XBART: Accelerated Bayesian Additive Regression Trees

Jingyu He
University of Chicago

Saar Yalov
Arizona State University

P. Richard Hahn
Arizona State University

Abstract

Bayesian additive regression trees (BART) (Chipman et al., 2010) is a powerful predictive model that often outperforms alternative models at out-of-sample prediction. BART is especially well-suited to settings with unstructured predictor variables and substantial sources of unmeasured variation as is typical in the social, behavioral and health sciences. This paper develops a modified version of BART that is amenable to fast posterior estimation. We present a stochastic hill climbing algorithm that matches the remarkable predictive accuracy of previous BART implementations, but is many times faster and less memory intensive. Simulation studies show that the new method is comparable in computation time and more accurate at function estimation than both random forests and gradient boosting.

1 INTRODUCTION

Tree-based regression methods — CART (Breiman et al., 1984), random forests (Breiman, 2001), and gradient boosting (Breiman, 1997; Friedman, 2001, 2002) — are highly successful and widely used for supervised learning. Bayesian additive regression trees — or BART — is a closely related but less well-known method that often achieves superior prediction/estimation accuracy. The "Bayesian CART" (single-tree) model was introduced in Chipman et al. (1998) and the BART model first appeared in Chipman et al. (2010), although software was publicly available as early as 2006. Contrary to common perception, BART is not merely a version of random forests or boosted regression trees in which prior distributions have been placed over model parameters. Instead, the Bayesian perspective leads to a fundamentally new tree growing criterion and algorithm, which yields a number of practical advantages — robustness to the choice of user-selected tuning parameters, more accurate predictions, and a natural Bayesian measure of uncertainty.

Despite these virtues, BART's wider adoption has been slowed by its more severe computational demands relative to alternatives, owing to its reliance on a random walk Metropolis-Hastings Markov chain Monte Carlo (MCMC) approach. The current fastest implementation, the `R` package `dbarts`, takes orders of magnitude longer than the widely-used `R` package `xgboost`, for example. This paper develops a variant of BART that is amenable to fast posterior estimation, making it almost as fast as `xgboost` (after cross-validating), while still retaining BART's hyperparameter robustness and remarkable predictive accuracy.

First, we describe the BART model to motivate our computational innovations. We derive the BART model's tree-growing criterion, which is notably different than the traditional sum-of-squares criterion used by other methods. We then describe the new algorithm accelerated Bayesian additive regression trees heuristic (XBART) and illustrate its impact on fast, accurate statistical prediction. Specifically, we compare the new method's performance to random forests, boosted regression trees, neural networks as well as the standard MCMC implementations of BART.

2 BART IN DETAIL

2.1 The Model: Likelihood and Prior

The BART model is an additive error mean regression model

$$y_i = f(\mathrm{x}_i) + \epsilon_i \tag{1}$$

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

where the ϵ_i are assumed to be independent mean zero Gaussians and $f(\cdot)$ is an unknown function. The BART prior represents the unknown function $f(\mathrm{x})$ as a sum of many piecewise constant binary regression trees:

$$f(\mathrm{x}) = \sum_{l=1}^{L} g_l(\mathrm{x}, T_l, \mu_l) \tag{2}$$

where T_l denotes a regression tree and μ_l denotes a vector of scalar means associated to the leafs nodes of T_l. Each tree T_l, $1 \leq l \leq L$, consists of a set of internal decision nodes which define a partition of the covariate space (say $\mathcal{A}_1, \ldots, \mathcal{A}_{B(l)}$), as well as a set of terminal nodes or leaves corresponding to each element of the partition. Further, each element of the partition $\mathcal{A}_b$ is associated a parameter value, μ_{lb}. Taken together the partition and the leaf parameters define a piecewise constant function: $g_l(x) = \mu_{lb}$ if $x \in \mathcal{A}_b$; see Figure 1.

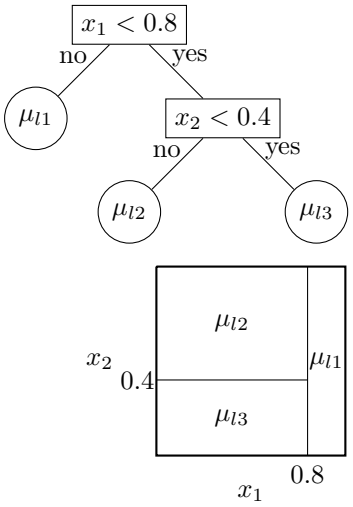

Figure 1: (Top) An example binary tree, with internal nodes labelled by their splitting rules and terminal nodes labelled with the corresponding parameters μ_{lb}. (Bottom) The corresponding partition of the sample space and the step function.

