Supplementary material for: Pushing the Envelope of Gradient Boosting Forests via Globally-Optimized Oblique Trees

Abstract

We provide the following. 1) Exploration of deeper trees in XGBoost (section 1). 2) Analysis of tree diversity in GB TAO (section 2). 3) Experiments with different number of trees per GB step (section 3). 4) The effect of the number of TAO iterations on training time and model accuracy (section 4). 5) Comparison with other baselines (section 5). 6) Description of the experiments' setup, for reproducibility: datasets, comparison methods, hyperparameters, etc. (section 6).

1 Exploring deeper trees in XGBoost

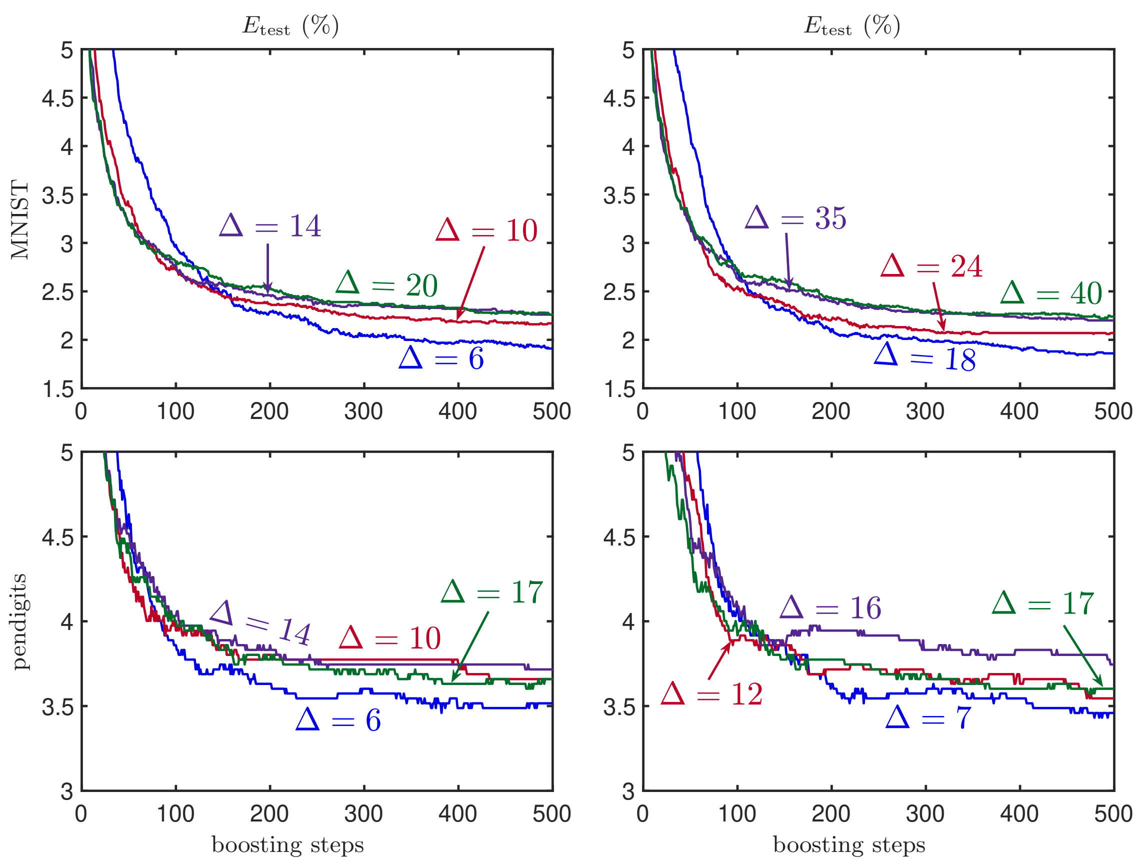

Figure 1: Exploration of different tree depths in XGBoost. The left column uses a `depthwise` tree growing method, where nodes closer to the root are expanded first. The right column uses a `lossguide` tree growing method, where nodes with highest loss change are expanded first, and the tree complexity is controlled by the number of leaves `max_leaves`. Δ is the max depth of the forest.

As it is argued in section 2 of the main paper, axis-aligned forests have limited ability to model higher order feature interactions. The trees must grow very deep in order to capture such high interaction levels. Though in experiments we do consider depth of up to 20 during cross validation in XGBoost, in fig. 1 we explicitly explore the behavior of tree depth. In the left column of fig. 1 we use `depthwise` tree growing option, and control the tree depth via `max_depth`, and in the right column we use a `lossguide` tree growing method, and the complexity of the tree is controlled by `max_leaves`. As the plots clearly indicate, deeper trees result in an overall less accurate ensemble, in spite of having more capacity to model higher order interaction levels. This is in accordance with the XGBoost documentation, which recommend to use shallower trees to mitigate overfitting.

