An Experimental Comparison of Old and New Decision Tree Algorithms

Arman Zharmagambetov [*] Suryabhan Singh Hada [*]
Miguel Á. Carreira-Perpiñán Magzhan Gabidolla
Dept. of Computer Science and Engineering, University of California, Merced
http://eecs.ucmerced.edu

March 18, 2020

Abstract

This paper presents a detailed comparison of a recently proposed algorithm for optimizing decision trees, tree alternating optimization (TAO), with other popular, established algorithms. We compare their performance on a number of classification and regression datasets of various complexity, different size and dimensionality, across different performance factors: accuracy and tree size (in terms of the number of leaves or the depth of the tree). We find that TAO achieves higher accuracy in nearly all datasets, often by a large margin.

1 Introduction

Decision trees are one of the widely used statistical models. Apart from being a good classifier, they have some very unique properties which separate them from other models. A path from root to any leaf can be described as a sequence of decisions: '$x_i < b$' (axis-aligned trees) or $\mathbf{w}^T\mathbf{x} > b$ (oblique tree). This not only makes decision trees very fast at inference but also makes them good interpretable models. This sequence of decisions can be used as IF-THEN rules to understand the prediction of the model for a given input.

However, there is one major drawback with decision trees: learning a tree from data is a very difficult optimization problem, involving a search over a complex, large set of tree structures, and over the parameters at each node. In fact, it is well-known that the optimal binary tree construction problem is NP-hard (Hyafil and Rivest, 1976). Recently, Carreira-Perpiñán and Tavallali (2018) proposed Tree Alternating Optimization (TAO) algorithm to improve this problem, where authors directly optimize the misclassification error, using alternating optimization over separable subsets of nodes. In this work, we compare TAO against some well-known decision tree algorithms (both axis-aligned and oblique) over a wide range of classification and regression datasets.

We have structured this paper in the following way: In section 2 we briefly describe all the algorithms that we use for the comparison. Next, in section 3.2 we describe all the data sets including the number of instances and dimensionality for each data set. In section 3 and 4, we describe our experimental setup and results of the comparison.

2 The algorithms

The literature of tree learning both for classification and regression is immense. In this paper, we restrict our comparison to the most popular and established methods for training axis-aligned and oblique decision trees with constant leaves. Below we provide a short description of the algorithms which were used in our comparison. More details can be found on the corresponding cited papers.

[*] equal contribution

- **CART:** CART (Breiman et al., 1984) is one of the most widely used algorithms for training axis-aligned decision trees. It learns the tree by greedy recursive partitioning, to optimize the impurity measure at each node. At each growing stage for a given node, it enumerates through all the attributes to find the best split that reduces the Gini-index for that node. It grows the tree up to the max depth and then starts pruning nodes one by one such that it does not increase the misclassification error by a certain threshold.

- **C5.0:** Quinlan (1993) is known as an established univariate decision tree learning software. Similarly to CART, it uses a greedy recursive partitioning of the tree nodes. At each recursive split, the algorithm enumerates over different feature-threshold combinations and picks the best one according to the information gain criterion. Pruning can be applied once the tree growing phase is finished. We use this version of the decision tree learning algorithm developed by Ross Quinlan since it is the latest and extended version of the earlier ID3 and C4.5 algorithms (Quinlan, 1993).

- **OC1:** Oblique Classifier 1 (OC1) (Murthy et al., 1993) is also based on the idea of greedily growing a large tree to minimize some impurity measure, and pruning it using the same method as in CART. However, it considers only oblique hyperplanes in internal nodes, where all the features are used in the decision making. To find the weights of the those nodes, it starts with a random guess, and iteratively perturbs one weight at a time until impurity measure does not improve. Then, the weights are changed in some random direction with the aim of moving out from a local minimum. The random movements that help to improve impurity are further optimized using the iterative weight perturbation. The initial random guess procedure and the subsequent movements in random direction are repeated some predetermined number of times, which is specified by the user. The idea behind this method is similar to multivariate version of CART (i.e. coordinate descent over weights) but it picks the best of several random restarts.

- **GUIDE:** Generalized, Unbiased, Interaction Detection and Estimation (GUIDE) is the latest of the algorithms developed by Wei-Yin Loh (2002, 2009), which improves upon his previous algorithms (Loh, 2014). GUIDE shares a similar tree inducing strategy as CART: greedily growing a large tree, and pruning it by cross validation. The major difference lies in the way features are selected and the choice of the split point. GUIDE utilizes various statistical tests in a single feature selection (axis-aligned) during node splitting with the aim of eliminating variable selection bias. GUIDE for classification also provides an option for bivariate linear splits in the nodes using only two features at a time, which are selected based on the chi-squared statistic of two variables.

- **OCT:** Bertsimas and Dunn (2017) have recently proposed to formulate a tree induction algorithm as an mixed-integer optimization that finds globally optimum tree and can be solved by using MIO solvers. However, this method has an exponential time complexity in the worst-case and thus can be applied only to small trees (usually up to depth 4). Otherwise, some early stopping criterion should be used which does not guarantee any optimality. We refer to this method as Optimal Classification Trees (OCT).