The tree prior $p(T_l)$ is specified by three components: (i) the probability of a node having children at depth d

$$\alpha(1+d)^{-\beta}, \qquad \alpha \in (0,1), \beta \in [0, \infty),$$

(ii) the uniform distribution over available predictors for splitting rule assignment at each interior node, and (iii) the uniform distribution on the discrete set of available splitting values for the assigned predictor at each interior node. This last choice has the appeal of invariance under monotone transformations of the predictors. Chipman et al. (2010) recommend $\alpha = .95$ and $\beta = 2$ to enforce small trees. Finally, the leaf mean parameters, μ_{lb} are assigned independent mean-zero normal priors: $\mu_{lb} \sim \mathrm{N}(0, \tau)$. The parameter τ is a crucial regularization parameter; pointwise prior variance of f is τL.

2.2 The BART Splitting criterion

By viewing the model as a data generating process, the Bayesian vantage point motivates modifications to the usual splitting criterion. Because the model stipulates that observations in the same leaf node share the same mean parameter, the prior predictive distribution — obtained by integrating out the unknown group specific mean — is simply a mean-zero multivariate normal distribution with covariance matrix

$$\mathbf{V} = \tau \mathrm{J}\mathrm{J}^t + \sigma^2 \mathbf{I},$$

where τ is the prior variance of the leaf-specific mean parameter, σ^2 is the variance of the additive error, and J is a column vector of all ones. Observe that the prior predictive density of $\mathrm{y} \sim \mathrm{N}(0, \mathbf{V})$ is

$$p(\mathrm{y} \mid \tau, \sigma^2) = (2\pi)^{-n/2} \det(\mathbf{V})^{-1/2} \exp\left(-\frac{1}{2}\mathrm{y}^t \mathbf{V}^{-1} \mathrm{y}\right),$$

which can be simplified by a direct application of the matrix inversion lemma to $\mathbf{V}^{-1}$:

$$\mathbf{V}^{-1} = \sigma^{-2}\mathbf{I} - \frac{\tau}{\sigma^2(\sigma^2 + \tau n)}\mathrm{J}\mathrm{J}^t.$$

Applying Sylvester's determinant theorem to $\det \mathbf{V}^{-1}$ and taking logarithms yields a marginal log-likelihood of

$$-\frac{n}{2}\log(2\pi) - n\log(\sigma) + \frac{1}{2}\log\left(\frac{\sigma^2}{\sigma^2 + \tau n}\right) - \frac{1}{2}\frac{\mathrm{y}^t\mathrm{y}}{\sigma^2} + \frac{1}{2}\frac{\tau}{\sigma^2(\sigma^2 + \tau n)}s^2,$$

where we write $s \equiv \mathrm{y}^t\mathrm{J} = \sum_i y_i$ so that $\mathrm{y}^t\mathrm{J}\mathrm{J}^t\mathrm{y} = (\sum_i y_i)^2 = s^2$. This likelihood is applied separately to partitions of the data corresponding to the leaves of a single fixed regression tree. Because observations in different leaf nodes are independent (conditional on σ^2), the full marginal log-likelihood is

given by

$$\sum_{b=1}^{B}\left\{-\frac{n_b}{2}\log(2\pi)-n_b\log(\sigma)+\frac{1}{2}\log\left(\frac{\sigma^2}{\sigma^2+\tau n_b}\right)\right.$$
$$\left.-\frac{1}{2}\frac{\mathrm{y}_b^t\mathrm{y}_b}{\sigma^2}+\frac{1}{2}\frac{\tau}{\sigma^2(\sigma^2+\tau n_b)}s_b^2\right\}$$
$$=-n\log(2\pi)-n\log(\sigma)-\frac{1}{2}\frac{\mathrm{y}^t\mathrm{y}}{\sigma^2}$$
$$+\frac{1}{2}\sum_{b=1}^{B}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n_b}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n_b)}s_b^2\right\}$$

where b runs over all the leaf nodes and $\sum_{b=1}^{B} n_b = n$. Notice that the first three terms are not functions of the partition (the tree parameter), so they are constant, leaving

$$\frac{1}{2}\sum_{b=1}^{B}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n_b}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n_b)}s_b^2\right\} \quad (3)$$

as the model-based split criterion, where (n_b, s_b, B) are functions of the data and the tree T.

2.3 The BART MCMC

The basic BART MCMC proceeds as a Metropolis-within-Gibbs algorithm, with the key update of the individual regression trees being conducted as a local random walk Metropolis-Hastings (MH) update, given all of the other trees as well as the residual variance parameter, σ^2. Let $\mathcal{T}$ denote the set of trees and $\mathcal{M}$ denote the set of leaf parameter vectors. Recall that $|\mathcal{T}| = |\mathcal{M}| = L$, and each $\mu_l \in \mathcal{M}$ is length $B(l)$.

The sequence of Gibbs updates are

1. $T_l, \mu_l \mid \mathcal{T}_{-l}, \mathcal{M}_{-l}, \sigma^2, \mathrm{y}$, for $l = 1, \ldots, L$, which is done compositionally (for each l) as
   (a) $T_l \mid \mathcal{T}_{-l}, \mathcal{M}_{-l}, \sigma^2, \mathrm{y}$,
   (b) $\mu_l \mid \mathcal{T}, \mathcal{M}_{-l}, \sigma^2, \mathrm{y}$,
2. $\sigma^2 \mid \mathcal{T}, \mathcal{M}, \mathrm{y}$.