2 Tree diversity in GB TAO

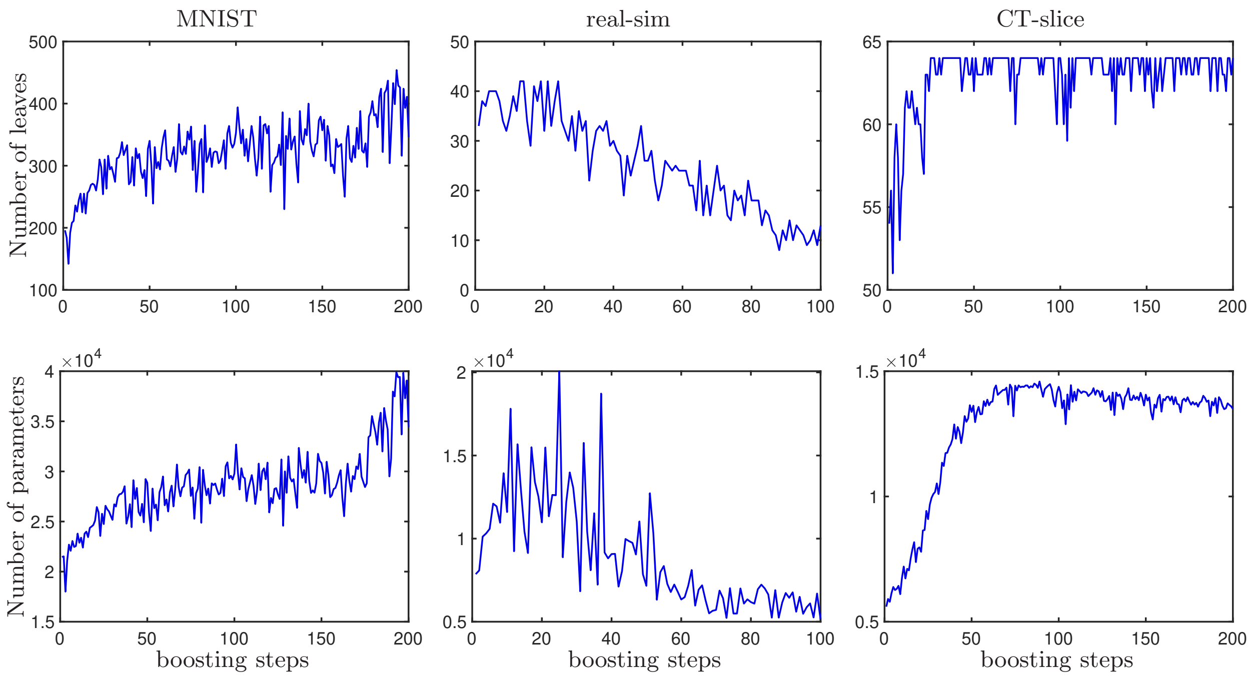

Figure 2: The number of tree leaves (top) and the number of nonzero parameters per tree (bottom) in GB TAO oblique trees.

At each boosting step of GB, TAO starts with some initial, complete tree of depth Δ and random node parameters $\boldsymbol{\Theta}$, and optimizes the objective function GB provides. After performing some fixed number I alternating optimization paths through all the nodes, it performs pruning of dead nodes/subtrees, and so the resulting tree will usually be smaller. In fig. 2 we explore how the individual trees produced at each boosting step differ in terms of the number of leaves and the number of parameters. Interestingly, for the real-sim dataset, as GB steps progress, the resulting trees become much smaller, whereas for MNIST we observe the opposite trend. Unlike XGBoost, which specifically include a penalty term on the number of leaves in the objective function, TAO has an indirect control on the tree complexity through the parameter α of the ℓ_1 penalty term on node parameters. The erratic trend of the curves in fig. 2 clearly indicate for the diversity of the trees in the final forest, which in general is important in ensemble learning.

