Supplement to "XBART: Accelerated Bayesian Additive Regression Trees"

Jingyu He
University of Chicago

Saar Yalov
Arizona State University

P. Richard Hahn
Arizona State University

Abstract

This abstract contains simulation results that were unable to fit in the main document as well as a comparison on a variety of smaller empirical data sets.

1 Additional Simulation Study

This simulation study is identical to the one in the main text, but is at a smaller sample size of $n = 10,000$, so that it can include two additional methods, the standard MCMC BART model, and random forests, implemented by R packages `dbarts` and `ranger` (Wright and Ziegler, 2015) respectively.

This simulation was computed on an Ubuntu 18.04.1 LTS with Intel i7-8700K Hexa-core 3.7GHz 12MB Cache-64-bit processing, 4.3GHz overclocking speed, as were the simulations from the main paper. The simulation results in Tables 1 and 2 should be fairly self-explanatory, but the overall narrative that emerges is that BART and XBART look quite similar and that on this collection of functions they perform better than the alternatives in terms of function estimation. Further, random forests works well in high noise settings and neural networks work best in low noise settings and in particular on the linear data generating process. In both high noise and low noise case, XBART and BART do similar in terms of root mean squared error (RMSE), but XBART is about 10 times faster than `dbarts`. Further, XBART performs better than XGBoost with cross validation.

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

Table 1: Low noise case, $\kappa = 1$ and $n = 10K$

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

Table 2: High noise case, $\kappa = 10$ 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

2 Real Data Experiments

To compare the performance on real data we recreated the data experiment used in the original BART paper (Chipman et al., 2010). We used 40 datasets of varying sizes. For each dataset, 20 random train test splits were made, yielding a total of 800 experiments. In each experiment we fit XBART, BART, Random Forest, XGBoost with default features, and a tuned XGBoost model with the same tuning parameters choices as in the main. Following (Chipman et al., 2010), we compare the methods using relative RMSE (RRMSE) defined as the RMSE divided by the minimum RMSE achieved, across methods, in each train/test split. Because the smaller data sets permitted more iterations, we set $K = 800$ and $I = 200$, rather than $K = 40$ and $I = 15$ for the larger sample sizes.

The results are reported in Table 3. For reference, the dataset names and dimensions are given in Table 4; see Kim et al. (2007) for more details. Our appraisal is that this collection of datasets is not especially informative, especially in light of our simulation studies where the truth is known. In particular, for most of the 40 data sets there was essentially a three-way tie between random forests, gradient boosting, and XBART. From the simulation study above, this suggests that the data have quite weak signal, as that is the regime where random forests is competitive with the other two methods. BART fit by the standard random walk MCMC does appear to do somewhat better on average, although at three times the computation time. It will be interesting to see if implementing the proper Metropolis-Hastings adjustment brings XBART and BART into closer agreement on these high-noise, low sample size data sets.

Table 3: Results

	BART	XBART	XGB_CV	RF	XGB
Avg. RRMSE	1.067	1.128	1.137	1.139	1.170
Avg. Time	16.930	5.680	6.207	0.385	0.117

Table 4: Data Sets

Name	n	Name	n
Abalone	4177	Fat	252
Ais	202	Fishery	6806
Alcohol	2462	Hatco	100
Amenity	3044	Insur	2182
Attend	838	Labor	2953
Baseball	263	Laheart	200
Baskball	96	Medical	4406
Boston	506	Mpg	392
Budget	1729	Mumps	1523
Cane	3775	Mussels	201
Cardio	375	Ozone	330
College	694	Price	159
Cps	534	Rate	144
Cpu	209	Rice	171
Deer	654	Scenic	113
Diabetes	375	Servo	167
Diamond	308	Smsa	141
Edu	1400	Strike	625
Enroll	258	Tecator	215
Fame	1318	Tree	100

References

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.

Kim, H., Loh, W.-Y., Shih, Y.-S., and Chaudhuri, P. (2007). Visualizable and interpretable regression models with good prediction power. *IIE Transactions*, 39(6):565–579.

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*.