Pushing the Envelope of Gradient Boosting Forests via Globally-Optimized Oblique Trees

Magzhan Gabidolla Miguel Á. Carreira-Perpiñán
Dept. CSE, University of California, Merced
{mgabidolla, mcarreira-perpinan}@ucmerced.edu

Abstract

Ensemble methods based on decision trees, such as Random Forests or boosted forests, have long been established as some of the most powerful, off-the-shelf machine learning models, and have been widely used in computer vision and other areas. In recent years, a specific form of boosting, gradient boosting (GB), has gained prominence. This is partly because of highly optimized implementations such as XGBoost or LightGBM, which incorporate many clever modifications and heuristics. However, one gaping hole remains unexplored in GB: the construction of individual trees. To date, all successful GB versions use axis-aligned trees trained in a suboptimal way via greedy recursive partitioning. We address this gap by using a more powerful type of trees (having hyperplane splits) and an algorithm that can optimize, globally over all the tree parameters, the objective function that GB dictates. We show, in several benchmarks of image and other data types, that GB forests of these stronger, well-optimized trees consistently exceed the test accuracy of axis-aligned forests from XGBoost, LightGBM and other strong baselines. Further, this happens using many fewer trees and sometimes even fewer parameters overall.

The last 30 years have established ensemble methods as some of the most accurate models for classification, regression and other ML tasks. The most successful of these are based on decision trees, ensembled using bagging (such as Random Forests [5]) or boosting. Many types of boosting exist, such as AdaBoost and its variations [33], but one that has gained much attention in recent years is gradient boosting (GB). This is partly due to GB's performance, empirically demonstrated in many works, and to the development of extremely efficient implementations such as XGBoost [12], LightGBM [23] or CatBoost [28]. These toolkits, continuously refined by a large team of academic and industrial researchers and developers, provide a convenient user interface and train forests of thousands of trees very efficiently—not an easy task given that GB requires trees to be constructed sequentially. The success of these forests is evident in their widespread use in applications in computer vision and other areas, and their many victories in data science challenges such as those organized by Kaggle.

Another advantage of forests is that a user can typically get a highly accurate model with little hyperparameter exploration, usually just the number of trees, tree size (depth or number of leaves) and learning rate (weight of each tree). This is far simpler than selecting the architecture and SGD hyperparameters of a deep net, hence Random Forests and GB forests are often considered "off-the-shelf" models. In reality, they do depend on a larger set of hyperparameters (many of them controlling the training of the individual trees), but it is fair to say that these have secondary importance. Finally, GB forests are also much faster to train than neural nets. On the negative side, forests can be very big, using thousands of trees or more and comprising many parameters.

Many variations of GB forests have been researched, which has resulted in the very powerful GB toolkits mentioned above. These involve modifications to the GB ensembling mechanism (e.g. choice of loss function) and the tree optimization (e.g. choice of purity function). Many of them are algorithmically heuristic or guided by system-level optimizations; this makes XGBoost very different from LightGBM, for example. Given this, how can we further improve GB forests? We observe that all popular GB forest variants are consistently based on the same type of trees (axis-aligned, where each decision node tests a single input feature), and on the same type of training algorithm (greedy recursive partitioning as in CART [6], C4.5 [30] or variations of it [12, 23]). On the one hand, such trees are very unstable learners, and this introduces diversity in the ensemble, which is necessary for its success. On the other hand, axis-aligned trees impose restrictive modeling assumptions and have low individual accuracy. Yet, while more complex types of trees (such as oblique trees) have been proposed for use in forests, they have not been able to compete robustly with axis-aligned trees in either accuracy or model size. Are axis-aligned forests then the best option?

We convincingly show that oblique trees, *when properly trained to optimize the loss that GB dictates*, do result in more accurate forests and use very few, shallow trees. The key to this is twofold. First, in section 2 we argue that axis-aligned forests have an intrinsically restricted model ability when many-feature interactions are present in the data (often the case in computer vision), and that oblique forests improve this. Second, in section 4 we show that optimizing the GB loss over an oblique tree—a very difficult problem so far unresolved—can be effectively achieved using a variation of the *tree alternating optimization (TAO)* algorithm [9, 10]. Finally, in section 5 we demonstrate over a range of classification and regression datasets that our resulting oblique forests achieve better accuracy than any competing GB forest. As additional advantages, the resulting forests use fewer, shallower trees, often having fewer parameters overall; and many of the heuristics involved in training axis-aligned trees (such as controlling for the split or pruning criterion, the minimum number of instances per node, etc.) become unnecessary.