3 How many trees per boosting step?

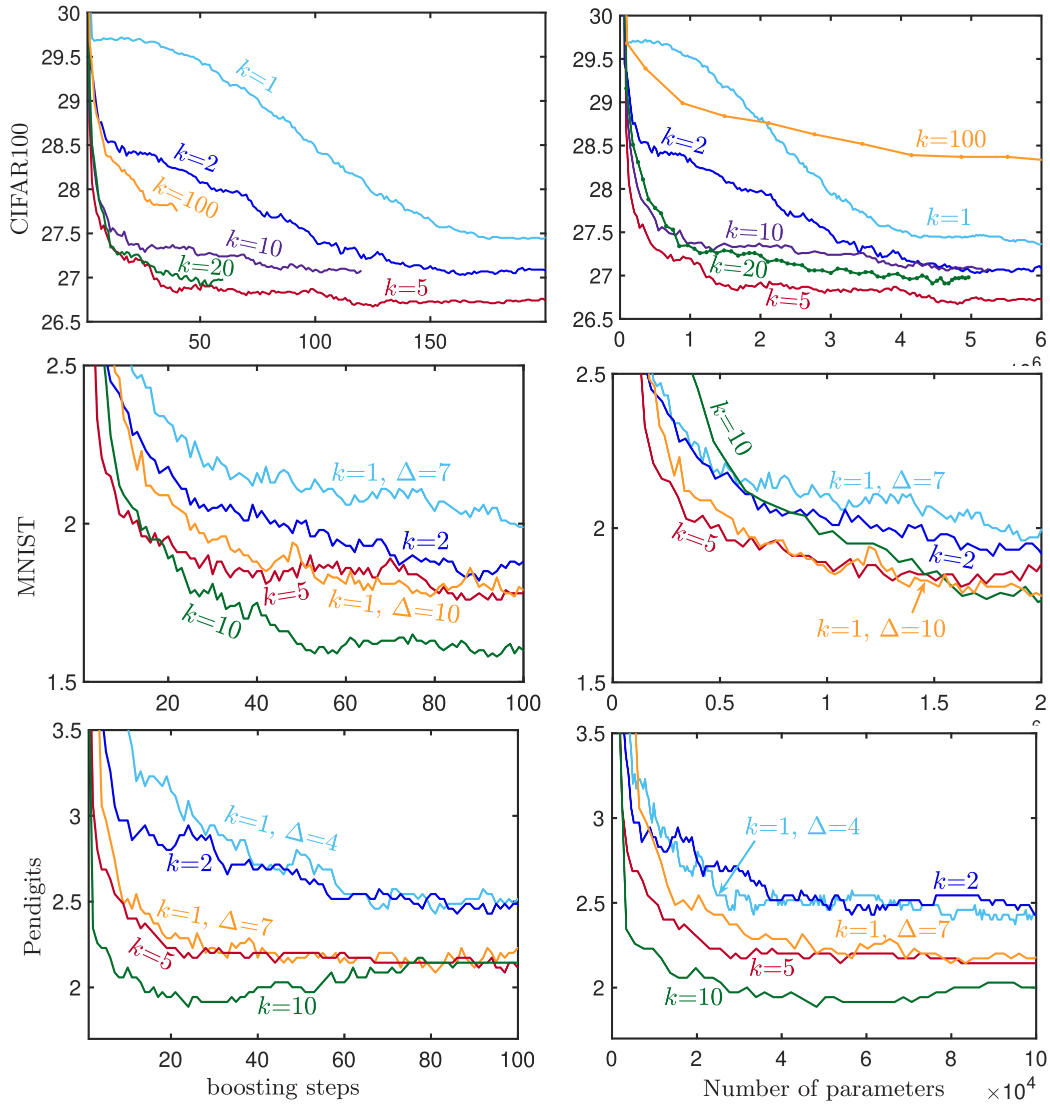

Figure 3: Test error as a function of boosting steps (left column) and as a function of the number of parameters (right column) when using different k number of trees at each boosting step. The tree depth for CIFAR100 is Δ=8, for MNIST Δ=7, and for Pendigits Δ=4. For MNIST and Pendigits we additionally plot the result of using a higher depth Δ when using a single tree k=1.

Cross entropy loss for K-class ($K > 2$) classification in GB requires a base learner to output a real valued K dimensional vector $\boldsymbol{\gamma} \in \mathbb{R}^K$. With diagonal Hessian approximation, a base learner's objective function separates over K, and so there is a choice of using a single tree with vector outputs or K trees with scalar outputs or something in between. In general, K trees have more representation capacity than a single vector valued tree, because the latter can exactly be represented with K trees with the same structure and decision node parameters, but the leaves outputting the corresponding entry of the vector leaves, while the joint partition of the input space produced by K trees cannot in general be represented with a single tree of a reasonable size. Possibly because of this, Friedman's [4] original paper, along with XGBoost and LightGBM, use K trees at each boosting step.

Oblique trees are no different than axis-aligned ones on this regard, except that a single oblique tree trained with TAO is much stronger than a greedily induced axis-aligned tree, and using K oblique trees per GB step can result in the vast increase on the number of parameters. In fig. 3 we explore how using different number of trees k per boosting step affects the test error and model size as measured in terms of the number of nonzero parameters. Unsurprisingly, as the left column shows, for the fixed number of boosting steps, using more trees k per GB step in general produces a more accurate ensemble, however, on the left column we can observe that for a given model size it is not always clear which option is better. Because it is sensible to use a single large tree or K shallow trees per boosting step, one must perform more extensive comparisons to draw some conclusion. In general, this issue of selecting optimal k is more complex, and can fall onto the category of a general model selection problem in machine learning.

4 Training time and the number of TAO iterations

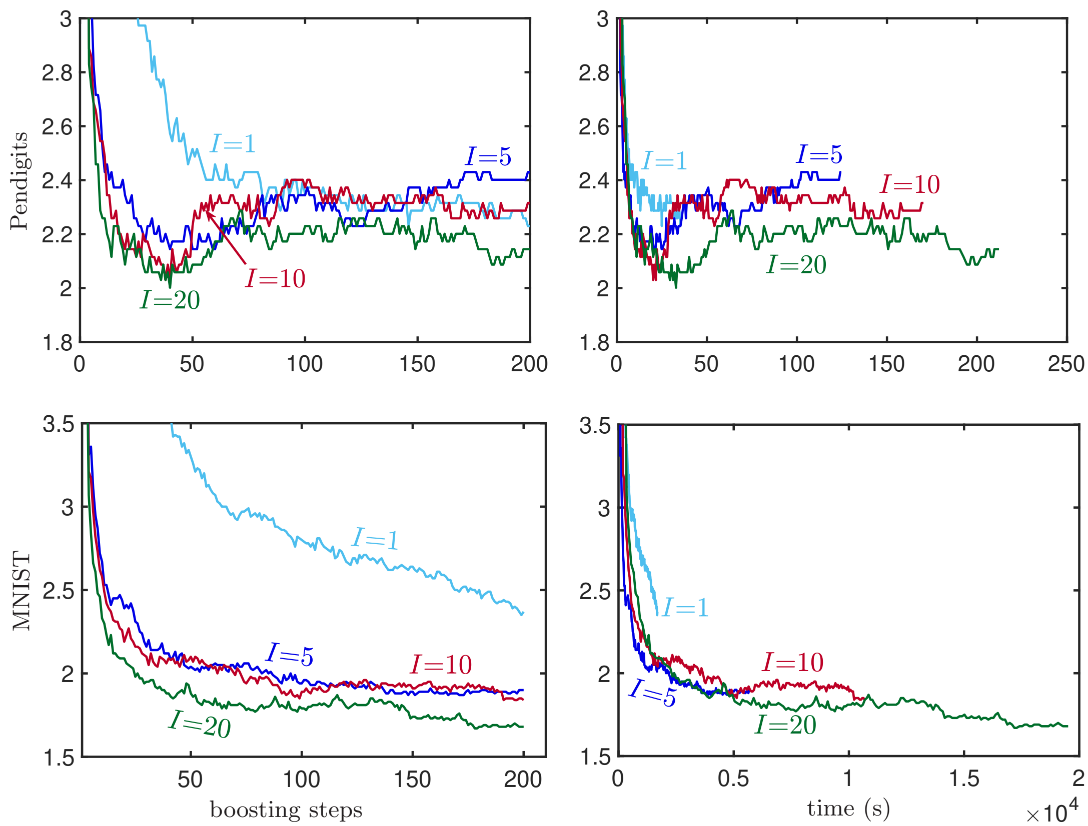

Figure 4: Test error as a function of the number of boosting steps M and training time when using different number of TAO iterations I. All models are trained using parallel processing with 8 threads.

