XBART: 一个更快更准的提升树算法

何靖宇
芝加哥大学布斯商学院

2019 年 5 月 2 日

1 前言

决策树 (decision tree) 和树的集成 (ensemble of trees) 可以说是机器学习领域最出名的方法之一，它们出现在每一本机器学习的入门课本里。决策树方法计算速度快、易于解释并且在很多应用中取得了成功。梯度提升树 (gradient boosting) Friedman [2001], Chen and Guestrin [2016] 则是当前最热门的算法之一，例如 Kaggle 数据挖掘比赛的大部分胜利者使用 xgboost 训练模型。不过，另一个最新的算法 BART (Bayesian Additive Regression Tree) Chipman et al. [2010] 在很多模拟和真实数据测试中取得了更高的预测准确率。BART 有着广泛的应用和变体，比如用于 log-linear regression Murray [2017]; 在 causal inference 的应用 Hahn et al. [2017]; 做变量选择 Linero [2018]；生存分析 Sparapani et al. [2016]；异方差 Pratola et al. [2017] 等等。作为一个标准的贝叶斯模型，BART 采取了 random walk Metropolis-Hastings 从后验分布抽样，然而导致了训练缓慢，不适用较大数据集等缺点。

本文简要介绍 XBART (Accelerated Bayesian Additive Regression Trees) 算法，受到 BART 启发，但是有着和 xgboost 一样的训练速度和甚至更好的表现。具体技术细节请参见 He et al. [2019]。XBART 的软件包 (R, python) 和例子请于http://xbart.io下载。

2 CART

说到树算法不得不从经典的 CART (Classification and Regression Trees) 说起。CART 模型由 Breiman 等人于 1984 年提出，它采用了一个递归并且贪婪的算法训练决策树，即在每个节点，遍历所有可能的切分点，计算如果在当前点切分的目标函数值，并且在最小化目标函数的点切分。对左右两个子节点递归调用步骤直到满足停止条件。CART 采用贪婪算法选取切分点，没有对树的复杂度惩罚，所以很容易过拟合数据。然而 CART 采用的递归方法能高效地完成训练任务。

3 BART

XBART 受到 BART 启发，采取 BART 的后验分布作为划分点判断准则，但是采用类似 CART 递归式的训练方法。可以跳过本段直接看 XBART 章节。

3.1 BART 先验分布

BART 受到了提升树 (boosting trees) 算法的启发，假设各个树之间可加性。可以把 BART 理解成一个在 boosting tree 空间上的正则先验分布 (regularization prior)。BART 模型假设

$$y_i = f(\mathrm{x}_i) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2) \tag{1}$$

$$f(\mathrm{x}) = \sum_{h=1}^{L} g(\mathrm{x}, T_h, M_h) \tag{2}$$

函数 $g(\mathrm{x}, T_h, M_h)$ 表示第 h 棵树对数据 x 的预测值，T 代表树的结构，M 代表树每个子节点的预测值 (leaf parameter)。BART 的先验分布可以写成

$$\pi(\theta) = \pi\left((T_1, M_1), \cdots, (T_L, M_L), \sigma\right) = \left[\prod_{j}^{L} \pi(M_j \mid T_j)\pi(T_j)\right] \pi(\sigma) \tag{3}$$

这个分解可以理解成 BART 假设每棵树之间独立，先假设树结构 T 的分布，再假设 leaf parameter M 对 T 的条件分布，最后是 σ^2 的分布，下面我们一一解释。

3.1.1 T 的先验分布

BART 对 T 的先验分布是生成性的 (generative)，即可以理解为一个从根结点构建树的概率过程。BART 假设，对于深度 d 的节点，它分成两个子节点的概率是 $\alpha(1+d)^{-\beta}$，停止分裂的概率则是 $1-\alpha(1+d)^{-\beta}$。

从根节点开始，$d=1$ 计算分裂的概率并依照此概率抽样，假设抽到了分裂，那么在所有可能的划分变量和划分值中均匀地随机取一个作为这个节点的决策。再依次看生成的左右子节点，$d=2$ 计算分裂的概率并抽样，如果分裂则继续均匀随机地抽一个划分变量和划分值，如果不分裂则停止生长。

BART 建议取 $\alpha = 0.95$ 和 $\beta = 2$，这样当 $d=5$ 的时候分裂概率几乎为零。所以这组参数隐含的假设是所有树都达不到深度 5，每个树的结构不复杂，所以可以尽量避免过拟合。