1. Related work

The literature on boosted decision trees is huge [18, 33]. We review some of the notable works on GB and then on decision tree learning.

Friedman [15] introduced the GB framework as a generic mechanism to learn greedily an additive model for any differentiable loss function. Given the flexibility of its formulation, it applies to problems in regression, binary and multiclass classification [15], ranking [8], and density estimation [32], [7]. While the original GB formulation [15] uses only the functional gradient information, one can also incorporate second order information, and derive LogitBoost [14] for classification. The use of this functional Newton step seems to have prevailed, as all the modern popular toolkits implement it. One can also incorporate various forms of regularization into the GB forest construction such as penalizing a tree structure or the value of a leaf prediction [20]. Empirically, this appears to help, and these forms of regularization terms can be found as some (among the many) of the hyperparameters of the modern GB software. Finally, XGBoost [12], LightGBM [23] and CatBoost [28] are some of the popular, quite recent and highly optimized implementations of GB forests. At a more fundamental algorithmic level, they implement the same core GB procedure, but they differ in specific practical implementation details such as usage of histograms, clever subsampling, special handling of categorical features, etc.

The de facto choice to optimize individual base learners in GB are axis-aligned decision trees with a greedy algorithm. This widely established paradigm of learning trees dates back to the 1980s and its core idea is based on the process of greedy top-down induction: one starts with a root node, and continues to split the nodes until a predefined stopping criteria is met. Axis-aligned splits evaluate every feature-threshold pair, and chooses the one that optimizes some splitting criteria. The exact form of this splitting criteria can be a proxy of the objective function (e.g. Gini index or cross entropy [18]), though in the GB setting, the tree objective function is usually directly used to find the optimal split. Once the tree is fully grown, an optional pruning step based on some cost function can be performed, but this step is rarely implemented in the GB framework. Traditional algorithms on decision tree learning, such as CART [6], C5.0 [30], ID3 [29], and base learner trees in XGBoost and others, all fall onto this paradigm of tree learning.

Oblique trees, which use a linear combination of features at a decision node, are a much more powerful class of models, but they have found little practical use. This is due to the fact that learning such models is more difficult [18]. Works based on greedy recursive partitioning for oblique trees in isolation [6,26] or in a bagged forest [21,37] produce little improvement and considerably increase the model size. Axis-aligned trees with linear models at the leaves have been used in GB [35], but such trees are still learned greedily. GB regression forests have been used for a face alignment problem [22], where the decision nodes threshold the difference of two pixels; this is a limited form of oblique forest where the location of these two pixels is chosen suboptimally.

Our work is based on TAO [9, 10], a recent algorithm for learning oblique decision trees, which has been shown to exceed the state-of-the-art both in single trees [41] and in forests using bagging [11, 38] or AdaBoost [39, 40]. In this paper, we adapt TAO to the GB setting. We describe TAO in section 4.

2. Modeling high-order feature interactions: axis-aligned vs oblique trees and forests

Our fundamental hypothesis is that 1) forests of oblique trees are intrinsically more powerful than forests of axis-aligned trees, and 2) realizing this requires being able to do a good optimization of the GB loss over each individual tree. Here we give intuitive evidence for this and demonstrate it in a specially constructed experiment.

High-order feature interactions A critical advantage that oblique trees offer over axis-aligned trees is their ability to model high-order feature interactions effectively. For simplicity, here we consider binary trees where each leaf i outputs a constant value $\theta_i \in \mathbb{R}$. Consider an axis-aligned tree of depth Δ and call $\Delta_i \leq \Delta$ the depth of leaf i. The tree predictive function for an input instance $\mathbf{x} \in \mathbb{R}^D$ can be written as a linear combination (l.c.) of basis functions (BFs) $\tau(\mathbf{x}) = \sum_{i \in \text{leaves}} \theta_i \lambda_i(\mathbf{x})$, where the BF $\lambda_i(\mathbf{x})$ is 1 if $\mathbf{x}$ reaches leaf i and 0 otherwise ("routing function" of