The number of TAO iterations I has a significant effect on training time and model accuracy. In general, one would expect more accurate models from a larger number of TAO iterations I at the expense of longer training time. In fig. 4 we explore this behavior for two datasets. Using a single TAO iteration $I = 1$ results in a very fast training time, but the resulting forests have a significant drop in accuracy. From moderately smaller TAO iterations ($I = 5, 10$), we observe a slight increase in test error, but with a considerable improvement in training time. In practice, if one wants to quickly test the performance of GB with oblique trees, then using fewer TAO iterations I might provide a reasonable picture on model accuracy. However, as the curve of $I = 20$ on MNIST indicate, higher values of TAO iterations I should produce overall more accurate GB forests.

5 Comparison with other baselines

Comparison with GB neural networks (NNs) The focus of our paper was on forests (which are by far the most popular form of GB ensemble). But it would be interesting to compare with GB NNs, as NNs can

	Forest / NNs	E_{test}	#pars.	M	Δ/H	#leav./U
year	GrowNet [2]	8.82±0.01	not provided in [2]			
	GB-TAO	8.73±0.01	402k	100	6	63
CT	GrowNet	5.57±0.23	14M	95	2	192
	GrowNet [2]	5.31±0.35	not provided in [2]			
	GB-TAO	4.61±0.02	778k	50	8	164
cpuact	GrowNet	2.52±0.03	152k	100	2	20
	GB-TAO	2.23±0.02	31k	50	6	48
casp	GrowNet	5.03±0.06	172k	100	4	18
	GrowNet	3.93±0.05	94k	90	2	18
	GB-TAO	3.43±0.00	481k	100	12	603
superc.	GrowNet	9.90±0.32	2M	100	2	81
	GB-TAO	8.76±0.02	573k	50	6	216
	GB-TAO	8.68±0.02	1M	100	6	218
pen	Bagged TAO	2.06±0.05	110k	100	10	95
	GB-TAO	2.00±0.04	44k	30	7	83
MNIST	Bagged TAO	2.37±0.05	2.5M	100	8	124
	Bagged TAO [3]	2.31±0.08	1.2M	30	8	-
	GB-TAO	1.94±0.00	671k	30	10	252
CIFAR10	**GB-TAO**	26.64±0.02	3.3M	200	6	46
	Bagged TAO	26.63±0.03	4.1M	200	8	79
news	Bagged TAO	20.13±0.29	4M	100	8	218
	GB-TAO	18.13±0.00	1M	100	6	35
real-sim	GrowNet	3.54±0.02	17M	20	2	30
	Bagged TAO	2.72±0.04	531k	50	8	16
	Bagged TAO	2.72±0.02	1.1M	100	8	16
	GB-TAO	2.12±0.02	1.3M	20	6	54

Table 1: Comparison with GrowNet (an implementation of GB Neural Network) and Bagged TAO.

also capture higher-order interactions and can directly optimize the objective function of GB. We use the recent GrowNet [2], that employs shallow (1 to 4 hidden layers) multilayer perceptrons as base learners in GB. The two regression datasets in [2] (year and CT) are also used in our paper, and so we cite their results in the table (in CT our train/test splits differ, but we retrain GB-TAO according to their split in the table). For other datasets we use their code (https://github.com/sbadirli/GrowNet) and based on their experimental results [2], we tune the following hyperparameters with grid search: `boost_rate` = {0.1, 1.0}, `epochs_per_stage` = {1, 10}, `lr` = {0.005, 0.01}. We explore MLPs with number of hidden layers $U \in \{2, 4\}$ and number of hidden units $U \in \{\frac{1}{3}D, \frac{1}{2}D, D, 2D\}$ (the same for each layer in GrowNet), where D is the feature dimension. We set the maximum number of boosting steps to 100, but the best step M is cross-validated. GrowNet does not support multiclass losses, so we compare only on regression and binary classification in Table 1.

Comparison with Bagged TAO [3] uses bagging with oblique decision trees trained with TAO for classification. From [3] we cite the result of MNIST, and for other classification datasets we run bagged TAO ourselves. We follow the experimental setup and recommendation in [3]: TAO minimizes a 0-1 loss with a small sparsity penalty (α=0.01) and is trained on a 90% random sample for 40 iterations. We cross-validate the depth Δ={6,8,10} and the number of trees M={30, 50, 100, 200}.

As the Table 1 shows, GB TAO clearly outperforms GrowNet and (with one exception where it is comparable) bagged TAO in accuracy, often with fewer parameters.