Taking advantage of the additive structure of the model, these updates can be written as

1. $T_l, \mu_l \mid \mathrm{r}_l, \sigma^2$, for $l = 1, \ldots, L$, which is done compositionally (for each l) as
   (a) $T_l \mid \mathrm{r}_l, \sigma^2$,
   (b) $\mu_l \mid T_l, \mathrm{r}_l, \sigma^2$,
2. $\sigma^2 \mid \mathrm{r}$.

for "residuals" defined as

$$\mathrm{r}_l^{(k+1)} \equiv \mathrm{y} - \sum_{l'<l} g(\mathbf{X}; T_{l'}, \mu_{l'})^{(k+1)} - \sum_{l'>l} g(\mathbf{X}; T_{l'}, \mu_{l'})^{(k)},$$

and

$$\mathrm{r}^{(k)} \equiv \mathrm{y} - \sum_{l=1}^{L} g(\mathbf{X}; T_l, \mu_l)^{(k)},$$

where k indexes the Monte Carlo iteration. Update 1(a) is a Metropolis-Hastings update based on the integrated likelihood given in (3). Update 1(b) is a conditionally conjugate Gaussian mean update done separately for each leaf node parameter μ_{lb}, $b = 1 \ldots B(l)$. Update 2 is a conditionally conjugate inverse-Gamma update.

Step 1(a) is handled with a random walk as follows. Given a current tree, T, modifications are proposed and either accepted or rejected according to a likelihood ratio based on (3). Chipman et al. (1998) describes proposals comprising a birth/death pair, in which a birth spawns to children from a given bottom node and a death kills a pair of sibling children; see Pratola (2016) for alternative choices. For example, in a birth move, a variable to split on, as well as a cut-point to split at, are selected uniformly at random from the available splitting rules. Via these simple MH updates, BART stochastically searches through regression models of varying complexity (in terms of tree-depth). For "smaller" problems, with dozens of predictors and thousands of observations, this MCMC approach has proven to be remarkably effective; for larger problems, with hundreds of thousands of observations, it does not work well on standard desktops.

In the next section, we present our new stochastic hill climbing algorithm called accelerated Bayesian additive regression trees (XBART), see algorithm 2. It follows the Gibbs update framework but replace the Metropolis-Hastings updates of each single tree by a new grow-from-root backfitting strategy; see Algorithm 1.

3 XBART

3.1 Grow-from-root backfitting

Rather than making small moves to a given tree $T_l^{(k)}$ at iteration $k+1$, here we ignore the current tree and grow an entirely new tree $T_l^{(k+1)}$ from scratch. We grow each tree recursively and stochastically and the tree growing process is also terminated stochastically, based on the "residual" data defined above. The pseudo-code is presented in Algorithm 1.

Specifically, at each level of the recursion we consider every available cut-point (decision rule threshold) for each variable[1] and evaluate the integrated likelihood criterion, the exponential of expression (3). We also consider the no-split option, which corresponds to a cut-point outside of the range of the available data. How many such *null* cut-points to consider is a modeling decision; we default to one such null cut-point per variable. Accordingly, with C available active cut-points and V total variables we perform $C \times V + 1$ likelihood evaluations. Each of the active cut-points is weighted by $\alpha(1+d)^{-\beta}$ and the unweighted cut-points weighted by $1 - \alpha(1+d)^{-\beta}$, as per the prior[2]. Since data is pre-sorted, we index candidate cut-points by their rank, $c = 0, 1, \cdots, C \times V$ and $c = 0$ denotes a *null* cut-point, the "do not split" option. Selection of a variable to split on, and a cut-point to split at,are then chosen by Bayes rule:

$$\pi(v, c) = \frac{\exp\left(\ell(c, v)\right)\kappa(c)}{\sum_{v'=1}^{V}\sum_{c'=0}^{C}\exp\left(\ell(c', v')\right)\kappa(c')} \qquad (4)$$

where

$$\begin{aligned}\ell(v, c) = \frac{1}{2}\Bigg\{&\log\left(\frac{\sigma^2}{\sigma^2 + \tau n(\leq, v, c)}\right) \\ &+ \frac{\tau}{\sigma^2(\sigma^2 + \tau n(\leq, v, c))}s(\leq, v, c)^2\Bigg\} \\ + \frac{1}{2}\Bigg\{&\log\left(\frac{\sigma^2}{\sigma^2 + \tau n(>, v, c)}\right) \\ &+ \frac{\tau}{\sigma^2(\sigma^2 + \tau n(>, v, c))}s(>, v, c)^2\Bigg\}\end{aligned}$$

for $c \neq 0$. Here $n(\leq, v, c)$ is the number of observations in the current leaf node that have $x_v \leq c$ and $s(\leq, v, c)$ is the sum of the residual $\mathrm{r}_l^{(k)}$ of those same observations; $n(>, v, c)$ and $s(>, v, c)$ are defined analogously. Also, $\kappa(c \neq 0) = 1$.