- **CO2:** Norouzi et al. (2015) formulate a convex-concave upper bound on the tree's empirical loss and optimize that loss using stochastic gradient descent. The initial tree structure must be provided and usually initialized using greedy procedure (similar to CART). The use of SGD enables efficient optimization for large scale datasets.

- **TAO:** The TAO algorithm proposed in Carreira-Perpiñán and Tavallali (2018) optimizes a decision tree with predetermined structure and can be trained to minimize the desired objective function such as misclassification error. Each iteration of TAO is guaranteed to decrease or leave unchanged the objective function. The algorithm can be applied to both axis aligned and oblique decision trees. Moreover, the algorithm can handle various penalty terms on objective function such as ℓ_1-regularization which we briefly describe here (see Carreira-Perpiñán and Tavallali (2018) for details). TAO assumes a given tree structure with initial parameter values (possibly random), and minimizes the following objective function jointly over the parameters $\mathbf{\Theta} = \{\boldsymbol{\theta}_i\}$ of all nodes i of the tree:

$$E(\mathbf{\Theta}) = \sum_{n=1}^{N} L(y_n, T(\mathbf{x}_n; \mathbf{\Theta})) + \lambda \sum_{\text{nodes } i} \|\mathbf{w}_i\|_1 \tag{1}$$

where $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \{1, \dots, K\}$ is a training set of D-dimensional real-valued instances and their labels (in K classes), $L(\cdot, \cdot)$ is the loss function (e.g. cross-entropy, MSE, 0/1 loss, etc.) and $T(\mathbf{x}; \mathbf{\Theta})$: $\mathbb{R}^D \rightarrow \{1, \dots, K\}$ is the predictive function of the tree and $\boldsymbol{\theta}_i$ is parameters at a node i. For example, in case of oblique decision nodes, $\boldsymbol{\theta}_i$ is a hyperplane with weight vector $\mathbf{w}_i \in \mathbb{R}^D$ and bias $b_i \in \mathbb{R}$, which thus sends an input instance $\mathbf{x}$ down its right child if $\mathbf{w}_i^T \mathbf{x} \geq b_i$ and down its left child otherwise.

The basis of the TAO algorithm is given by the separability condition theorem. It states that for any nodes i and j (internal or leaves) that are not descendants of each other (e.g. all nodes at the same depth) the error $E(\mathbf{\Theta})$ in eq. (1) separates over $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$. Since the loss function now separates algorithm can optimize eq. (1) over each node separately. This much simpler problem is referred as a "reduced problem". TAO algorithm applies alternating optimization over separable subsets of nodes:

- Optimizing over internal nodes is equivalent to optimizing a weighted binary classifier over $\boldsymbol{\theta}_i$ over the training instances $\{(\mathbf{x}_n, \overline{y}_n)\}$ that currently reach node i. Each such instance $\mathbf{x}_n$ is assigned a weight and pseudo label $\overline{y}_n \in \{-1, +1\}$ based on the child whose subtree gives the better prediction for $\mathbf{x}_n$. Specifically, we send $\mathbf{x}_n$ to the left and right subtrees. All parameters in those subtrees are fixed and depending on which one gives correct (or better) output we assign a pseudo label (either -1 or $+1$). This pseudo label indicates where to send the given instance (either `left` or `right`). We also assign a weight per sample because the loss of the best child is different for each instance.
- Optimizing over a leaf which is a K-class classifier on the training points that reach that particular leaf. In this paper, we focus on constant leaves. Therefore, the solution, in this case, will be the majority label of the training points that reach leaf i.

Depending on the decision node type, the first step can be solved either exactly (for axis-aligned nodes the best split is determined by enumerating over all the features) or approximated by surrogate loss (for oblique nodes it solves the linear binary classification problem).

3 Experimental setup

3.1 Algorithm-specific

Below we describe each algorithm-specific experimental setup:

- **CART-R:** We use R implementation of CART called `rpart` (Therneau et al., 2019). For each dataset during training we let the tree grow up to the max allowed depth of 30 (max-depth constraint by `rpart`). For this, we set the "minsplit" parameter to 1 and the complexity parameter ("cp") to 0. Once the tree is fully grown we use `rpart` internal k-fold cross-validation (k=10), to get list of pruning parameters and choose best pruning parameter based on SE-1 rule (as suggested by `rpart` documentation). We report tests and train accuracy of the pruned tree.
- **CART-P:** We also use the Python implementation of CART provided by scikit-learn (Pedregosa et al., 2011, version 0.22.2). For each dataset, during training we let the tree grow full i.e. training error is zero. For this, we set the "minsplit" parameter to 1 and the complexity parameter ("ccp_alpha") to 0. Next, we find the best pruning parameter using k-fold cross-validation.
- **C5.0:** We use the single-threaded Linux version of the C5.0 (provided by authors[1]) written in C language. For each of the datasets, we apply a grid search on the k-fold validation set to get the

[1] https://rulequest.com/download.html

best parameters. Specifically, we tune "-c CF" which controls the pruning severity and "-m cases" which is the minimum number of points to perform a node split. We use the default options for all other parameters. It worth to mention that empirically we have found that in many cases the tuned parameters are not far away from the default settings.