6 Experimental setup

We implemented GB-TAO in C++. For all the experiments in GB TAO we start with a complete tree of depth Δ and random initial parameters (Gaussian (0,1)). Though it is possible to initialize a tree with some heuristic or from CART, we use random parameters at each GB step to induce more diversity into the ensemble. We parallelize the optimization over the nodes at a given depth using OpenMP. When solving a reduced problem at a decision node, we use an ℓ_1 regularized logistic regression in LIBLINEAR solver of version 2.43. α is a hyperparameter in GB TAO so ideally it should be cross-validated. To save training time, instead of cross-validating it for the whole forest, we cross-validate α only for a single tree. This simple choice already results in leading performance in our experiments. Specifically, we train a single tree for 3 choices of α (0.01, 0.1, 1.0) and choose the α that gives a most accurate tree on a validation set. The set of depth Δ parameters we evaluate during cross validation is $\{4, 6, 8, 10, 12\}$. The set of the number of boosting steps M we consider is $\{1, 5, 10, 20, 30, 100, 200\}$. Because of the longer runtime, we select only 2-3 of values from those sets depending on the complexity of the dataset. All the experiments are performed on Intel(R) Xeon(R) CPU E5-2699 v3 with 256GB memory.

6.1 Baselines

To compare models of different size, for the baselines we fix the number of boosting steps M (or number of trees T in SPORF) to $\{10, 100, 300, 500, 1000\}$, and perform grid search over other hyperparameters.

XGBoost We use a Python package of version 1.4.1. We use the exact `tree_method`. During cross validation we perform grid search over the following hyperparameter values: `max_depth` = $\{4, 6, 8, 10, 20\}$, `eta` = $\{0.01, 0.05, 0.1, 0.3\}$.

LightGBM We use a Python package of version 3.2.1. During cross validation we perform grid search over the following hyperparameter values: `num_leaves` = $\{16, 31, 64, 128, 256, 512\}$, `learning_rate` = $\{0.01, 0.05, 0.1, 0.3\}$.

scikit-learn We use a version 0.24.1. We use `GradientBoostingClassifier` and `GradientBoostingRegressor` classes (not histogram versions). During cross validation we perform grid search over the following hyperparameter values: `max_depth` = $\{4, 6, 10, 14\}$, `learning_rate` = $\{0.05, 0.1\}$.

SPORF We use a Python package of version 2.0.5. During cross validation we perform grid search over the following hyperparameter values: `projection_matrix` = {RerF, S-RerF}, `max_depth` = {10, 20, None}, `max_features` = {sqrt, log2, None}.

A full exploration of all hyperparameters of the baselines is infeasible. But we attempt to do more thorough grid search for small scale cpuact and pendigits datasets over the following for XGBoost: `max_depth` = $\{1,4,6,8\}$, `eta` = $\{0.01,0.05,0.1,0.3\}$, `gamma` = $\{0,1,2,10\}$, `min_child_weight` = $\{0.0,1.0,5.0\}$, `subsample` = $\{0.5,0.7,1.0\}$, `colsample_bytree` = $\{0.5,0.7,1.0\}$. We set the number of boosting steps to 5000 M, and use early stop based on validation. While the results slightly improved over those in the main paper, GB TAO still wins by a large margin. We also used the Bayesian optimization package `hyperopt`, but the results were worse.

6.2 Esimation of model size

For axis-aligned GB forests (XGBoost, LightGBM, and GB-sklearn) we count the number of parameters as follows: we sum the number of parameters of each node of all the trees in the forest, where an axis-aligned split node counts for two parameters (feature index and threshold) and a constant leaf counts for one parameter. In GB TAO we exactly estimate the number of nonzero parameters at a decision node. The interface of SPORF does not provide explicit access to tree parameters, and so we provide a reasonable upper bound: the `max_features` parameter controls how many features are used at a decision node, so by assuming that each split node uses exactly `max_features` parameters we estimate the total number of parameters in SPORF. We estimate FLOPS as an average number of nonzero parameters a test point encounters during forest inference. Except for GB TAO, we provide an upper bound on FLOPS by assuming that each test point reaches a maximum depth.

Dataset	N_{train}	N_{test}	D	$\bar{D}_{\text{nnz}}$	K
pendigits	7 494	3 498	16	16	10
MNIST	60 000	10 000	784	150	10
CIFAR100 (VGG16 feats)	50 000	10 000	512	324	100
News20	15 935	3 993	62 061	80	20
real-sim	50 616	21 693	20 958	51	2

Table 2: Specs of the datasets used in our experiments for classification. N is a sample size, D is a feature dimension size, $\bar{D}_{\text{nnz}}$ is the average number of nonzero features, and K is the number of classes.

6.3 Classification datasets

MNIST a standard benchmark. We use direct pixel intensities scaled between 0 and 1 as input [7].

Pendigits a digit recognition dataset, where inputs are resampled x and y pixel coordinates recorded from a pressure sensitive tablet [1]. We preprocess the inputs to have zero mean and variance one. Obtained from the UCI Machine Learning Repository [8].

CIFAR100 a standard image classification benchmark in computer vision. We use as features the output of the last convolutional layer of a pretrained VGG16 network [6].

News20 a standard document classification benchmark. The features are normalized word counts. Obtained from a LIBSVM multiclass data collection [1].

real-sim a document classification dataset. The features are normalized word counts. Obtained from a LIBSVM binary data collection [2].

6.4 Regression datasets

cpuact the task is to predict the percentage of time a CPU spends in user mode. The features consist of various statistics of memory and other operations. Obtained from the Delve project [3].

CT slice the task is to predict the relative location of the CT slice on the axial axis of the human body. The features are histograms describing bone structures and air inclusions. Obtained from the UCI Machine Learning Repository [8].