For $c = 0$, corresponding to null cut-points or the stop-splitting option, we have instead

$$\ell(v, c) = \frac{1}{2}\left\{\log\left(\frac{\sigma^2}{\sigma^2 + \tau n}\right) + \frac{\tau}{\sigma^2(\sigma^2 + \tau n)}s^2\right\}$$

and $\kappa(0) = \frac{1-\alpha(1+d)^{-\beta}}{\alpha(1+d)^{-\beta}}$, where n denotes the number of observations in the current leaf node, $n = n(\leq, v, c) + n(>, v, c)$ and s denotes the sum over all the current leaf data.

[1]For simplicity, in this paper we consider only continuous predictor variables.

[2]Equivalently, the active cut-points are equally weighted and the no split option is weighted $V(\alpha^{-1}(1+d)^{\beta} - 1)$. An additional multiplier could be used here to encourage/discourage tree growth.

Using this new tree-growing strategy, we find that different default parameters are advisable. We recommend $L = \frac{1}{4}(\log n)^{\log\log n}$, $\alpha = 0.95$, $\beta = 1.25$ and $\tau = \frac{3}{10}\mathrm{var(y)}/L$. This choice of L is a function that is faster growing than $\log n$, but slower than $\sqrt{n}$, while the lower value of β permits deeper trees (than BART's default $\beta = 2$). Allowing L to grow as a function of the data permits smoother functions to be estimated more accurately as the sample size grows, whereas a sample size-independent choice would be limited in its smoothness by the number of trees. The suggested choice of τ dictates that *a priori* the function will account for 30% of the observed variance of the response variable. Finally, while BART must be run for many thousands of iterations with a substantial burn-in period, our default suggestion is just 40 sweeps through the data, discarding the first 15 as burn-in.

3.2 Pre-sorting Features for Efficiency

Observe that the BART criterion depends on the partition sums only. An important implication of this, for computation, is that with sorted predictor variables the various cut-point integrated likelihoods can be computed rapidly via a single sweep through the data (per variable), taking cumulative sums. Let $\mathbf{O}$ denote the V-by-n array such that o_{vh} denotes the index, in the data, of the observation with the hth smallest value of the vth predictor variable x_v. Then, taking the cumulative sums gives

$$s(\leq, v, c) = \sum_{h \leq c} \mathrm{r}_{o_{vh}} \qquad (5)$$

and

$$s(>, v, c) = \sum_{h=1}^{n} r_{lh} - s(\leq, v, c). \qquad (6)$$

The subscript l on the residual indicates that these evaluations pertain to the update of the lth tree.

The above formulation is useful if the data can be presorted and, furthermore, the sorting can be maintained at all levels of the recursive tree-growing process. To achieve this, we must "sift" each of the variables before passing to the next level of the recursion. Specifically, we form two new index matrices $\mathbf{O}^{\leq}$ and $\mathbf{O}^{>}$ that partition the data according to the selected split rule. For the selected split variable v and selected split c, this is automatic: $O_v^{\leq} = O_{v,1:c}$ and $O_v^{>} = O_{v,(c+1):n}$. For the other $V - 1$ variables, we sift them by looping through all n available observations, populating $O_q^{\leq}$ and $O_q^{>}$, for $q \neq v$, sequentially, with values o_{qj} according to whether $x_{vo_{qj}} \leq c$ or $x_{vo_{qj}} > c$, for $j = 1, \ldots, n$.

Algorithm 1 Grow-from-root backfitting

procedure GROW_FROM_ROOT(y, $\mathbf{X}$, C, m, w, σ^2) ▷ Fit a tree using data y and $\mathbf{X}$ by recursion.
output A tree T_l and a vector of split counts w_l.
  $N \leftarrow$ number of rows of y, x
  Sample m variables use weight w as shown in section 3.4.
  Select C cutpoints as shown in section 3.3.
  Evaluate $C \times m + 1$ candidate cutpoints and no-split option with equation (4).
  Sample one cutpoint propotional to equation (4).
  if sample no-split option **then**
    Sample leaf parameter from normal distribution $\mu \sim N\left(\sum y / \left[\sigma^2\left(\frac{1}{\tau} + \frac{N}{\sigma^2}\right)\right], 1 / \left[\frac{1}{\tau} + \frac{N}{\sigma^2}\right]\right)$. **return**
  else
    $w_l[j] = w_l[j] + 1$, add count of selected split variable.
    Split data to left and right node.
    GROW_FROM_ROOT(y_{left},$\mathbf{X}_{\text{left}}$, C, m, w, σ^2)
    GROW_FROM_ROOT(y_{right},$\mathbf{X}_{\text{right}}$, C, m, w, σ^2)

Because the data is processed in sorted order, the ordering will be preserved in each of the new matrices $\mathbf{O}^{\leq}$ and $\mathbf{O}^{>}$. This strategy was first presented in Mehta et al. (1996) in the context of tree classification algorithms.