- **GUIDE:** We use the GUIDE version 32.3 provided by the authors in the form of executables[2]. To run the executable, one needs to provide an input file, with all the parameters listed in a specific order. For *axis-aligned classification*, we specify option (1) which gives univariate splits a higher priority, while for *oblique trees*, we choose option (0) where linear splits are given a higher priority. GUIDE for regression uses only axis-aligned splits. We do not experiment with the kernel and nearest neighbor node models for classification and variants of linear regression used in regression trees, because they are beyond the scope of the conventional trees considered in this paper. The following hyperparameters were searched during cross validation: estimated or equal priors for class distribution, mean or median based cross validation, standard error for pruning, maximum number of splits and minimum node size. The latter two were the most important ones affecting the tree structure and accuracy, particularly for large datasets.

- **OC1:** We use the implementation provided by the authors[3] written in C. The default impurity measure used by their code is twoing criterion, and we only experiment with that option. For other impurity measures, one needs to change the code and recompile it again. We choose the option where only oblique hyperplanes are considered. During cross validation, we experiment with the number of random restarts and with the number of random jumps. For the most part, these two parameters affect the performance, but the results were not consistent. Moreover, this implementation only supports classification and thus we run OC1 on classification benchmarks.

- **TAO:** We implement both axis-aligned and oblique versions of the TAO.
  - *Oblique*: We use oblique (i.e. linear splits) decision trees with constant leaves. We take as an initial tree a deep enough, a complete binary tree with random parameters at each node. We use the fixed number of TAO iterations which is equal to 30, and algorithm proceeds until the maximum number of iterations are reached (i.e. there is no other stopping criterion). We also use a simple grid search on k-fold validation set to find the best hyperparameters. Specifically, we tune the "λ" parameter which controls sparsity of the tree and maximum depth of the initial tree. TAO algorithm is implemented in Python (version 3.5) without parallel processing in a single CPU. TAO uses an ℓ_1-regularized logistic regression to solve the decision node optimization (using LIBLINEAR (Fan et al., 2008)) where the mentioned "λ" parameter is used as an regularization parameter ($C = 1/\lambda$).
  - *Axis-aligned*: For the TAO axis-aligned trees, we use a decision tree with constant leaves (both for regression and classification). We initialized the tree with a pre-trained and pruned CART tree. Since TAO is implemented in Python 3.5, so we use scikit-learn implementation of CART as the initialization. We use a fixed number of TAO iterations which is equal to 30, and algorithm proceeds until the maximum number of iterations are reached or there is no more training error improvement up to a threshold of 1e-5. Unlike other algorithms, there are no hyper-parameters, as the tree is initialized with a pre-trained CART tree.

- **OCT** We report the results from the corresponding paper (for both axis aligned and oblique) since the implementation is not available online.

- **CO2** We report the results from the corresponding paper since the implementation is not available online.

We ran all experiments on a single Linux PC with the following specifications: OS - Ubuntu 18.04 LTS, CPU - 8 × Intel Core i7-7700 3.60GHz, Memory - 16 GiB DDR4 3600 MHz.

[2] http://pages.stat.wisc.edu/~loh/guide.html

[3] http://ccb.jhu.edu/software/oc1/oc1.tar.gz

	Dataset	N_{train}	N_{test}	D	K	Comments
classification	Iris	120	30	4	3	—
	Wine	142	36	13	3	—
	Dermatology	293	73	34	6	—
	Balance scale	500	125	4	3	—
	Breast Cancer	559	140	9	2	—
	Blood Trans	598	150	4	2	—
	German	800	200	20	2	categorical features encoded as one-hot
	Banknote auth	1098	274	4	2	—
	Contraceptive	1178	295	9	3	categorical features encoded as one-hot
	Car Eval	1382	346	6	4	categorical features encoded as one-hot
	Segment	1848	462	19	7	—
	Spambase	3681	920	57	2	—
	Optical recog	3823	1797	64	10	—
	Landsat	4435	2000	36	6	—
	Pendigits	7494	3498	16	10	—
	Letter[a]	16000	4000	16	26	—
	Connect4	54046	13511	126	3	—
	MNIST (pixels)[a]	60000	10000	784	10	grayscale image with pixels in [0,1]
	MNIST (LeNet5)[b]	60000	10000	800	10	output of LeNet5-conv2 on MNIST pixels
	SensIT[a]	78823	19705	100	3	—
regression	concrete	687	343	8	1	—
	airfoil	1002	501	5	1	—
	abalone	2506	1671	8	1	categorical features encoded as one-hot
	cpuact[c]	4915	3277	21	1	—
	ailerons[d]	7154	6596	40	1	—
	CT slice	42800	10700	384	1	—
	YearPredictionMSD	463715	51630	90	1	—

[a] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html

[b] https://faculty.ucmerced.edu/mcarreira-perpinan/teaching/CSE176/Labs/datasets/

[c] http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html

[d] https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

Table 1: Specs of the datasets used in our experiments. N is sample size, D is feature dimensionality and K is output dimension for regression and number of classes for classification. All datasets are from the UCI collection (Lichman, 2013) unless otherwise indicated by a footnote.