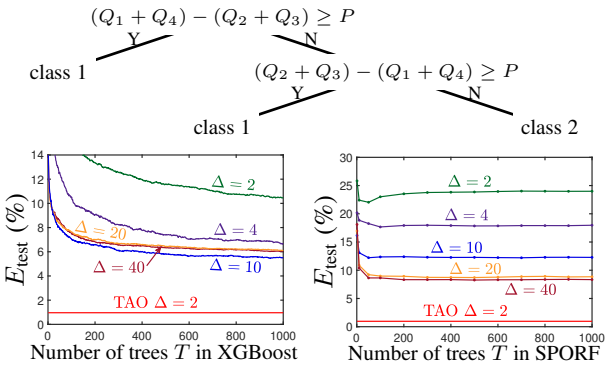

Figure 1. Synthetic MNIST classification problem. *Top*: exact oblique tree. *Bottom*: results of forests of different depth Δ and number of trees T with axis-aligned XGBoost trees (left) and oblique SPORF trees (right). The result of a single, depth-2 oblique tree trained with TAO is shown in both plots.

leaf i). Each $\lambda_i(\mathbf{x})$ involves the AND of Δ_i conditions of the form "$x_d \geq s_d$" (i.e., a split on feature d with threshold s_d). Hence, λ_i is a function of at most Δ_i features (it can be fewer if some features appear more than once in the root-leaf path). From this point of view, an axis-aligned tree is a generalized additive model (GAM) [17] with member functions of up to Δ variables. In contrast, an oblique tree uses *all* features in *each* decision node, so *each* λ_i is an AND of Δ conditions of the form "$\mathbf{w}_j^T\mathbf{x} \geq b_j$", where $(\mathbf{w}_j, b_j)$ determine a hyperplane split at node j. Hence, from a representation point of view, axis-aligned trees can include interactions of up to Δ features, while oblique trees include interactions of all D features[1]. When the number of features D is large, an axis-aligned tree will have to be very deep to be able to handle high-order interactions. However, this would result in leaves receiving very few training instances and hence in severe overfitting (this is a reason why trees are usually pruned in CART, C4.5 and other tree induction algorithms). Conversely, if the tree is not deep (e.g. XGBoost's default maximum depth is 6), then it will miss many possibly important higher-order interactions. It will also impose a drastic feature selection (per leaf i): it will use at most Δ_i features of the total D. The clearly superior accuracy of oblique trees (trained with TAO) over axis-aligned trees has been demonstrated empirically [10, 41].

Consider now a forest of T axis-aligned trees of depth Δ. Its predictive function is $F(\mathbf{x}) = \sum_{t=1}^{T} \eta_t \tau_t(\mathbf{x})$, i.e., a l.c. of the individual tree predictive functions, and hence a l.c. of the $\{\lambda_{ti}(\mathbf{x})\}$ BFs. Even though each BF is limited to at most order-Δ interactions, some higher-order interactions arise in the forest (contrarily to the statement in [18, p. 362]). For example, a function $f(x_1, x_2) + g(x_2, x_3)$ is an order-3 interaction (not order-2), because of the shared variable x_2 (while a function $f(x_1, x_2) + g(x_3, x_4)$ remains at order 2). However, this is still quite a restricted type of order-3 interaction. In contrast, with oblique trees each $\lambda_{ti}(\mathbf{x})$ BF is of order D and their addition is a much more complex order-D interaction. This suggests that, if we are modeling a complex classification or regression function, a forest of oblique trees should achieve higher accuracy and require fewer and shallower trees. Our experiments in section 5 convincingly confirm this with various real-world datasets, but here we demonstrate it with a carefully constructed synthetic (but not unrealistic) example. We consider a binary classification of MNIST digit images where class 1 satisfies that $(Q_1+Q_4)-(Q_2+Q_3) \geq P$ or $(Q_2+Q_3)-(Q_1+Q_4) \geq P$, where the 28×28 pixel image is split into 4 quadrants $\begin{smallmatrix}1&2\\3&4\end{smallmatrix}$ and Q_i is the sum of the $[0,1]$ pixel intensities in quadrant i, and $P = 30$. That is, class 1 is an "(anti)diagonally dominant" image like ◪ or ◩, and it contains 28% of the training instances. Class 2 contains the 72% remaining instances. Obviously, the true classifier involves interactions between all $D = 784$ features (pixels), as is common with image data, and can be exactly represented by a depth-2 oblique tree (see fig. 1). Indeed, a single depth-2 tree trained with TAO achieves a near-perfect training/test error of 0.0%/0.96% (the test error is nonzero because the training set does not uniquely determine the tree in fig. 1). However, an XGBoost forest does much worse even using 1000 trees of depth up to 40 (fig. 1 left). This is in stark contrast to the suggestion in [18, p. 363] that trees with more than 10 leaves would be rarely required in boosting.