3.3 Recursively Defined Cut-points

Evaluating the integrated likelihood criterion is straightforward, but the summation and normalization required to sample the cut-points contribute a substantial computational burden in its own right. Therefore, it is helpful to consider a restricted number of cut-points C. This can simply be achieved by taking every jth value (starting from the smallest) as an eligible split point with $j = \lfloor \frac{n_b - 2}{C} \rfloor$. As the tree grows deeper, the amount of data that is skipped over diminishes. Eventually we get $n_b < C$, and each data point defines a unique cut-point. In this way the data could, without regularization, be fit perfectly, even though the number of cut-points at any given level is given an upper limit. As a default, we set the number of cut-points to $\max(\sqrt{n}, 100)$, where n is the sample size of the entire data set.

Our cut-point subsampling strategy is more naive than the cut-point subselection search heuristics used by `XGBoost` (Chen and Guestrin, 2016) and `LightGBM` (Ke et al., 2017), which both consider the gradient evaluated at each cut-point when determining the next split. Our approach does not consider the response information at all, but rather defines a predictor-dependent prior on the response surface. That is, given a design matrix $\mathbf{X}$, a sample functions can be drawn from the prior distribution by sampling trees, splitting uniformly at random among the cut-points defined by the node-specific quantiles, in a sequential fashion. In further contrast, the proposed method stochastically samples cut-points proportional to its objective function, rather than deterministically maximizing the likelihood-prior. Then, multiple sweeps are made through the data. Rather than greedy (approximate) optimization, like `XGBoost` and `LightGBM`, the proposed algorithm performs a stochastic hill climb by coordinate ascent over multiple sweeps through the parameters.

3.4 Sparse Proposal Distribution

As a final modification, we strike an intermediate balance between the local BART updates, which randomly consider one variable at a time, and the all-variables Bayes rule described above. We do this by considering $m \leq V$ variables at a time when sampling each splitting rule. Rather than drawing these variables uniformly at random, as done in random forests, we introduce a parameter vector w which denotes the prior probability that a given variable is chosen to be split on, as suggested in Linero (2016). Before sampling each splitting rule, we randomly select m variables with probability proportional to w. These m variables are sampled sequentially and *without replacement*, with selection probability proportional to w.

The variable weight parameter w is given a Dirichlet prior with hyperparameter $\bar{\mathrm{w}}$ set to all ones and subsequently incremented to count the total number of splits across all trees. The split counts are then updated in between each tree sampling/growth step:

$$\bar{\mathrm{w}} \leftarrow \bar{\mathrm{w}} - \bar{\mathrm{w}}_l^{(k-1)} + \bar{\mathrm{w}}_l^{(k)} \tag{7}$$

where $\bar{\mathrm{w}}_l^{(k)}$ denotes the length-V vector recording the number of splits on each variable in tree l at iteration k. The weight parameter is then resampled as $\mathrm{w} \sim \text{Dirichlet}(\bar{\mathrm{w}})$. Splits that improve the likelihood function will be chosen more often than those that don't. The parameter w is then updated to

reflect that, making chosen variables more likely to be considered in subsequent sweeps. In practice, we find it is helpful to use all V variables during an initialization phase, to more rapidly obtain an accurate initial estimate of w.

3.5 The Estimator

Given K iterations of the algorithm, the final $K - I$ samples are used to compute a point-wise average function evaluation, where $I < K$ is denotes the length of the burn-in period. As mentioned above, we recommend $K = 40$ and $I = 15$ for routine use. The final estimator is therefore expressible as

$$\bar{f}(\mathbf{X}) = \frac{1}{K-I} \sum_{k>I}^{K} f^{(k)}(\mathbf{X}). \tag{8}$$

where $f^{(k)}$ denotes a sample of the forest, as in expression 2, drawn by algorithm 2. We note that this corresponds to the Bayes optimal estimator under mean squared error estimation loss, provided that we have samples from a legitimate posterior distribution. As the grow-from-root strategy is not a proper full conditional, this estimator must be considered a greedy stochastic approximation (but see section 3.6). Nonetheless, simulation results strongly suggest that the approximation is adequate.

A few remarks on posterior uncertainty. First, with only $K = 40$ sweeps, the XBART posterior uncertainty is likely understated. However, the standard BART MCMC is probably not mixing well in most contexts, either, and yet still provides useful, if approximate, uncertainty quantification. Second, experiments with a version of XBART based on only the final sweep, $K - I = 1$, performed worse than methods with $K - I > 1$, suggesting that our posterior exploration, while imperfect, is still beneficial.