3.2 Datasets

Table 1 summarizes all datasets used in this study with their corresponding sources. All datasets are available in the public domain. The exception is "MNIST (LeNet5)". This dataset consists of features extracted by "conv2" layer of a pre-trained LeNet5 (LeCun et al., 1998) neural network for all MNIST images. Similar to MNIST, it also has 60000 training and 10000 test instances. Each instance has 800 non-negative real-valued attributes. For some of the datasets from UCI which do not have separate test set, we shuffle the entire dataset and keep 20% of the entire data as the test set. We repeat the training procedure 10 times for each dataset, reshuffling the training data each time.

4 Results

4.1 Classification

We summarize the results for classification datasets in Table 2-3. We report the train/test average accuracy (in %) and stdev over 10 repeats. Although we report the training error for reference, *the important error to consider is the test error.* Because the training error can be trivially made zero with a tree by simply growing it large enough that each leaf contains instances of the same class, at the cost of overfitting. Please note that some results in the tables are missing, because they were not provided in the original papers.

First, consider the case of an axis-aligned split. OCT shows a good performance only on relatively small datasets like Iris or Wine. We think that this is due to limited depth of the tree. Solving for deeper trees require a huge amount of time since OCT solves tree optimization problem exactly which is NP-hard, unless some early stopping criteria applied which does not guarantee an optimal solution. Therefore, we observe that OCT shows the worst performance in most cases. Next, both versions of the CART and C5.0 perform similarly in terms of test accuracy with a little favor to C5.0. GUIDE also performs similar to CART and C5.0. These methods share the same flavor of greedy recursive partitioning of the input space and thus it is expected that they have similar performance. Finally, TAO outperforms all algorithms in most datasets. In the very few datasets where TAO is not the winner, the difference with the winner is very small, and is due to a handful of instances, since these are small datasets in sample size (and dimension).

Second, consider the case of oblique decision trees. Here, we can observe similar behavior as with axis-aligned trees. TAO outperforms all other methods (including non-greedy approaches like CO2) in most dataset and often by considerable margin. The accuracy margin between TAO and the other methods becomes more as the dataset complexity grows like in MNIST. For instance, in the case of "MNIST (pixels)" and "MNIST (LeNet5)" not only the dataset size is big (60000 training data points) but also the number of attributes is very high. In the few cases where another algorithm produced a better tree, we could always further improve TAO by the following way: *TAO can take any given initial tree and improve its training loss or leave it unchanged* (unlike traditional top-down induction algorithms such as CART or C5.0, which build the tree from scratch). However, we do not use this advantage, and in all our experiments we initialize TAO oblique trees randomly.

Since decision trees are considered as interpretable models, it is important to also compare the size of the trained trees since if a decision tree is deep and has a large number of nodes then it becomes extremely difficult to interpret it. Moreover, the larger tree has large inference time and need more space. For this, in Table 4-5 we compare the average maximum depth and average number of leaves for both axis-aligned and oblique trees. Results makes sense: bigger datasets require larger trees (more depth and more leaves) and oblique trees generate more compact trees but with more complex node structure (i.e. generic hyperplane instead of axis-aligned split). The exception is OCT since authors have limited the maximum depth due to reasons described above. C5.0 usually generates larger trees compared to CART given that both perform similarly in terms of test accuracy. In general, decision trees obtained from TAO have comparable or smaller sizes as from the other methods. In some datasets (like Letter), the number of leaves and maximum depth are quite large for TAO axis aligned but it is due to large number of classes. More compact models can be obtained by using oblique tree.

4.2 Regression

We also perform comparison on regression datasets. Table 6 reports the results. All reported errors are rooted mean squared error (RMSE): $E = \sqrt{\frac{1}{NK} \sum_{n=1}^{N} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2}$ unless otherwise specified, where N is the sample size, K is the output dimension, and $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the ground truth and predicted vectors, respectively. Please, note that GUIDE for regression uses only axis-aligned splits. Results clearly demonstrate that TAO trees outperform all the other methods. Especially, TAO oblique shows superior accuracy and often by a large margin followed by TAO axis aligned. Moreover, the resulting tree size (see Table 7) is small in TAO oblique which makes it easier to interpret. In the few cases where TAO oblique tree performs worse than the axis-aligned tree, we believe this is due to the dataset size. Furthermore, we could always further improve TAO by initializing from axis-aligned tree as was described in the previous section.