Superconductivity the task is to predict the critical temperature of a superconductor from the features extracted based on the chemical formula such as thermal conductivity, atomic radius, valence, electron affinity, and atomic mass [5]. Obtained from the UCI Machine Learning Repository [8].

CASP a dataset of Physicochemical Properties of Protein Tertiary Structure. The task is to predict the size of the residue. Obtained from the UCI Machine Learning Repository [8].

Year Prediction MSD the task is to predict the release year of a song from audio features. Obtained from the UCI Machine Learning Repository [8].

7 Extended table results

References

[1] F. Alimoglu and E. Alpaydin. Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96)*, 1996.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html
[2] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
[3] http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html

Dataset	N_{train}	N_{test}	D	$\bar{D}_{\text{nnz}}$
cpuact	4 915	3 277	21	21
CASP	29 999	15 731	9	9
CT slice	42 800	10 700	384	378
Superconductivty	17 010	4 253	81	81
Year prediction MSD	463 715	51 630	90	90

Table 3: Similar to Table 2, but for regression datasets. The output is 1D for all datasets.

[2] S. Badirli, X. Liu, Z. Xing, A. Bhowmik, K. Doan, and S. S. Keerthi. Gradient boosting neural networks: GrowNet. arXiv:2002.07971, 2020.

[3] M. Á. Carreira-Perpiñán and A. Zharmagambetov. Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting. In *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, pages 35–46, Seattle, WA, Oct. 19–20 2020.

[4] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29 (5):1189–1232, 2001.

[5] K. Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346–354, 2018.

[6] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Dept. of Computer Science, University of Toronto, Apr. 8 2009.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

[8] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[9] Y. Shi, J. Li, and Z. Li. Gradient boosting with piece-wise linear regression trees. In *Proc. of the 28th Int. Joint Conf. Artificial Intelligence (IJCAI'19)*, pages 3432–3438, Macao, China, Aug. 10–16 2019.

	Forest	E_{test} (%)	E_{train} (%)	#pars.	FLOPS	M	k	Δ	leaves
MNIST (60k,784,10)	XGBoost	4.38±0.00	0.05±0.00	70 795	(1 000)	10	10	10	236.7
	GB-TAO	4.17±0.08	1.90±1.10	21 038	2 295	1	1	12	192
	LightGBM	3.73±0.00	0.00±0.00	149 227	(3 515)	10	10	35	498.1
	SPORF	3.08±0.11	0.00±0.00	(14 369 600)	(128 800)	100	1	46	4956
	SPORF	2.95±0.06	0.00±0.00	(42 987 000)	(394 800)	300	1	47	4942
	SPORF	2.89±0.04	0.00±0.00	(143 493 000)	(1 400 000)	1000	1	50	4949
	GB-TAO	2.33±0.00	0.00±0.00	199 994	(23 193)	10	1	10	209
	XGBoost	2.20±0.00	0.00±0.00	106 759	(5 992)	100	10	6	36.3
	LightGBM	2.02±0.00	0.00±0.00	120 811	(10 085)	100	10	10	40.9
	GB-sklearn	1.96±0.03	0.00±0.00	1 232 860	51966	100	10	10	42
	GB-TAO	1.94±0.00	0.00±0.00	671 344	(71 124)	30	1	10	252
	XGBoost	1.91±0.00	0.00±0.00	404 525	(29 290)	1000	10	6	27.6
	GB-TAO	1.65±0.02	0.00±0.00	3 046 165	846 133	50	10	7	37
	LightGBM	1.62±0.00	0.00±0.00	642 034	(84 874)	1000	10	21	22
	GB-TAO	1.55±0.02	0.00±0.00	7.2M	2M	140	10	7	34
pendigits (7.5K,16,10)	XGBoost	5.15±0.00	0.08±0.00	8 701	(772)	10	10	8	30
	LightGBM	4.92±0.00	0.15±0.00	9 055	(1121)	10	10	11	31
	GB-sklearn	4.19±0.01	0.0±0.00	168 911	(600)	100	10	6	57
	SPORF	4.00±0.02	0.03±0.01	(16 560)	(760)	10	1	19	332
	GB-sklearn	3.66±0.04	0.01±0.00	16 932	(600)	1 000	10	6	57
	XGBoost	3.52±0.00	0.00±0.00	57 018	(9679)	300	10	4	7
	LightGBM	3.49±0.00	0.00±0.00	89 692	(8177)	100	10	11	31
	XGBoost	3.46±0.00	0.00±0.00	18 442	(3157)	100	10	4	7
	XGBoost	3.46±0.00	0.00±0.00	137 176	(25451)	1000	10	4	5
	LightGBM	3.43±0.00	0.00±0.00	96 702	(9844)	300	10	8	31
	LightGBM	3.31±0.00	0.00±0.00	895 366	(41311)	1000	10	4	31
	GB-TAO	3.15±0.25	0.08±0.03	1324	104	1	1	8	60
	SPORF	2.91±0.09	0.00±0.00	(1 646 000)	(80 000)	1000	1	20	330
	SPORF	2.87±0.01	0.00±0.00	(105 600)	(8 000)	100	1	20	212
	GB-TAO	2.17±0.02	0.00±0.00	13 341	991	10	1	7	65
	GB-TAO	2.00±0.04	0.01±0.00	44 456	3 031	30	1	7	83
CIFAR100 (50k,512,100)	GB-sklearn	32.64±0.03	0.00±0.00	502 459	(36 174)	100	100	6	17
	XGBoost	30.20±0.00	0.00±0.00	133 261	(27 828)	100	100	4	5
	LightGBM	31.47±0.00	0.00±0.00	459 970	(81033)	100	100	14	16
	LightGBM	30.32±0.00	0.00±0.00	1 033 235	(140558)	143	100	18	25
	XGBoost	30.15±0.00	0.00±0.00	174 238	(19035)	1000	100	6	1.2
	GB-TAO	29.38±0.04	0.00±0.00	39 410	2 491	1	1	12	178
	SPORF	28.62±0.07	0.00±0.00	(26 110)	(1 800)	10	1	20	262
	SPORF	27.07±0.02	0.00±0.00	(106 100)	(9 000)	100	1	10	107
	GB-TAO	26.98±0.04	0.00±0.00	1 227 547	324 825	30	5	8	33
	GB-TAO	26.86±0.02	0.00±0.00	2 086 469	539 038	50	5	8	33
	SPORF	26.71±0.05	0.00±0.00	(530 500)	(45 000)	500	1	10	107
	GB-TAO	26.64±0.02	0.00±0.00	3 345 599	678 351	200	2	6	46