3.6 Metropolis-Hastings Proposal Distribution

A fully Bayesian algorithm can be obtained by using the grow-from-root fitting algorithm as a data-driven Metropolis-Hastings proposal distribution. Importantly, the MH accept-reject step should be completed at the end of each *sweep*, that is, after proposing an entirely new set of trees and their associated parameters. Denote the current and proposed sets, repectively, by $F = \{\mathcal{T}, \mathcal{M}\}$ and $F' = \{\mathcal{T}', \mathcal{M}'\}$, where $\mathcal{T} = \{T_1, T_2, \ldots, T_L\}$ and $\mathcal{M} = \{\mu_1, \mu_2, \ldots, \mu_L\}$ denote the set of trees and leaf parameters, respectively. The grow-from-root algorithm generates a proposal of moving from F to F' with density $q(F', F)$ defined by a recursive product of terms as in 3.1. The probability of growing any particular tree is characterized by the probability of a certain sequence of split (or no-split) decisions encountered as one navigates down a given tree. The density of the leaf parameters, conditional on a given tree structure, follows from the corresponding conjugate normal update. See Algorithm 3. To show that this MH procedure is valid, we need only show that any set of trees and parameters can be reached from any other set (positive recurrence) and that the proposal density is well-defined upon interchanging the sets of tree/parameter pairs; the construction of the usual Metropolis-Hastings ratio ensures detailed balance. Observe that one initializes the proposal process starting from a residual vector defined by F. To propose the first tree in F', we "kill" the first tree from F and grow an entirely new tree. In the second step, we recompute the residual and repeat, and so forth. After L steps, L new trees have been regrown in an unrestricted fashion. Although the trees grown in this sequence are not independent, their joint density is given by a product of conditional densities, all of the dependence being passed through the redefinition of the residual at each step; see Algorithm 4. Consequently, one can interchange the roles of F and F' in this elaborate proposal mechanism simply by beginning the process with the residual defined by F' rather than F. Further work will consider the efficacy of this approach.

4 SIMULATION STUDIES

4.1 Data Generating Process

To demonstrate the performance of the new accelerated BART heuristic, which we call XBART, we estimate function evaluations with a hold-out set that is a quarter of the training sample size and judge accuracy according to root mean squared error (RMSE). We consider four different challenging functions, f, as defined in Table 1. In all cases, $x_j \overset{\text{iid}}{\sim} \text{N}(0, 1)$ for $j = 1, \ldots, d = 30$. The data is generated according to the additive error mode (1), with $\epsilon_i \overset{\text{iid}}{\sim} \text{N}(0, 1)$. We consider $\sigma = \kappa \text{Var}(f)$ for $\kappa \in \{1, 10\}$.

4.2 Methods

We compare to leading machine learning algorithms: random forests, gradient boosting machines, neural networks, and BART MCMC. All implementations had an `R` interface and were the current fastest implementations to our knowledge: `ranger` (Wright and Ziegler, 2015), `xgboost` (Chen and Guestrin,

Algorithm 2 Accelerated Bayesian Additive Regression Trees (XBART)

procedure XBART(y, $\mathbf{X}, C, m, L, I, K, \alpha, \eta$) ▷ ($\alpha, \eta$ are prior parameter of σ^2)
output Samples of forest
  $V \leftarrow$ number of columns of $\mathbf{X}$
  $N \leftarrow$ number of rows of $\mathbf{X}$
  Initialize $\mathrm{r}_l^{(0)} \leftarrow \mathrm{y}/L$.
  for k in 1 to K **do**
    for l in 1 to L **do**
      Calculate residual $\mathrm{r}_l^{(k)}$ as shown in section 2.3.
      if $k < I$ **then**
        GROW_FROM_ROOT($\mathrm{r}_l^{(k)}$,$\mathbf{X}$, C, V, w, σ^2) ▷ use all variables in burnin iterations
      else
        GROW_FROM_ROOT($\mathrm{r}_l^{(k)}$,$\mathbf{X}$, C, m, w, σ^2)
      $\bar{\mathrm{w}} \leftarrow \bar{\mathrm{w}} - \bar{\mathrm{w}}_l^{(k-1)} + \bar{\mathrm{w}}_l^{k}$ ▷ update $\bar{\mathrm{w}}$ with split counts of current tree
      $\mathrm{w} \sim$ Dirichlet($\bar{\mathrm{w}}$)
      $\sigma^2 \sim$ Inverse-Gamma($N + \alpha, \mathrm{r}_l^{(k)t}\mathrm{r}_l^{(k)} + \eta$)
  return

2016), and `Keras` (Chollet et al., 2015), `dbarts` respectively. For `Keras` we used a single strong architecture but varied epochs depending on the noise in the problem. For `xgboost` we consider two specifications, one using the software defaults and another determined by by 5-fold cross-validated grid optimization (see Table 2); a reduced grid of parameter values was used at sample sizes $n > 10,000$. Comparison with `ranger` and `dbarts` are shown in supplementary material.