3.1.2 $M \mid T$ 的先验分布

当树的结构确定下来后，每个数据根据树的结构必然落入某一个子节点中。BART 假设每个子节点的 leaf parameter 有共轭的正态先验分布 $N(0, \tau)$。

3.1.3 σ^2 的先验分布

当森林里每一棵树都确定，我们可以计算出 $y - \sum_{h=1}^{L} g(\mathrm{x}, T_h, M_h)$ 即 BART 拟合的余项。假设余项有共轭的 inverse-gamma 先验分布。

3.2 BART 的 MCMC 算法

训练 BART 采用 Markov-chain Monte Carlo 算法，对 T, M 和 σ^2 的更新步骤可以写成如下的 Gibbs sampler。对于每一轮 MCMC 迭代

1. 更新 $T_l, \mu_l \mid \mathrm{r}_l, \sigma^2$, for $l = 1, \ldots, L$, 对每个 l 可以细分为两部：
   (a) 更新 $T_l \mid \mathrm{r}_l^{(k+1)}, \sigma^2$,
   (b) 更新 $\mu_l \mid T_l, \mathrm{r}_l^{(k+1)}, \sigma^2$,
2. 更新 $\sigma^2 \mid \mathrm{r}^{(k+1)}$.

其中 $\mathrm{r}_l^{(k+1)}$ 是第 $k+1$ 轮 MCMC 迭代，更新第 l 棵树时，对于除 l 之外其他所有树的余项

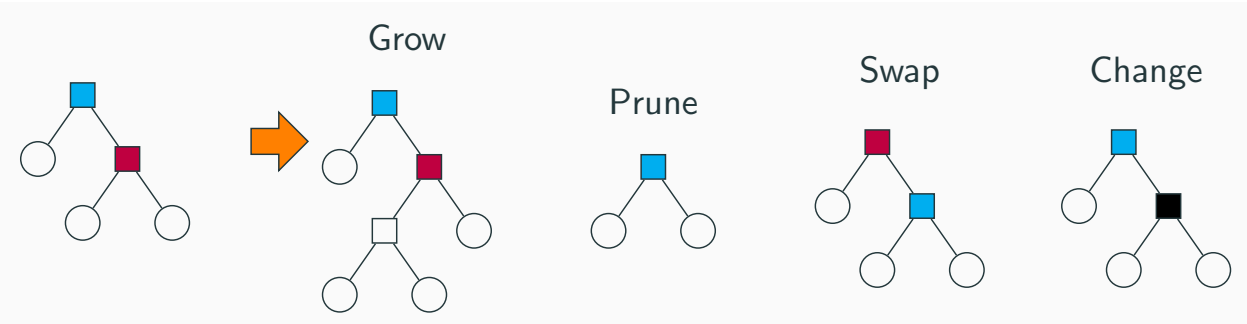
$$\mathrm{r}_l^{(k+1)} \equiv \mathrm{y} - \sum_{l'<l} g(\mathbf{X}; T_{l'}, \mu_{l'})^{(k+1)} - \sum_{l'>l} g(\mathbf{X}; T_{l'}, \mu_{l'})^{(k)},$$

$\mathrm{r}^{(k+1)}$ 是第 $k+1$ 轮迭代，所有树更新完后的余项

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

那么 $T_l \mid \mathrm{r}_l^{(k+1)}, \sigma^2$ 具体是怎么操作的呢？BART 采用了 random-walk Metropolis-Hastings 算法，每一次 proposal 采取随机游动有相同概率采取以下四种步骤的一种

- Grow: 随机均匀地选一个子节点分裂，并且随机均匀地选一个变量和划分点作为判断准则。
- Prune：随机均匀地选取一对子节点，删除。
- Swap：随机均匀地选两个中间节点，交换他们的判断准则。



- Change：随机均匀地选一个中间节点，更新判断准则，仍然是随机均匀地选一个变量和划分点作为新的判断准则。

看到这里你可能会问，所有判断准则都是随机均匀抽取的，效果怎么可能好？请注意这里只是 Metropolis-Hastings 算法的一个 proposal，接下来需要计算相应的 Metropolis-Hastings ratio 决定是否接受这个 proposal。如果 proposal 很离谱，大概率会被拒绝。所以不难想象，BART 的 proposal 接受率非常低，大量的计算资源浪费在 proposal 上。