Dataset		TAO	CART-R	CART-P	C5.0	GUIDE	OCT
Iris	train	97.03±0.65	97.08±1.19	97.02±0.99	99.03± 1.25	97.00±1.35	—
	test	95.41±3.81	93.33±3.15	94.75±3.20	92.64± 8.00	93.67±4.58	93.5
Wine	train	96.01±1.49	96.97±2.63	95.86±1.22	96.68± 1.58	96.76±2.28	—
	test	91.21±3.64	90.00±3.97	90.10±2.21	85.19±15.95	91.94±5.04	94.2
Dermatology	train	98.61±0.90	97.19±1.02	98.54±0.88	97.14± 0.72	97.33±0.68	—
	test	96.14±2.72	93.51±1.66	94.44±3.41	90.44± 4.78	95.54±3.08	89.2
Balance scale	train	85.95±1.28	85.94±0.42	82.66±0.92	88.38± 1.43	86.42±2.81	—
	test	82.21±3.36	78.96±0.34	79.62±2.09	78.19± 1.43	77.44±3.26	71.6
Breast Cancer	train	95.39±2.44	96.10±0.01	95.26±1.22	97.36± 0.61	96.62±0.79	—
	test	95.91±1.54	94.57±0.02	93.64±2.80	94.83± 0.90	94.57±1.75	91.5
Blood Trans	train	82.11±1.47	76.45±0.01	79.42±2.31	79.69± 0.59	80.90±1.97	—
	test	77.89±3.27	75.20±0.02	76.45±2.77	78.40± 2.45	78.80±3.26	77.0
Banknote auth	train	99.67±0.19	99.45±0.02	99.30±0.24	99.63± 0.12	99.38±0.08	—
	test	97.98±0.91	97.93±0.06	96.33±2.19	98.70± 0.68	98.25±1.69	90.7
Contraceptive	train	54.45±1.38	57.53±1.05	53.88±1.27	76.15± 1.41	58.79±1.89	—
	test	56.03±2.89	54.37±1.83	54.25±4.43	49.39± 4.79	55.19±2.13	53.3
Car Eval	train	99.94±0.12	98.36±1.41	99.91±0.12	92.71± 3.85	69.97±0.48	—
	test	96.67±2.67	96.35±1.90	96.51±2.69	87.59± 4.19	70.23±1.93	78.8
Segment	train	98.39±0.54	98.87±0.69	98.34±0.58	98.42± 0.88	98.31±0.44	—
	test	96.18±1.41	95.91±0.11	92.17±3.85	94.86± 1.69	95.37±1.11	—
Spambase	train	94.27±1.52	94.96±0.01	93.18±2.71	96.18± 0.38	95.36±0.85	—
	test	92.65±1.18	91.92±0.01	89.62±3.28	92.85± 0.84	92.77±0.64	86.1
Optical recog	train	98.48±0.06	95.33±2.18	97.85±0.12	96.76± 0.84	94.64±1.10	—
	test	86.08±0.22	84.26±1.03	85.19±0.11	80.33± 5.32	84.56±1.32	54.7
Landsat	train	96.97±0.26	91.13±0.49	97.37±0.12	95.19± 0.71	90.05±0.86	—
	test	86.10±0.25	85.94±0.27	85.21±0.23	84.12± 1.09	85.24±0.58	78.0
Pendigits	train	98.87±0.26	99.09±0.29	98.79±0.13	98.98± 0.09	97.90±0.50	—
	test	92.52±0.24	91.62±0.55	90.19±0.38	91.32± 0.71	89.93±0.77	—
Letter	train	96.38±0.03	94.30±0.01	95.93±0.11	97.97± 0.14	93.54±0.59	—
	test	86.39±0.12	86.04±0.04	86.07±0.14	85.26± 0.33	83.93±0.48	—
Connect4	train	86.99±1.74	82.94±1.08	83.63±0.13	82.60± 0.93	71.76±0.24	—
	test	79.88±1.53	78.29±0.21	77.55±1.18	77.84± 1.41	71.66±0.32	—
MNIST (pixels)	train	93.53±0.24	92.54±0.03	93.12±0.15	94.52± 0.23	75.76±0.40	—
	test	88.52±0.19	88.03±0.07	88.05±0.02	88.31± 0.35	78.52±0.20	—
MNIST (LeNet5)	train	96.94±0.05	95.71±0.04	96.64±0.08	97.89± 0.14	84.15±0.28	—
	test	93.52±0.03	93.31±0.05	93.32±0.07	93.48± 0.21	85.25±0.28	—
SensIT	train	83.56±0.12	84.38±0.01	82.82±0.23	86.66± 0.11	79.05±0.23	—
	test	81.84±1.11	81.71±0.01	81.00±0.19	81.41± 0.04	78.52±0.20	—
wins	train	6	3	1	9	1	0
wins	**test**	15	0	0	2	1	1

Table 2: Train and test accuracy (%, avgstdev over 10 repeats) for decision tree learning algorithms with axis-aligned splits. Method names are in section 3. Colors within each test row represents: Blue - the best performing method, Dark Red - the 2nd best performing method. Authors in OCT paper do not report training error or errorbars, only the average test error.