Table 4: As Table 1 in the main paper, but with more details.

	Forest	E_{test} (%)	E_{train} (%)	#pars.	FLOPS	M	k	Δ	leaves
news20 (16k,62k,20)	**GB-TAO**	27.75±0.00	4.12±0.00	18 662	6 900	1	1	6	61
	GB-sklearn	27.60±0.04	8.47±0.03	168 593	(2 800)	10	20	14	282
	XGBoost	27.25±0.00	5.89±0.00	67 421	(4 000)	10	20	20	113
	LightGBM	25.90±0.00	11.19±0.00	18 200	(4 705)	10	20	24	31
	GB-sklearn	23.42±0.03	2.32±0.02	155 984	(12 000)	100	20	6	27
	SPORF	22.51±0.10	2.58±0.04	(110 000 100)	(14 168 100)	100	1	569	4401
	XGBoost	22.19±0.00	3.30±0.00	83 909	(12 000)	100	20	6	15
	GB-sklearn	21.71±0.02	2.22±0.02	346 821	(36 000)	300	20	6	20
	SPORF	21.65±0.26	2.49±0.01	(1 103 501 000)	(142 179 000)	1000	1	571	4415
	XGBoost	21.39±0.00	2.50±0.00	704 948	(120000)	1000	20	6	12
	XGBoost	21.34±0.00	2.30±0.00	187 626	(36000)	300	20	6	11
	LightGBM	20.69±0.00	2.49±0.00	1 820 000	(539183)	1000	20	27	31
	LightGBM	20.01±0.00	2.25±0.00	182 000	(57075)	100	20	28	31
	GB-TAO	19.84±0.02	2.22±0.01	534 327	195323	30	1	6	61
	LightGBM	19.78±0.00	2.27±0.00	546 000	(166448)	300	20	28	31
	GB-TAO	18.13±0.05	2.21±0.01	478 900	415 652	10	20	4	8
	GB-TAO	18.76±0.01	2.22±0.00	745 575	293 651	50	1	6	52
	GB-TAO	18.13±0.00	2.22±0.00	1 002 318	461 802	100	1	6	35
	GB-TAO	16.65±0.04	2.21±0.03	1 673 838	1 484 054	40	20	4	8
real-sim (51k,21k,2)	XGBoost	7.68±0.00	6.01±0.00	6 199	(200)	10	1	20	207
	GB-sklearn	5.65±0.05	2.68±0.04	150 547	(1400)	100	1	14	502
	SPORF	5.34±0.08	0.72±0.03	(6 278 010)	(996 150)	10	1	670	4435
	GB-sklearn	4.72±0.03	0.09±0.00	247 821	(4200)	300	1	14	276
	GB-sklearn	4.25±0.02	0.01±0.00	421 642	(14000)	1000	1	14	141
	SPORF	4.14±0.05	0.24±0.00	(62 780 100)	(9 961 500)	100	1	687	4301
	SPORF	4.06±0.03	0.23±0.00	(188 953 500)	(30 015 000)	300	1	690	4315
	XGBoost	3.54±0.00	0.99±0.00	24 202	(2000)	100	1	20	81
	GB-TAO	3.44±0.24	0.16±0.00	17 674	11 212	1	1	6	42
	XGBoost	3.41±0.00	0.75±0.00	22 908	(3000)	300	1	10	26
	LightGBM	3.31±0.00	0.01±0.00	27 300	(8614)	300	1	29	31
	XGBoost	3.31±0.00	0.42±0.00	100 597	(14000)	1000	1	14	34
	GB-TAO	3.12±0.00	0.16±0.00	52 564	39 302	5	1	4	15
	LightGBM	3.05±0.00	0.42±0.00	100 597	(14000)	1000	1	30	31
	GB-TAO	2.77±0.00	0.14±0.00	113 426	84 899	10	1	4	15
	GB-TAO	2.41±0.00	0.14±0.00	422 635	317 967	30	1	4	15
	GB-TAO	2.12±0.02	0.14±0.00	1 319 368	566 360	20	1	6	54

Table 5: As Table 2 in the main paper, but with more details.

