$) with correspondingly much less computation. These near–optimal approximations can be used for initial exploration and to provide an indication of whether the final approximation will be of sufficient accuracy to warrant continuation. If lack–of–fit improves very little in the first few iterations (say 100), it is unlikely that there will be dramatic improvement later on. If continuation is judged to be warranted, the procedure can be restarted where it left off previously, so that no computational investment is lost. Also, one can use larger values of the shrinkage parameter to speed initial improvement for this purpose. As seen in Fig. 1, using $\nu = 0.25$ provided accuracy within 10% of the optimal ($\nu = 0.1$) solution after only 20 iterations. In this case however, boosting would have to be restarted from the beginning if a smaller shrinkage parameter value were to be subsequently employed.

The ability of TreeBoost procedures to give a quick indication of potential predictability, coupled with their extreme robustness, makes them a useful preprocessing tool that can be applied to imperfect data. If sufficient predictability is indicated, further data cleaning can be invested to render it suitable for more sophisticated, less robust, modeling procedures.

If more data become available after modeling is complete, boosting can be continued on the new data starting from the previous solution. This usually improves accuracy provided an independent test data set is used to monitor improvement to prevent overfitting on the new data. Although the accuracy increase is generally less than would be obtained by redoing the entire analysis on the combined data, considerable computation is saved.

Boosting on successive subsets of data can also be used when there is insufficient random access main memory to store the entire data set. Boosting can be applied to "arcbites" of data (Breiman 1997) sequentially read into main memory, each time starting at the current solution, recycling over previous subsets as time permits. Again, it is crucial to use an independent test set to stop training on each individual subset at that point where the estimated accuracy of the combined approximation starts to diminish.

11 Acknowledgments

Helpful discussions with Trevor Hastie, Bogdan Popescu, and Robert Tibshirani are gratefully acknowledged. This work was partially supported by CSIRO Mathematical and Information Sciences, Australia, the Department of Energy under contract DE-AC03-76SF00515, and by grant DMS9764431 of the National Science Foundation.

References

[1] Becker, R. A. and Cleveland, W. S (1996). The design and control of Trellis display. *J. Comput. & Statist. Graphics* **5**, 123–155.

[2] Breiman, L. (1997). Pasting bites together for prediction in large data sets and on–line. Univ. of Calif., Berkeley, Dept. of Statistics technical report.

[3] Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Comp.*, **11**, 1493–1517.

[4] Breiman, L., Friedman, J. H., Olshen, R., and Stone, C. (1983). *Classification and Regression Trees.* Wadsworth.

[5] Copas, J. B. (1983). Regression, prediction, and shrinkage (with discussion). *J. R. Statist. Soc. B* **45**, 311–354.

[6] Donoho, D. L. (1993). Nonlinear wavelete methods for recovery of signals, densities, and spectra from indirect and noisy data. In *Different Perspectives on Wavelets, Proc. of Symp. in Applied mathematics,* I. Daubechies (ed.) v. 47, Amer. math. Soc., Providence R. I., 173–205.

[7] Drucker, H. (1997). Improving regressors using boosting techniques. Proceedings of *the Fourteenth International Conference on Machine Learning.* ed. D. Fisher, Jr., pp. 107–115. Morgan–Kaufmann.

[8] Duffy, N. and Helmbold, D. (1999). A geometric approach to leveraging weak learners. *Computational Learning Theory*, Proceedings of *4th European Conference EuroCOLT99.* P. Fischer and H. U. Simon Eds. Springer, 18–33.

[9] Freund, Y and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 148–156.

[10] Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Annals of Statistics* **19**(1), 1–141.

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

[12] Griffin, W. L., Fisher, N. I., Friedman J. H., Ryan, C. G. and O'Reilly, S. (1997). Cr–Pyrope garnets in lithospheric mantle. *J. Petrology* (to appear).

[13] Hastie, T. and Tibshirani, R (1990). *Generalized Additive Models.* Chapman & Hall.

[14] Huber, P. (1964). Robust estimation of a location parameter. *Annals of Mathematical Statistics,* **35**, 73–101.

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

[16] Powell, M. J. D. (1987). Radial basis functions for multivariate interpolation: a review. In *Algorithms for Approximation,* eds. J. C. Mason and M. G. Cox, pp 143–167. Oxford: Clarendon Press.

[17] Ratsch, G., Onoda, T., and Muller, K. R. (1998). Soft margins for AdaBoost. NeuroCOLT Technical Report NC-TR-98-021.

[18] Ripley, B. D. (1996). *Pattern Recognition and Neural networks.* Cambridge University Press.

[19] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back–propagating errors. *Nature* **323**, 533–536.

[20] Schapire, R. and Singer, Y. (1998). Improved boosting algorithms using confidence–rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory.*

[21] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* Springer.

[22] Warner, J. R., Toronto, A. E., Veasey, L. R. and Stephenson, R. (1961). A mathematical model for medical diagnosis – application to congenital heart disease. *J. Amer. Med. Assoc.* **177**, 177–184.