A good oblique tree optimization is essential To hold in practice, the theoretically stronger power of oblique trees requires an effective learning algorithm. While the desire to learn oblique trees has long existed [6, p. 132, 173], until recently the algorithms proposed resulted in oblique trees that were more complex and with more parameters than axis-aligned trees, and yet were typically no more accurate [3, p. 233]. As a result, all widespread forest models use axis-aligned trees [2,5,12,16,23,27,28]. This is primarily due to the difficulty of the optimization: trees define a nondifferentiable function, so gradient methods are not applicable, and the tree learning problem is NP-hard [19]. Besides, while for axis-aligned trees each individual split can be learned by enumeration (over all features and thresholds), this is not possible with oblique trees. Fig. 1 (right) illustrates this: a recent algorithm to train oblique forests (SPORF [37]) performs far worse than the XGBoost axis-aligned trees, whether for a single tree or a forest, and whether in accuracy or number of parameters.

In summary, we conclude that well-optimized oblique trees and forests provide a much more effective way to learn high-order feature interactions. Our experiments in section 5 confirm that this translates to real-world datasets consistently and often by a large margin.

[1]The form of the interaction function is also restricted by the functional form of $\lambda_i(\mathbf{x})$ and by the number of leaves (and, in a forest, by the number of trees and by how different leaf regions intersect). However, our focus is on the maximum interaction order that can be represented.

3. Overview of Gradient Boosting (GB)

The goal of the GB framework is to learn a model of the following additive form:

$$\mathbf{F}(\mathbf{x}) = \sum_{m=1}^{M} \boldsymbol{\tau}_m(\mathbf{x}) \quad (1)$$

where $\boldsymbol{\tau}(\mathbf{x})$ can in general be any base learner, but in this work we focus on decision trees. Given a training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N \subset \mathbb{R}^D \times \mathbb{R}^K$ and a per instance loss function $L(\mathbf{y}, \hat{\mathbf{y}})$, the global objective function is:

$$\min_{\boldsymbol{\tau}_1,\dots,\boldsymbol{\tau}_M} \sum_{n=1}^{N} L\big(\mathbf{y}_n, \sum_{m=1}^{M} \boldsymbol{\tau}_m(\mathbf{x}_n)\big). \quad (2)$$

Because this is a difficult optimization problem with non-differentiable base learners $\boldsymbol{\tau}(\mathbf{x})$, one resorts to greedy forward stagewise modeling. Starting with some initial prediction $\mathbf{F}_0(\mathbf{x})$, at each step m we want to add a new tree to minimize the global loss:

$$\min_{\boldsymbol{\tau}_m} \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{F}_{m-1}(\mathbf{x}_n) + \boldsymbol{\tau}_m(\mathbf{x}_n)) \quad (3)$$

Following the established approach in [12, 14], we perform a second order Taylor approximation:

$$\min_{\boldsymbol{\tau}_m} \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{F}_{m-1}(\mathbf{x}_n)) + \mathbf{g}_n^T \hat{\mathbf{y}}_n + \tfrac{1}{2} \hat{\mathbf{y}}_n^T \mathbf{H}_n \hat{\mathbf{y}}_n \quad (4)$$

where $\hat{\mathbf{y}}_n = \boldsymbol{\tau}_m(\mathbf{x}_n)$, and $\mathbf{g} \in \mathbb{R}^K, \mathbf{H} \in \mathbb{R}^{K \times K}$ are the gradient and Hessian of the loss function $L(\mathbf{y}, \hat{\mathbf{y}})$ (see fig. 2). Ignoring the constant term in eq. (4), we obtain the following objective function of a base learner at boosting step m:

$$\min_{\boldsymbol{\tau}_m} \sum_{n=1}^{N} \mathbf{g}_n^T \boldsymbol{\tau}_m(\mathbf{x}_n) + \tfrac{1}{2} \boldsymbol{\tau}_m(\mathbf{x}_n)^T \mathbf{H}_n \boldsymbol{\tau}_m(\mathbf{x}_n) \quad (5)$$

Though this is an unusual loss function and might look complicated to optimize, in practice a full Hessian matrix $\mathbf{H}$ is seldom used, and in regression problems with squared error loss $L(\mathbf{y}, \hat{\mathbf{y}})$, the Hessian $\mathbf{H}$ vanishes. Once a base learner $\boldsymbol{\tau}_m(\mathbf{x})$ is trained at boosting step m, its contribution is shrunk by the learning rate η and then added to the ensemble. This shrinkage step has been found to be essential for the generalization performance of boosting. The algorithm then proceeds for some number of predefined boosting steps M or one might early stop based on the validation metric. The pseudocode in fig. 2 summarizes the GB algorithm.