当树的结构确定，我们可以得知每个数据点落在哪个叶节点内，并且计算每个叶节点数据的均值 $\bar{y}$，所以 $\mu \sim N(\bar{y}, \sigma^2) \times N(0, \tau)$，仍然是一个共轭的正态分布。

当所有树更新完毕，计算最终的余项，根据共轭的 inverse gamma 分布更新 σ^2，和经典的 Bayesian linear regression 一样。

4 XBART

有如下两点结论

- CART 采取了递归算法，训练速度快。但是因为划分点的选取法则不好所以容易过拟合。
- BART 预测效果好 (好划分点的选取法则和中止条件)，但是 random walk Metropolis-Hastings 训练方法太低效。

有没有可能采用 CART 的递归算法，但是用 BART 的后验分布作为划分点选取法则？这就是 XBART 的最基本想法。

4.1 XBART 算法

XBART 的算法可以拆解成两部：如何更新单个树以及如何更新整个森林。

4.1.1 单棵树算法

经过计算，BART 先验分布对应的**单个叶节点的**对数似然函数 (log-likelihood)，省略常数项是

$$\frac{1}{2}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n)}s^2\right\} \tag{4}$$

其中 n 是叶节点中数据点个数，$s=\sum_i y_i$ 是叶节点中数据的输出变量之和。每一个潜在的划分点自然地把数据分成左右两份，那么潜在划分点对应的对数后验分布 (log-posterior) 是

$$\frac{1}{2}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n_L}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n_L)}s_L^2\right\}+\frac{1}{2}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n_R}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n_R)}s_R^2\right\}+\log\left(\alpha(1+d)^{-\beta}\right) \tag{5}$$

其中 n_L, n_R 是左右两边数据点个数 $n_L+n_R=n$，s_L 和 s_R 是对应的输出变量之和。$\alpha(1+d)^{-\beta}$ 是这个节点可以划分的先验概率。假设一共有 V 个变量，每个变量有 C 个潜在的划分点，那么我们一共计算 $C\times V$ 个潜在划分点 log-posterior.

我们还需要考虑中止分裂的概率，注意中止分裂等价于当前节点所有数据在一个叶节点内，所以中止的概率是

$$\frac{1}{2}\left\{\log\left(\frac{\sigma^2}{\sigma^2+\tau n}\right)+\frac{\tau}{\sigma^2(\sigma^2+\tau n)}s^2\right\}+C\times V\times\log\left(1-\alpha(1+d)^{-\beta}\right) \tag{6}$$

注意 $\left(1-\alpha(1+d)^{-\beta}\right)$ 是这个节点不可以继续分裂的先验概率。

当所有潜在划分点以及中止分裂一共 $C\times V+1$ 个概率计算完毕，我们依照这组概率抽一个作为当前节点的判断准则（或者停止分裂）。那么单棵树生成算法是

```
grow_from_root(当前节点数据){
    计算潜在划分点和中止分裂的概率，并抽样
    if(中止分裂){
        return
    }else{
        把数据依照抽到的判断准则分成左右两部分
        grow_from_root(右边数据)
        grow_from_root(左边数据)
    }
}
```

注意我们采取了递归的方法生成单棵树。

4.1.2 森林算法

第 $(k+1)$ 轮 MCMC 迭代

1. 更新 $T_l, \mu_l \mid \mathrm{r}_l, \sigma^2$, for $l = 1, \ldots, L$, 对每个 l 可以细分为两部：
   (a) 更新 $T_l \mid \mathrm{r}_l^{(k+1)}, \sigma^2$, 用单棵树算法拟合除当前树外其他所有树的余项
   (b) 更新 $\mu_l \mid T_l, \mathrm{r}_l^{(k+1)}, \sigma^2$, 确定树结构后也确定数据点所在叶节点，根据共轭的正态后验分布抽取。
2. 更新 $\sigma^2 \mid \mathrm{r}^{(k+1)}$. 根据共轭的 inverse-gamma 后验分布抽取。

其中 $\mathrm{r}_l^{(k+1)}$ 是第 $k+1$ 轮 MCMC 迭代，更新第 l 棵树时，对于除 l 之外其他所有树的余项

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

$\mathrm{r}^{(k+1)}$ 是第 $k+1$ 轮迭代，所有树更新完后的余项

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

假设我们一共进行 K 轮 MCMC 迭代，前 I 轮作为 burn-in 算法，那么最终的预测值是

$$\bar{f}(\mathbf{X}) = \frac{1}{K-I} \sum_{k>I}^{K} f^{(k)}(\mathbf{X}). \tag{7}$$