Dataset		TAO	OC1	GUIDE	OCT	CO2
Iris	train	96.89±9.05	85.42±15.84	98.42±0.58	—	—
	test	94.40±5.12	85.67±14.53	94.33±3.00	95.1	—
Wine	train	98.22±5.33	89.30±10.25	97.75±1.50	—	—
	test	92.00±9.38	84.45± 8.89	93.33±5.00	91.6	—
Dermatology	train	95.10±3.96	91.58± 6.10	98.60±0.50	—	—
	test	92.27±8.57	84.46± 7.79	97.84±1.73	92.6	—
Balance scale	train	91.68±0.72	93.22± 2.17	91.72±3.94	—	—
	test	88.48±2.56	88.96± 2.29	85.60±5.68	87.6	—
Breast Cancer	train	98.21±0.79	82.99±12.16	96.82±0.51	—	—
	test	97.71±1.04	81.07±12.75	95.64±1.61	94.0	—
Blood Trans	train	81.74±0.89	80.33± 2.69	81.02±0.63	—	—
	test	78.93±3.12	77.93± 3.72	79.87±3.33	77.4	—
German	train	82.90±0.71	78.44± 5.85	70.18±1.07	—	—
	test	81.24±0.87	68.65± 4.17	70.15±2.17	71.0	—
Banknote auth	train	99.83±0.33	94.12±12.91	99.63±0.27	—	—
	test	99.18±0.14	91.64±13.57	98.80±0.59	98.7	—
Contraceptive	train	66.04±7.24	61.44± 5.71	57.58±0.93	—	—
	test	57.47±3.18	49.66± 3.06	56.58±2.58	53.3	—
Car Eval	train	94.27±4.12	97.35± 2.92	69.97±0.48	—	—
	test	91.55±4.66	95.49± 2.32	70.23±1.93	87.5	—
Segment	train	99.48±0.21	91.61± 8.84	98.41±0.41	—	97
	test	96.48±1.31	88.53± 7.47	95.48±1.02	—	96
Spambase	train	95.55±0.47	80.72±16.51	95.74±0.99	—	—
	test	93.31±1.22	78.20±15.48	92.24±0.59	86.6	—
Optical recog	train	97.68±0.59	72.50±19.62	94.54±1.36	—	—
	test	91.27±1.74	62.00±17.60	79.19±1.20	54.3	—
Landsat	train	94.45±0.49	80.25± 2.20	91.54±1.29	—	—
	test	87.81±0.88	73.54± 2.00	85.97±0.80	78.2	—
Pendigits	train	99.81±0.13	91.72± 7.81	98.85±0.14	—	96
	test	96.80±0.70	84.42± 7.02	91.80±0.69	—	92
Connect4	train	82.40±0.53	79.02± 1.45	72.11±0.31	—	81
	test	81.09±0.39	75.42± 0.64	72.01±0.36	—	78
Letter	train	95.39±0.24	75.85± 3.80	90.80±1.05	—	94
	test	89.15±0.88	65.81± 4.83	82.65±0.90	—	87
MNIST (pixels)	train	98.43±0.07	78.62± 9.62	73.02±0.79	—	94
	test	94.74±0.11	74.34± 9.94	73.79±0.91	—	90
MNIST (LeNet5)	train	99.98±0.01	89.52±14.76	84.15±0.28	—	—
	test	98.22±0.18	87.97±14.24	85.25±0.28	—	—
SensIT	train	85.68±0.13	76.10±12.69	79.64±0.28	—	83
	test	85.12±0.20	73.70±11.31	79.25±0.33	—	82
wins	train	15	2	3	0	0
wins	**test**	14	2	3	1	0

Table 3: Similar to 2 but for decision trees with oblique splits.