	Forest	E_{test} (%)	E_{train} (%)	#pars.	FLOPS	T	Δ	leaves
cpuact (16k,62k,20)	XGBoost	3.74±0.00	3.30±0.00	1 021	(60)	10	6	35
	XGBoost	2.60±0.00	0.54±0.00	60 211	(1000)	100	10	201
	XGBoost	2.55±0.00	0.75±0.00	39 231	(1800)	300	6	201
	XGBoost	2.51±0.00	1.13±0.00	41 662	(4000)	1000	4	15
	LightGBM	2.51±0.00	1.51±0.00	5 686	(179)	10	23	190
	GB-sklearn	2.51±0.06	1.33±0.00	14 377	(600)	100	6	48
	GB-TAO	2.42±0.02	1.96±0.01	17 689	2 905	30	6	46
	GB-sklearn	2.41±0.04	0.68±0.00	42 754	(4000)	1000	4	15
	LightGBM	2.27±0.00	0.99±0.00	19 000	(1875)	100	34	64
	LightGBM	2.26±0.00	1.15±0.00	27 300	(3581)	300	21	31
	LightGBM	2.25±0.00	0.50±0.00	91 000	(12468)	1000	21	31
	GB-TAO	2.23±0.02	1.61±0.01	31 158	4 981	50	6	48
CT-slice (43k,384)	LightGBM	2.11±0.00	1.28±0.00	15 340	(370)	10	42	512
	LightGBM	1.53±0.00	0.41±0.00	153 400	(5 543)	100	107	512
	LightGBM	1.52±0.00	0.68±0.00	91 000	(12557)	1 000	23	31
	LightGBM	1.45±0.00	0.39±0.00	229 800	(11811)	300	79	256
	GB-sklearn	1.43±0.02	0.66±0.01	192 208	(1 000)	100	10	641
	XGBoost	1.50±0.00	0.73±0.00	106 897	(1 000)	100	10	357
	GB-TAO	1.28±0.02	1.05±0.01	28 261	2 083	1	8	223
	GB-sklearn	1.26±0.03	0.07±0.01	900 640	(10 000)	1000	10	301
	XGBoost	1.26±0.00	0.19±0.00	767 227	(1 000)	1000	10	256
	GB-TAO	0.90±0.02	0.61±0.02	81 466	24 571	30	4	16
	GB-TAO	0.52±0.01	0.27±0.00	475 253	71 960	50	6	61
	GB-TAO	0.45±0.01	0.13±0.00	1 179 507	160 147	100	6	64
casp (45k,9)	**GB-TAO**	4.38±0.03	4.11±0.02	4 113	107	1	12	430
	XGBoost	3.66±0.00	2.12±0.00	118 957	(1 000)	100	10	397
	GB-sklearn	3.65±0.02	1.51±0.00	727 282	(1 400)	100	14	2424
	XGBoost	3.61±0.00	1.36±0.00	267 585	(3 000)	300	10	297
	XGBoost	3.58±0.00	0.99±0.00	793 174	(10 000)	1000	10	265
	GB-sklearn	3.58±0.01	0.34±0.01	854 104	(10 000)	1000	10	285
	LightGBM	3.54±0.00	1.55±0.00	153 400	(5 297)	100	114	512
	LightGBM	3.53±0.00	1.87±0.00	229 800	(10 256)	300	80	256
	GB-TAO	3.49±0.01	2.76±0.02	255 985	5 243	50	12	645
	LightGBM	3.48±0.00	0.76±0.00	766 000	(43 440)	1000	109	256
	GBDT-PL [9]	3.46±0.00	--	-	-	-	-	-
	GB-TAO	3.43±0.00	2.53±0.01	480 752	10 267	100	12	603
	GB-TAO	3.39±0.01	2.31±0.01	886 707	19 955	200	12	552
superconduct (43k,384)	**GB-TAO**	11.02±0.10	6.11±0.01	8410	485	1	10	466
	GB-sklearn	9.38±0.01	4.51±0.01	138 658	(6 000)	1000	6	47
	XGBoost	9.20±0.00	5.41±0.00	130 465	(6 000)	1000	6	44
	GB-sklearn	9.14±0.03	4.74±0.02	128 821	(1 000)	100	10	430
	XGBoost	8.98±0.00	5.64±0.00	132 040	(1 000)	100	10	441
	GBDT-PL [9]	8.80±0.00	--	-	-	-	-	-
	LightGBM	8.77±0.00	6.00±0.00	38 200	(2 667)	100	45	128
	GB-TAO	8.76±0.02	6.52±0.02	572 841	29 974	50	6	216
	LightGBM	8.73±0.00	4.90±0.00	190 000	(18 477)	1000	39	64
	GB-TAO	8.68±0.02	6.11±0.01	1 095 134	58 636	100	6	218
year (450k,90)	**GB-TAO**	9.17±0.01	8.67±0.01	18 725	715	1	8	252
	XGBoost	9.05±0.00	7.75±0.00	153 226	(849)	100	10	511
	LightGBM	9.03±0.00	6.88±0.00	153 400	(2 586)	100	37	512
	GB-sklearn	9.03±0.02	7.19±0.01	247 987	(1 000)	100	10	827
	XGBoost	9.00±0.00	6.20±0.00	567 984	(2 849)	300	10	632
	LightGBM	8.92±0.00	6.21±0.00	460 200	(8 026)	300	43	512
	LightGBM	8.92±0.00	3.96±0.00	1 534 000	(25 622)	1 000	43	512
	XGBoost	8.91±0.00	5.31±0.00	1 822 273	(9 694)	1000	10	608
	GB-TAO	8.88±0.02	8.67±0.01	78 127	10 076	20	6	62
	GB-TAO	8.81±0.02	8.50±0.01	118 608	14 976	30	6	62
	GB-TAO	8.77±0.01	8.31±0.01	199 615	24 683	50	6	62
	GB-TAO	8.73±0.01	8.01±0.01	401 719	48 592	100	6	63

Table 6: As Table 3 in the main paper, but with more details.