input training set; twice diff. loss function $L(\mathbf{y}, \hat{\mathbf{y}})$; number of boosting steps M; learning rate η
$\mathbf{F}_0(\mathbf{x}) = \arg\min_{\boldsymbol{\rho}} \sum_{n=1}^{N} L(\mathbf{y}_n, \boldsymbol{\rho})$
for $m = 1$ **to** M
  $\mathbf{g}_n = \frac{\partial L(\mathbf{y}_n, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}\Big|_{\hat{\mathbf{y}}=\mathbf{F}_{m-1}(\mathbf{x}_n)}, n = 1, \dots, N$
  $\mathbf{H}_n = \frac{\partial^2 L(\mathbf{y}_n, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}^2}\Big|_{\hat{\mathbf{y}}=\mathbf{F}_{m-1}(\mathbf{x}_n)}, n = 1, \dots, N$
  Train a base learner $\boldsymbol{\tau}_m$ to minimize eq. (5)
  $\mathbf{F}_m(\mathbf{x}_n) = \mathbf{F}_{m-1}(\mathbf{x}_n) + \eta \boldsymbol{\tau}_m(\mathbf{x}_n), n = 1, \dots, N$
return $\mathbf{F}(\cdot) = \mathbf{F}_0(\cdot) + \sum_{m=1}^{M} \eta \boldsymbol{\tau}_m(\cdot)$

Figure 2. Pseudocode of Gradient Boosting (GB).

4. Learning each GB oblique tree using tree alternating optimization

As just described, GB prescribes a specific, if rather unusual, loss to optimize for each new tree that is added to the forest, eq. (5). To do this with oblique trees, we build on a recent algorithm, tree alternating optimization (TAO) [9, 10], originally proposed to optimize the 0/1 loss for oblique classification trees. We choose TAO because it can be extended to handle the GB loss while preserving the advantages of the original 0/1 loss TAO algorithm: globally updating all parameters of an oblique tree while monotonically decreasing the objective function; automatically learning the tree structure as a subproduct of an ℓ_1 penalty on the decision node weights; and scaling to large trees and datasets. We are not aware of any other algorithm that can do all that. We give a brief description of TAO (see more details in [9, 10]) and indicate how we can modify it to suit our needs.

The basic idea in TAO is that we optimize a parametric tree of fixed structure (here, complete of depth Δ) by optimizing in turn over the parameters of a single node (decision node or leaf) given all other nodes' parameters are fixed. This succeeds because of two theorems mentioned below. This is very different from how CART [6] or C4.5 [30] work. The latter grow the tree structure on the fly by greedily and recursively partitioning the input space, and pruning the resulting tree to reduce overfitting. TAO works much more like a regular ML optimization algorithm, say for a neural net, but instead of gradients (which do not apply) it uses alternating optimization on a fixed tree structure. This results in iteratively updating all the parameters in the tree (decision node hyperplanes and leaf output values), with a monotonic decrease of the objective function at each iteration over all nodes and convergence to a local optimum.

Unlike in [10], 1) we consider an oblique tree where each leaf i outputs a constant real vector, since this is what GB requires for either classification or regression; and 2) we seek to optimize the following objective function:

$$\min_{\boldsymbol{\Theta}} \sum_{n=1}^{N} l(\mathbf{g}_n, \mathbf{H}_n, \boldsymbol{\tau}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha \sum_{i \in \mathcal{D}} \|\mathbf{w}_i\|_1$$
$$\text{where} \qquad l(\mathbf{g}, \mathbf{H}, \boldsymbol{\gamma}) = \mathbf{g}^T \boldsymbol{\gamma} + \tfrac{1}{2} \boldsymbol{\gamma}^T \mathbf{H} \boldsymbol{\gamma}. \quad (6)$$

