XBART Accelerated Bayesian Additive Regression Trees

FASTER and MORE ACCURATE than XGBoost

R & python code now available. Coming soon on CRAN and pip.

BART boasts state-of-the-art prediction accuracy.
But, BART MCMC can be SLOW.

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

XBART grows trees stochastically but recursively, using the unique BART split criteria, so it is FAST.

By growing trees recursively, many efficiency tricks can be exploited: pre-sorting variables, adaptive nested cutpoints, sparse trees.
Scan the QR code to see the paper for details.

by Jingyu He, Saar Yalov and P. Richard Hahn

XBART: Accelerated Bayesian Additive Regression Trees

Jingyu He [1] Saar Yalov [2] P. Richard Hahn [2]

[1]University of Chicago, jingyuhe@chicagobooth.edu [2]Arizona State University

Highlights

XBART is motivated by Bayesian additive regression trees (BART), provides fast posterior estimation for BART model. Simulation shows that

1. XBART is faster and more accurate than `xgboost` with tuning parameters by cross validation.
2. Fit large data set (250K observations) in tolerable time, which BART can never do.

BART Prior

Bayesian Additive Regression Trees, first appeared in Chipman et al. (2010). BART is not merely a version of random forest or boosted regression trees in which prior distributions have been placed over model parameters, but **prior over tree structure** and **parameters**.

Pros Robust to tuning parameter, **more accurate prediction**, a natural Bayesian measure of uncertainty.

Cons The random walk Metropolis-Hastings Markov chain Monte Carlo algorithm is slow.

The BART model is

$$y = \sum_{l=1}^{L} g_l(x, T_l, \mu_l) + \epsilon \qquad (1)$$

where T_l denotes regression tree and μ_l is vector of means associated to all nodes of tree l. The BART prior has three components

1. Probability of a node having children at depth d is
$$\alpha(1+d)^{-\beta}$$
2. Uniform distribution over available predictors to split at.
3. Uniform distribution on a discrete set of available splitting values for the assigned predictor.

The basic BART MCMC takes a Metropolis-within-Gibbs algorithm, update each tree by local random walk Metropolis-Hastings (MH) update. **Slow, cannot work on large data set.**

Bayesian Backfitting

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

Code available upon request for now, will be on CRAN and pip (python version) soon.

Grow-from-root Backfitting

Given the current node, the likelihood of each cutpoint candidate is

$$\pi(v, c) = \frac{\exp(\ell(c, v))\kappa(c)}{\sum_{v'=1}^{V}\sum_{c'=0}^{C} \exp(\ell(c', v'))\kappa(c')} \qquad (2)$$

where

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

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

For $c = 0$, corresponding to **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 $n = n(\le, v, c) + n(>, v, c)$, $s = s(\le, v, c) + s(<, v, c)$.

Pre-sorting Features for Efficiency

Observe that the BART criterion **depends on the partition sum only**. With sorted predictor variables, the likelihood of cut-point can be computed via a single sweep through the data (per variable), taking cumulative sum.

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

and

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

Recursive cut-points

Take every jth value (starting from the smallest) as an eligible split point with $j = \lfloor \frac{n_b - 2}{C} \rfloor$.

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.

Sparse Trees

We considering $m \le V$ variables at a time when sampling each splitting rule.

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.

Variable Importance Weights

The variable weight parameter w is given a Dirichlet prior with hyperparameter $\bar{\mathrm{w}}$ set to all ones.

Split counts are updated in between each tree sampling/growth step:

$$\bar{\mathrm{w}} \leftarrow \bar{\mathrm{w}} - \bar{\mathrm{w}}_l^{(k-1)} + \bar{\mathrm{w}}_l^{(k)} \qquad (3)$$

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 $\mathrm{w} \sim$ Dirichlet($\bar{\mathrm{w}}$).

Posterior Prediction

Given K iterations of the algorithm, suppose $I < K$ is denotes the length of the burn-in period, the final prediction is

$$\bar{f}(\mathbf{X}) = \frac{1}{K - I} \sum_{k>I}^{K} f^{(k)}(\mathbf{X}). \qquad (4)$$

where $f^{(k)}$ denotes a sample of the forest.

XBART algorithm

Algorithm 1 Grow-from-root backfitting

```
$N \leftarrow$ number of rows of $y, x$
Sample $m$ variables use weight $w$ as shown in section sparse trees.
Select $C$ cutpoints as shown in section Grow-from-root backfitting.
Evaluate $C \times m + 1$ candidate cutpoints and no-split option with equation (2).
Sample one cutpoint propotional to equation (2).
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($y_{\text{left}}, \mathbf{X}_{\text{left}}, C, m, \mathrm{w}, \sigma^2$)
    GROW_FROM_ROOT($y_{\text{right}}, \mathbf{X}_{\text{right}}, C, m, \mathrm{w}, \sigma^2$)
end if
```

Algorithm 2 Accelerated Bayesian Additive Regression Trees (XBART)

```
$V \leftarrow$ 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 Bayesian Backfitting.
        if $k < I$ then
            GROW_FROM_ROOT($\mathrm{r}_l^{(k)}, \mathbf{X}, C, V, \mathrm{w}, \sigma^2$) {use all variables in burnin iterations}
        else
            GROW_FROM_ROOT($\mathrm{r}_l^{(k)}, \mathbf{X}, C, m, \mathrm{w}, \sigma^2$)
        end if
        $\bar{\mathrm{w}} \leftarrow \bar{\mathrm{w}} - \bar{\mathrm{w}}_l^{(k-1)} + \bar{\mathrm{w}}_l^{k}$ {update $\bar{\mathrm{w}}$ with split counts of current tree}
        $\mathrm{w} \sim$ Dirichlet($\bar{\mathrm{w}}$)
        $\sigma^2 \sim$ Inverse-Gamma($N + \alpha, \mathrm{r}_l^{(k)t}\mathrm{r}_l^{(k)} + \eta$)
    end for
end for
return
```

Is it a valid MCMC algorithm?

The algorithm works well on its own right. We can use it **as proposal of M-H algorithm**, rather than a random walk M-H, to get full Bayesian inference. Future work.

Simulations

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 1. Root mean squared error (running time).