Algorithm 3 Grow Probability

procedure GrowProb(r, T, μ, X, h)
  $\psi_h \leftarrow \pi(v_h(T), c_h(T)$ ▷ From equation (3)
  if $v_h(T) =$ NULL **then** ▷ If this is bottom node
    $\psi_h \leftarrow \psi_h \times \phi(\mu_h \mid \mu, \sigma^2)$
  else
    $\psi_h \leftarrow$GrowProb($r_{\text{left}}, T, \mu, 2h$)
    $\psi_h \leftarrow$GrowProb($r_{\text{right}}, T, \mu, 2h+1$)
  return ψ_h

Algorithm 4 Evaluate Proposal Density

procedure PropDens($F, F', y, \sigma^2, \tau, x$)
  Construct residual $r \leftarrow y - f(F_{2:L})$, initialize $q \leftarrow 1$
  for l in 1 to L **do**
    Set $\psi \leftarrow$Prod(GROWPROB($r, F'_l, \mu_l, x, h = 1$))
    $q \leftarrow q \times \psi$
    Update residual $r \leftarrow y - f(F_{(l+1):L}) - f(F'_{1:l})$
  return $q = q(F', F)$

4.3 Computation

The software used was `R` version 3.4.4 with `xgboost` 0.71.2, `dbarts` version 0.9.1, `ranger` 0.10.1 and `keras` 2.2.0. The default hyperparameters for XGBoost are `eta` = 0.3, `colsample_bytree` = 1, `min_child_weight` = 1 and `max_depth` = 6. Ranger was fit with `num.trees` = 500 and `mtry` = $5 \approx \sqrt{d}$. BART, with the package `dbarts`, was fit with the defaults of `ntrees` = 200, `alpha` = 0.95, `beta` = 2, with a burn-in of 5,000 samples (`nskip` = 5000) and 2,000 retrained posterior samples (`ndpost` = 2000).

Table 1: Four true f functions

Name	Function
Linear	$\mathrm{x}^t\gamma;\ \gamma_j = -2 + \frac{4(j-1)}{d-1}$
Single index	$10\sqrt{a} + \sin(5a)$; $a = \sum_{j=1}^{10}(x_j - \gamma_j)^2$; $\gamma_j = -1.5 + \frac{j-1}{3}$.
Trig + poly	$5\sin(3x_1) + 2x_2^2 + 3x_3x_4$
Max	$\max(x_1, x_2, x_3)$

The default `dbarts` algorithm uses an evenly spaced grid of 100 cut-point candidates along the observed range of each variable (`numcuts` = 100, `usequants = FALSE`). For `Keras` we build a network with two hidden layers (15 nodes each) using ReLU activation function, ℓ_1 regularization at 0.01, and with 50/20 epochs depending on the signal to noise ratio.

Table 2: Hyperparameter Grid for XGBoost

Parameter name	$N = 10$K	$N > 10$K
`eta`	$\{0.1, 0.3\}$	$\{0.1, 0.3\}$
`max_depth`	$\{4, 8, 12\}$	$\{4, 12\}$
`colsample_bytree`	$\{0.7, 1\}$	$\{0.7, 1\}$
`min_child_weight`	$\{1, 10, 15\}$	10
`subsample`	0.8	0.8
`gamma`	0.1	0.1

4.4 Results

The performance of the new XBART algorithm was excellent, showing superior speed and performance relative to all the considered alternatives on essentially every data generating processes. The full results, averaged across five Monte Carlo replications, are reported in Tables 3. Neural networks perform

as well as XBART in the low noise settings under the Max and Linear functions. Unsurprisingly, neural networks outperform XBART under the linear function with low noise. Across all data generating processes and sample sizes, XBART was 31% more accurate than the cross-validated XGBoost method and typically faster. Specifically, the supplement examines the empirical examples given in Chipman et al. (2010).

The XBART method was slower than the untuned default XGBoost method, but was 3.5 times more accurate. This pattern points to one of the main benefits of the proposed method, which is that it has excellent performance using the same hyperparameter settings across all data generating processes. Importantly, these default hyperparameter settings were decided on the basis of prior elicitation experiments using different true functions than were used in the reported simulations. While XGBoost is quite fast, the tuning processes is left to the user and can increase the total computational burden by orders of magnitude.

Random forests and traditional MCMC BART were prohibitively slow at larger sample sizes. However, at $n = 10,000$ several notable patterns did emerge; see the supplementary material for full details. First was that BART and XBART typically gave very similar results, as would be expected. BART performed slightly better in the low noise setting and quite a bit worse in the high noise setting (likely due to inadequate burn-in period). Similarly, random forests do well in higher noise settings, while XGBoost and neural networks perform better in lower noise settings.