Here, $\boldsymbol{\tau}(\mathbf{x}; \boldsymbol{\Theta})$ is a binary decision tree of some predetermined structure with parameters $\boldsymbol{\Theta} = \{(\mathbf{w}_i, w_{i0})\}_{i \in \mathcal{D}} \cup \{\boldsymbol{\theta}_i\}_{i \in \mathcal{L}}$, decision nodes in set $\mathcal{D}$ and leaves in set $\mathcal{L}$. The prediction of $\boldsymbol{\tau}(\mathbf{x}; \boldsymbol{\Theta})$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and outputting its vector $\boldsymbol{\theta}_i \in \mathbb{R}^K$ (more generally, this could be a, say, linear predictor, but here we focus on a constant output). At a decision node i we apply a decision function $f_i(\mathbf{x}; \mathbf{w}_i, w_{i0})$: $\mathbb{R}^D \rightarrow \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{D} \cup \mathcal{L}$ denoting "go to the right child if $\mathbf{w}_i^T \mathbf{x} + w_{i0} \geq 0$, else go to the left child". The objective function (6) is the sum of the GB loss (5) plus an ℓ_1 penalty

on each of the decision node weights with hyperparameter $\alpha \geq 0$. The latter encourages sparsifying the hyperplanes. This reduces the number of nonzero weights and can make decision nodes redundant (when $\mathbf{w}_i = \mathbf{0}$) so they can be pruned at the end. Thus, we automatically learn a tree structure that is a subset of the initial tree. The hyperparameters of the oblique tree are its (maximum) depth Δ and α.

TAO is based on two theorems. First, eq. (6) *separates over any subset of non-descendant nodes* (e.g. all the nodes at the same depth); this follows from the fact that the tree makes hard decisions. All such nodes may be optimized in parallel. Second, optimizing over the parameters of a single node i simplifies to a well-defined *reduced problem* over the instances that currently reach node i (the *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$). The form of the reduced problem depends on the type of node:

Decision node It is a *weighted 0/1 loss binary classification problem*, where the two classes correspond to the left and right child, which are the only possible outcomes for an instance. Child $\texttt{left}_i$ ($\texttt{right}_i$) incurs a loss (weight) given by the prediction of the leaf reached from the left (right) child's subtree. Thus, each instance is assigned as *pseudolabel* the child with lower loss. The reduced problem takes the form (where $\overline{L}$ is the said loss):

$$\min_{\mathbf{w}_i, w_{i0}} \sum_{n \in \mathcal{R}_i} \bar{L}(\mathbf{g}_n, \mathbf{H}_n, f_i(\mathbf{x}; \mathbf{w}_i, w_{i0})) + \alpha \left\|\mathbf{w}_i\right\|_1. \quad (7)$$

This is as in [10] except that the loss is the GB loss of the corresponding leaf. This problem is NP-hard but can be well approximated with a convex surrogate; we use ℓ_1-regularized logistic regression where each instance is weighted by the loss difference between the winner child and the other child, and solve it using LIBLINEAR [13]. We can guarantee a monotonic decrease in the objective by only accepting this update if it improves over the previous step.

Leaf The reduced problem consists of optimizing the original loss but over the leaf classifier on its reduced set:

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} \mathbf{g}_n^T \boldsymbol{\theta}_i + \frac{1}{2} \boldsymbol{\theta}_i^T \mathbf{H}_n \boldsymbol{\theta}_i. \quad (8)$$

If $\sum_{n \in \mathcal{R}_i} \mathbf{H}_n$ is positive definite, the exact solution is $\boldsymbol{\theta}_i = -\left(\sum_{n \in \mathcal{R}_i} \mathbf{H}_n\right)^{-1} \sum_{n \in \mathcal{R}_i} \mathbf{g}_n$. In practice either θ_i is scalar (e.g. binary classification) or one uses a diagonal approximation to the Hessian.

Given an initial tree structure with initial parameter values, the resulting algorithm repeatedly visits nodes in reverse breadth-first search order. Each iteration trains all nodes at the same depth (in parallel) from the leaves to the root, by solving either an ℓ_1-regularized logistic regression at each decision node, or the above exact solution as each leaf.

Computational complexity With a diagonal Hessian, the training time is dominated by the decision node reduced problem (logistic regression). Assuming this is linear on the sample size, training all the decision nodes at the same depth is approximately constant and equal to training one logistic regression on the whole training set. Thus, the total *sequential* cost of one iteration is approximately equal to that of Δ logistic regressions on the whole dataset. As noted above, all the nodes at the same depth can be trained *in parallel*.

5. Experiments

Experiment settings We compare GB oblique trees trained with TAO against established state-of-the-art implementations of GB: XGBoost [12] and LightGBM [23]. For reference, we also compare with a scikit-learn [27] implementation of GB, which closely follows the original formulation [15]. We could not find any existing implementation of GB oblique trees, but we include SPORF [37] which uses bagging to construct an ensemble of sparse oblique trees. We additionally cite the published results of other forest based methods: GBDT-PL [35], ADF [34], sNDF [24] and rRF [31].