	axis-aligned						oblique				
Dataset	TAO	CART-P	CART-R	C5.0	GUIDE	OCT	TAO	OC1	GUIDE	OCT	CO2
Iris	3.0	3.0	2.5	1.5	2.3	4.0	3.0	1.6	2.3	4.0	—
Wine	2.8	2.8	3.0	1.2	2.7	4.0	5.0	2.0	2.4	4.0	—
Dermatology	7.0	7.0	6.1	4.7	6.7	4.0	7.0	3.0	5.1	4.0	—
Balance scale	7.2	7.2	6.7	7.1	6.3	4.0	3.0	2.9	5.8	4.0	—
Breast Cancer	3.4	3.4	3.2	4.0	4.0	4.0	3.0	2.5	1.8	4.0	—
Blood Trans	7.4	7.4	0.0	2.5	4.8	4.0	5.0	3.9	2.4	4.0	—
Banknote auth	6.0	6.0	5.8	5.8	6.7	4.0	3.0	4.2	5.1	4.0	—
Contraceptive	4.6	4.6	4.3	11.1	5.6	4.0	5.0	5.1	4.0	4.0	—
Car Eval	12.7	12.2	11.8	3.6	4.6	4.0	4.0	3.5	3.6	4.0	—
Segment	14.0	14.0	13.8	7.3	13.2	—	8.0	5.3	15.2	—	8.0
Spambase	14.4	14.4	10.7	14.7	12.3	4.0	4.0	3.56	15.4	4.0	—
Optical recog	12.0	12.0	11.7	10.8	13.4	4.0	7.0	4.2	13.7	4.0	—
Landsat	12.0	12.0	11.8	11.1	9.9	4.0	7.0	6.3	17.0	4.0	—
Pendigits	15.2	15.2	13.8	13.6	14.0	—	8.0	5.9	20.1	—	12.0
Letter	27.0	27.0	26.0	17.0	23.4	—	11.0	10.2	28.0	—	12.0
Connect4	33.2	33.2	27.7	16.8	16.7	—	8.0	8.6	17.9	—	16.0
MNIST (pixels)	19.0	19.0	18.3	19.0	8.5	—	8.0	5.0	14.9	—	14.0
MNIST (LeNet5)	17.0	17.0	18.6	15.2	9.3	—	8.0	4.4	9.3	—	—
SensIT	12.0	12.0	14.0	15.2	9.8	—	7.0	8.1	10.8	—	6.0

Table 4: Average maximum depths over 10 repetitions for both axis-aligned and oblique classification trees.

	axis-aligned					oblique		
Dataset	TAO	CART	CART	C5.0	GUIDE	TAO	OC1	GUIDE
Iris	4.4	4.4	3.6	3.5	3.3	8.1	2.6	3.3
Wine	5.6	5.8	5.5	3.5	4.6	16.0	3.2	3.5
Dermatology	9.6	9.6	7.2	6.7	8.6	64.0	5.0	6.2
Balance scale	24.6	24.6	22.6	27.8	18.1	5.6	4.3	8.3
Breast Cancer	5.4	5.6	5.5	9.0	7.6	7.8	4.0	3.2
Blood Trans	20.0	20.8	1.0	4.6	8.2	10.8	5.7	3.5
Banknote auth	19.2	20.2	14.0	14.3	19.0	7.4	9.3	8.3
Contraceptive	7.2	7.4	7.6	89.6	12.7	24.4	9.3	7.3
Car Eval	68.0	68.0	56.7	41.2	6.7	14.3	5.6	5.0
Segment	38.2	39.2	41.3	21.0	39.7	135.0	11.0	31.2
Spambase	53.6	55.0	41.7	68.6	53.5	14.8	5.1	48.0
Optical recog	193.8	198.2	107.2	72.6	126.6	57.4	9.6	138.7
Landsat	231.4	236.4	57.6	67.1	50.8	70.6	13.4	50.5
Pendigits	177.2	183.6	153.6	129.0	161.5	146.0	19.4	135.5
Letter	1550.8	1579.6	920.8	1343.0	994.8	1077.6	88.7	673.1
Connect4	5336.0	5743.8	1212.6	813.0	38.4	210.0	32.8	26.5
MNIST (pixels)	899.5	899.5	805.4	941.6	58.8	177.8	12.8	38.5
MNIST (LeNet5)	484.0	484.0	363.2	582.0	42.4	166.8	11.8	42.4
SensIT	152.0	152.0	239.5	410.0	41.6	69.2	21.8	24.3

Table 5: Average number of leaves over 10 repetitions for both axis-aligned and oblique classification trees.

		oblique	axis-aligned			
Dataset		TAO	TAO	CART-R	CART-P	GUIDE
concrete	train	3.91 ±0.11	3.06 ±5.36	3.93 ±2.67	3.07 ±2.31	5.46 ± 0.37
	test	7.41 ±0.12	7.20 ±3.17	7.23 ±3.08	7.22 ±3.13	7.50 ± 0.27
airfoil	train	3.01 ±0.29	0.47 ±0.10	0.72 ±0.12	0.52 ±0.10	2.44 ± 0.10
	test	3.13 ±0.38	2.73 ±0.62	2.77 ±0.86	2.75 ±0.62	3.20 ± 0.19
abalone	train	2.11 ±0.02	2.29 ±0.12	2.31 ±0.25	2.32 ±0.11	2.21 ± 0.05
	test	2.18 ±0.05	2.32 ±0.58	2.38 ±0.31	2.34 ±0.59	2.30 ± 0.09
cpuact	train	2.47 ±0.07	2.68 ±0.69	2.91 ±0.71	2.71 ±0.65	10.00 ± 0.88
	test	2.71 ±0.04	3.26 ±0.51	3.36 ±1.32	3.28 ±0.44	10.99 ± 1.54
ailerons	train	1.65 ±0.02	2.39 ±0.00	1.81 ±0.12	2.83 ±0.23	1.86 ± 0.02
	test	1.76 ±0.02	2.55 ±0.00	2.21 ±0.63	2.85 ±0.57	2.06 ± 0.02
CT slice	train	1.42 ±0.04	1.01 ±0.04	1.12 ±0.15	1.06 ±0.06	8.12 ± 0.17
	test	1.54 ±0.05	2.66 ±0.04	2.91 ±0.95	2.69 ±0.03	8.23 ± 0.20
YearPredictionMSD	train	8.91 ±0.03	9.71 ±0.31	9.73 ±0.20	9.71 ±0.24	9.78 ± 0.01
	test	9.11 ±0.05	9.76 ±0.11	9.81 ±0.31	9.79 ±0.54	9.83 ± 0.01
wins	train	4	3	0	0	0
wins	**test**	5	2	0	0	0