$\kappa = 1$

n	XBART	XGB+CV	XGB	NN
		Linear		
10k	1.74 (20)	2.63 (64)	3.23 (0)	1.39 (26)
50k	1.04 (180)	1.99 (142)	2.56 (4)	0.66 (28)
250k	0.67 (1774)	1.50 (1399)	2.00 (55)	0.28 (40)
		Max		
10k	0.39 (16)	0.42 (62)	0.79 (0)	0.40 (30)
50k	0.25 (134)	0.29 (140)	0.58 (4)	0.20 (32)
250k	0.14 (1188)	0.21 (1554)	0.41 (60)	0.16 (44)
		Single Index		
10k	2.27 (17)	2.65 (61)	3.65 (0)	2.76 (28)
50k	1.54 (153)	1.61 (141)	2.81 (4)	1.93 (31)
250k	1.14 (1484)	1.18 (1424)	2.16 (55)	1.67 (41)
		Trig + Poly		
10k	1.31 (17)	2.08 (61)	2.70 (0)	3.96 (26)
50k	0.74 (147)	1.29 (141)	1.67 (4)	3.33 (29)
250k	0.45 (1324)	0.82 (1474)	1.11 (59)	2.56 (41)

$\kappa = 10$

n	XBART	XGB+CV	XGB	NN
		Linear		
10k	5.07 (16)	8.04 (61)	21.25 (0)	7.39 (12)
50k	3.16 (135)	5.47 (140)	16.17 (4)	3.62 (14)
250k	2.03 (1228)	3.15 (1473)	11.49 (54)	1.89 (19)
		Max		
10k	1.94 (16)	2.76 (60)	7.18 (0)	2.98 (15)
50k	1.22 (133)	1.85 (139)	5.49 (4)	1.63 (16)
250k	0.75 (1196)	1.05 (1485)	3.85 (54)	0.85 (22)
		Single Index		
10k	7.13 (16)	10.61 (61)	28.68 (0)	9.43 (14)
50k	4.51 (133)	6.91 (139)	21.18 (4)	6.42 (16)
250k	3.06 (1214)	4.10 (1547)	14.82 (54)	4.72 (21)
		Trig + Poly		
10k	4.94 (16)	7.16 (61)	17.97 (0)	8.20 (13)
50k	3.01 (132)	4.92 (139)	13.30 (4)	5.53 (14)
250k	1.87 (1216)	3.17 (1462)	9.37 (49)	4.13 (20)

Table 3: Root mean squared error (RMSE) of each method. Column XGB+CV is result of XGBoost with tuning parameter by cross validation. The number in parenthesis is running time in seconds. First column is number of data observations (in thousands).

5 DISCUSSION

The grow-from-root strategy proposed here opens the door for computational innovations to be married to the novel BART stochastic fitting algorithm. Further, the proposed adaptive cut-points and variable selection proposal together define a novel predictor-dependent prior, marking a distinct Bayesian model. The simulation studies clearly demonstrate the beneficial synergy realized by the proposed approach: XBART is a state-of-the-art nonlinear regression method with computational demands that are competitive with the current fastest alternatives. In particular, the excellent performance without the need to cross-validate recommends XBART as a suitable default method for function estimation and prediction tasks when little is known about the response surface.

The source of XBART's superior performance is not entirely clear, but preliminary investigations point to two important factors. One, the BART splitting criterion involves (the current estimate of) the error standard deviation, σ, meaning that it is adaptively regularizing within the model fitting process. Two, we conjecture that the stochastic nature of the algorithm leads to better exploration of the parameter space than iterative optimizers. With fast model fitting software now in hand, this issue can be investigated more systematically in future work. Another line of future research is to incorporate XBART within extended BART models such as Bayesian causal forests (Hahn et al., 2017) and BART for log-linear models (Murray, 2017).

References

Breiman, L. (1997). Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at Berkeley.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. J. (1984). *Classification and regression trees*. Chapman and Hall/CRC.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM.

Chipman, H. A., George, E. I., and McCulloch, R. E. (1998). Bayesian CART model search. *Journal of the American Statistical Association*, 93(443):935–948.

Chipman, H. A., George, E. I., McCulloch, R. E., et al. (2010). BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298.

Chollet, F. et al. (2015). Keras.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.

Hahn, P. R., Murray, J. S., and Carvalho, C. (2017). Bayesian regression tree models for causal inference: regularization, confounding, and heterogeneous effects. *arXiv preprint arXiv:1706.09523*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154.

Linero, A. R. (2016). Bayesian regression trees for high dimensional prediction and variable selection. *Journal of the American Statistical Association*, (just-accepted).

Mehta, M., Agrawal, R., and Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In *International Conference on Extending Database Technology*, pages 18–32. Springer.

Murray, J. S. (2017). Log-linear bayesian additive regression trees for categorical and count responses. *arXiv preprint arXiv:1701.01503*.

Pratola, M. (2016). Efficent Metropolis-Hastings proposal mechanism for Bayesian regression tree models. *Bayesian Analysis*, 11(3):885–911.

Wright, M. N. and Ziegler, A. (2015). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *arXiv preprint arXiv:1508.04409*.