In order to compare models of different size, for each comparison baseline we choose and fix some number of boosting steps M (or number of trees T in SPORF), and for the given M or T we tune the important hyperparameters such as tree depth Δ, number of leaves and the learning rate η. For GB-TAO we only tune the tree depth Δ. We find the optimal ℓ_1 penalty parameter α for a single TAO tree, and then use it for the whole GB ensemble. Because of the slower training time, we do not tune the learning rate η in GB-TAO. Unless otherwise stated, the number of TAO iterations is set to $I = 30$. Extended results including diversity of the trees and comparison with more methods can be found in the suppl. mat.

5.1. Classification tasks

Image classification Table 1 reports the results on standard image classification datasets. For CIFAR100 we extract the convolutional features of a pretrained VGG16 network, and use it as input for forests.

Let us first observe the result of a single TAO tree obtained from one GB step M=1. While axis-aligned trees produced from a single GB step in general perform quite poorly, this is not a case with properly optimized oblique trees. For MNIST the accuracy of a single TAO tree matches the performance of 100 XGBoost trees, and for pendigits and CIFAR100 it achieves considerably better performance than any other axis-aligned GB forest. This result clearly supports the discussion in section 2 and demonstrates the superiority of an oblique tree over an axis-aligned one for these image datasets.

	Forest	E_{test} (%)	#pars.	M	k	Δ	#leav.
MNIST (60k,784,10)	XGBoost	4.38±0.00	70k	10	10	10	237
	GB-TAO	4.17±0.08	21k	1	1	12	203
	LightGBM	3.73±0.00	149k	10	10	35	498
	sNDF [24]	2.80±0.12	-	80	1	10	-
	SPORF	2.89±0.04	-	1k	1	50	5k
	ADF [34]	2.71±0.10	-	100	1	25	-
	GB-TAO	2.33±0.00	200k	10	1	10	209
	XGBoost	2.20±0.00	107k	100	10	6	36
	rRF [31]	2.05±0.02	-	100	1	25	-
	LightGBM	2.02±0.00	121k	100	10	10	41
	GB-sklearn	1.96±0.03	1.2M	100	10	6	42
	GB-TAO	1.94±0.00	671k	30	1	10	252
	XGBoost	1.91±0.00	505k	1k	10	6	17
	GB-TAO	1.65±0.02	3M	50	10	7	37
	LightGBM	1.62±0.00	642k	1k	10	21	22
	GB-TAO	1.55±0.02	7.2M	140	10	7	34
pendigits (7.5K,16,10)	GB-sklearn	4.19±0.01	169k	100	10	6	57
	SPORF	4.00±0.02	-	10	1	19	332
	LightGBM	3.49±0.00	90k	100	10	11	31
	XGBoost	3.46±0.00	18k	100	10	4	7
	XGBoost	3.46±0.00	137k	1k	10	4	5
	LightGBM	3.31±0.00	895k	1k	10	4	31
	GB-TAO	3.15±0.25	1.3k	1	1	8	60
	SPORF	2.91±0.09	-	1k	1	20	330
	SPORF	2.87±0.01	-	100	1	20	212
	GB-TAO	2.17±0.02	13k	10	1	7	65
	GB-TAO	2.00±0.04	44k	30	1	7	83
CIFAR100 (50k,512,100)	GB-sklearn	32.64±0.03	502k	100	100	6	17
	XGBoost	31.20±0.00	133k	100	100	4	5
	LightGBM	31.47±0.00	460k	100	100	14	16
	LightGBM	30.32±0.00	1.0M	143	100	18	25
	XGBoost	30.15±0.00	174k	1k	100	6	1.2
	GB-TAO	29.38±0.04	39k	1	1	12	178
	SPORF	28.63±0.07	-	10	1	20	262
	SPORF	27.07±0.20	-	100	1	10	107
	GB-TAO	26.98±0.04	1.2M	150	5	8	33
	GB-TAO	26.86±0.02	2.0M	250	5	8	33
	SPORF	26.71±0.05	-	500	1	10	107
	GB-TAO	26.64±0.02	3.3M	200	2	6	46

Table 1. Comparison of different forest-based models for classification, sorted by decreasing test error. We report 0-1 test error E_{test} (mean±std over 5 repeats), the number of parameters in the model, and the average number of tree leaves in the forest. Dataset name is followed by the training set size$N$, feature dimension D and the number of classes K. M refers to the number of boosting steps, k is the number of trees used at each boosting step. The total number of trees $T = Mk$. Δ is the max depth of the forest.