Table 6: Train and test rooted mean squared error (RMSE) for different methods on regression datasets. Results for ailerons scaled to $E \times 10^{-4}$.

4.3 Runtime

It is commonly accepted that tree induction algorithms are fast. Especially, those which use greedy growing strategy like CART, C5.0, etc. We did not apply any parallel processing in our experiments and we observe the following:

- For small UCI datasets (like Balance Scale, Breast Cancer, etc.), all axis-aligned and oblique trees that we ran are quite fast (around 0.5-1.5 seconds).
- For relatively larger datasets (like MNIST, SensIT, etc.), we observe some fluctuations. In general, axis-aligned trees are still fast to train: for instance, CART and C5.0 took about 200-400 seconds to train on MNIST (pixels), whereas GUIDE and TAO axis-aligned took a little longer than that (about 1300-1500 seconds). As for oblique trees: GUIDE took extremely long time (about 26000 seconds for MNIST)and OC1 performs comparatively better (about 1800 seconds). Finally, TAO oblique decision tree optimization is much faster compare to other oblique tree training algorithms. It took about 1200-1400 seconds to train on MNIST.

5 Discussion

In this work, we demonstrate the performance comparison of some well-known decision tree algorithms along with the recently proposed TAO algorithm. We evaluate their performance on classification and regression tasks which are the main tasks in machine learning. Our experiments show that TAO not only performs better in accuracy but also provides smaller and more interpretable decision trees. The reason for such performance is the way how TAO optimization works. Traditional algorithms greedily optimize the decision trees: at each step they split the data by using a single or collection of attributes that optimize some criterion (usually impurity). This approach has no guarantees towards the reduction of the desired loss (e.g. misclassification loss). Thus, in the end, the obtained trees usually do not generalize well and also are big in size. On the other hand, TAO instead of optimizing the impurity of a node at each step, optimizes the desired objective function over a decision tree of given structure and finds much better approximate optima than CART-type algorithms. This is done by employing alternating optimization over the nodes of a tree. This approach leads to better-optimized decision trees that not only generalize well but also have a smaller

	oblique		axis-aligned							
	TAO		TAO		CART-R		CART-P		GUIDE	
Dataset	Δ	L	Δ	L	Δ	L	Δ	L	Δ	L
concrete	8.0	169.5	11.2	113.0	11.8	104.4	11.2	113.0	9.0	59.9
airfoil	8.0	147.1	15.0	479.8	15.6	457.2	15.0	479.8	10.6	95.3
abalone	6.0	58.6	5.0	12.8	4.8	11.0	5.0	12.8	6.0	18.1
cpuact	6.0	52.7	9.0	57.2	8.7	52.8	9.0	57.2	8.4	21.8
ailerons	6.0	60.2	7.0	15.0	7.8	66.6	7.0	15.0	8.5	66.1
CT slice	7.0	74.8	36.0	700.0	30.0	691.6	36.0	700.0	12.7	83.2
YearPredictionMSD	8.0	157.9	12.0	135.0	11.8	121.0	12.0	135.0	10.3	111.9

Table 7: Average depths (Δ) and average number of leaves (L) over 10 repetitions for regression datasets. TAO produces more compact and shallower trees which are easier to interpret.

size. Empirical results presented in this paper show that TAO outperforms other non-greedy approaches as well (e.g. CO2, soft trees, etc.)

References

D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, July 2017.

L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, Calif., 1984.

M. Á. Carreira-Perpiñán and P. Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 1211–1221. MIT Press, Cambridge, MA, 2018.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, Aug. 2008.

L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

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

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

W.-Y. Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12 (2):361–386, Apr. 2002.

W.-Y. Loh. Improving the precision of classification trees. *Annals of Applied Statistics*, 3(4):1710–1737, 2009.

W.-Y. Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348 (with discussion, pp. 349–370), Dec. 2014.

S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: A randomized algorithm for building oblique decision trees. In *Proc. of the 11th National Conference on Artificial Intelligence (AAAI 1993)*, pages 322–327, Washington, DC, July 11–15 1993.

M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 1720–1728. MIT Press, Cambridge, MA, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, Oct. 2011. Available online at https://scikit-learn.org.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

T. Therneau, B. Atkinson, and B. Ripley. rpart: Recursive partitioning and regression trees. R package version 4.1-15, Apr. 12 2019. Available online at https://cran.r-project.org/package=rpart.