对同一轮每棵树预测结果求和作为此轮森林的预测结果，对于不同轮森林的预测结果取平均。

4.1.3 其他技术细节

XBART 实际算法计算每个变量的重要性，并且根据重要性抽样选取使用的变量；采用适应性的潜在划分点个数；利用预先对数据排序优化计算过程。这些技术细节请参照 He et al. [2019]。

4.2 模拟结果

在 simulation 中 XBART 取得了不俗的表现：小数据集上取得和 BART 相近的误差，但是远远快过 BART 和 xgboost，在大的数据集上和 xgboost 计算速度相近但是预测误差更好。注意 xgboost 需要 cross validation 选取 gradient，如果不用 cross validation，xgboost 速度非常快，但是预测准确性不佳。

Function	Method	RMSE	Seconds
Linear	XBART	1.74	20
Linear	XGBoost Tuned	2.63	64
Linear	XGBoost Untuned	3.23	< 1
Linear	Random Forest	3.56	6
Linear	BART	1.50	117
Linear	Neural Network	1.39	26
Trig + Poly	XBART	1.31	17
Trig + Poly	XGBoost Tuned	2.08	61
Trig + Poly	XGBoost Untuned	2.70	< 1
Trig + Poly	Random Forest	3.04	6
Trig + Poly	BART	1.30	115
Trig + Poly	Neural Network	3.96	26
Max	XBART	0.39	16
Max	XGBoost Tuned	0.42	62
Max	XGBoost Untuned	0.79	< 1
Max	Random Forest	0.41	6
Max	BART	0.44	114
Max	Neural Network	0.40	30
Single Index	XBART	2.27	17
Single Index	XGBoost Tuned	2.65	61
Single Index	XGBoost Untuned	3.65	< 1
Single Index	Random Forest	3.45	6
Single Index	BART	2.03	116
Single Index	Neural Network	2.76	28

表 1: Low noise case, 信噪比 $1:1$ and $n = 10K$

Function	Method	RMSE	Seconds
Linear	XBART	5.07	16
Linear	XGBoost Tuned	8.04	61
Linear	XGBoost Untuned	21.25	< 1
Linear	Random Forest	6.52	6
Linear	BART	6.64	111
Linear	Neural Network	7.39	12
Trig + Poly	XBART	4.94	16
Trig + Poly	XGBoost Tuned	7.16	61
Trig + Poly	XGBoost Untuned	17.97	< 1
Trig + Poly	Random Forest	6.34	7
Trig + Poly	BART	6.15	110
Trig + Poly	Neural Network	8.20	13
Max	XBART	1.94	16
Max	XGBoost Tuned	2.76	60
Max	XGBoost Untuned	7.18	< 1
Max	Random Forest	2.30	6
Max	BART	2.46	111
Max	Neural Network	2.98	15
Single Index	XBART	7.13	16
Single Index	XGBoost Tuned	10.61	61
Single Index	XGBoost Untuned	28.68	< 1
Single Index	Random Forest	8.99	6
Single Index	BART	8.69	111
Single Index	Neural Network	9.43	14

表 2: High noise case, 信噪比 $1:10$ and $n = 10K$

$\kappa = 1$, **signal to noise ratio 1 to 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)

表 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).

$\kappa = 10$, **signal to noise ratio 1 to 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)

表 4: 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).

参考文献

T. Chen and C. Guestrin. 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, 2016.

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

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

P. R. Hahn, J. Murray, and C. M. Carvalho. Bayesian regression tree models for causal inference: regularization, confounding, and heterogeneous effects. *Confounding, and Heterogeneous Effects (October 5, 2017)*, 2017.

J. He, S. Yalov, and P. R. Hahn. Xbart: Accelerated bayesian additive regression trees. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1130–1138, 2019.

A. R. Linero. Bayesian regression trees for high-dimensional prediction and variable selection. *Journal of the American Statistical Association*, 113(522):626–636, 2018.

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

M. Pratola, H. Chipman, E. George, and R. McCulloch. Heteroscedastic bart using multiplicative regression trees. *arXiv preprint arXiv:1709.07542*, 2017.

R. A. Sparapani, B. R. Logan, R. E. McCulloch, and P. W. Laud. Nonparametric survival analysis using bayesian additive regression trees (bart). *Statistics in medicine*, 35(16): 2741–2753, 2016.