Now observing the performance of the overall ensemble of TAO trees for these classification problems, we can clearly see that oblique trees do indeed help to produce more accurate GB forests. The gap between GB TAO and the best performing axis-aligned GB forest is quite significant, and it is quite unlikely that by tuning many of the hyperparameters in XGBoost and LightGBM packages for longer one can cover such a significant gap. Another forest of oblique trees, SPORF, whose trees are induced greedily, produces quite accurate performance on CIFAR100, but for others, especially for MNIST, the results are quite worse.

Sparse high dimensional datasets A particular well-documented weakness of traditional GB trees is in handling inputs with sparse high dimensional features [25, 36]. To analyze how GB oblique trees perform with these types of problems, in Table 2 we compare its performance on two standard document classification benchmarks. The features are normalized word counts, and on average only 0.1-0.2% of them are nonzero. Consistent with the results on image classification, a single sparse oblique tree trained with TAO already matches the accuracy of 100 axis-aligned XGBoost trees, and with more GB steps the performance improves significantly. A notable result is the best performing GB-TAO on News20 dataset, which is 3.2% more accurate than the best performing LightGBM. The use of hyperplane splits might be attributable for the effectiveness of GB TAO for these high dimensional problems, but, SPORF forests, which also use oblique splits, produce much higher test error. This advocates for the value of the optimization performed by TAO in learning oblique trees as was discussed in section 2.

	Forest	E_{test} (%)	#pars.	M	k	Δ	#leav.
news20 (16k,62k,20)	**GB-TAO**	27.75±0.03	19k	1	1	6	61
	GB-sklearn	23.42±0.03	156k	100	20	6	27
	SPORF	22.51±0.09	(1.3M)	100	1	569	4.4k
	GB-sklearn	21.71±0.02	347k	300	20	6	20
	XGBoost	21.39±0.00	705k	1k	20	6	12
	XGBoost	21.34±0.00	188k	300	20	6	11
	LightGBM	20.69±0.00	1.8M	1k	20	27	31
	GB-TAO	19.84±0.02	534k	30	1	6	61
	LightGBM	19.78±0.00	546k	300	20	28	31
	GB-TAO	18.13±0.01	479k	20	20	4	8
	GB-TAO	18.76±0.01	746k	50	1	6	52
	GB-TAO	16.65±0.04	1.6M	40	20	4	8
real-sim (51k,21k,2)	GB-sklearn	5.65±0.05	150k	100	1	14	502
	GB-sklearn	4.25±0.02	422k	1k	1	14	141
	SPORF	4.14±0.05	(1.3M)	100	1	687	4.3k
	SPORF	4.06±0.03	(3.9M)	300	1	690	4.3k
	GB-TAO	3.44±0.24	18k	1	1	6	42
	XGBoost	3.41±0.00	23k	300	1	10	26
	LightGBM	3.31±0.00	27k	300	1	29	31
	XGBoost	3.31±0.00	101k	1k	1	14	34
	GB-TAO	3.12±0.00	53k	5	1	4	15
	LightGBM	3.05±0.00	101k	1k	1	30	31
	GB-TAO	2.77±0.02	113k	10	1	4	15
	GB-TAO	2.12±0.02	1.3M	20	1	6	54

Table 2. Similar to Table 1, but for sparse high-dimensional document classification datasets

5.2. Regression tasks

Table 3 reports the results on regression benchmarks. With squared error loss, a tree produced from the first GB step with no shrinkage step ($\eta = 1$) is just equivalent to a tree trained with that same original loss. Unlike classification, however, we do not observe a consistent match in performance between a single TAO oblique tree and hundreds of axis-aligned GB trees. Regression problems are in general difficult to model for piecewise constant models, and especially for a single tree. However, for one particular dataset, Computer Tomography (CT) slices, the accuracy of a single TAO tree of depth 8 is on par with 1000 axis-aligned XGBoost trees with depth up to 10, which certainly demonstrate a case for the importance of modeling higher order variable interaction in certain regression problems.

	Forest	E_{test}	#pars.	T	Δ	#leav.
cpuact (5k,21)	**GB-TAO**	2.67±0.03	440	1	6	34
	XGBoost	2.60±0.00	60k	100	10	201
	XGBoost	2.51±0.00	42k	1k	4	15
	GB-sklearn	2.51±0.06	14k	100	6	49
	GB-sklearn	2.41±0.04	43k	1k	4	15
	GB-TAO	2.42±0.02	17k	30	6	46
	LightGBM	2.27±0.00	19k	100	34	64
	LightGBM	2.25±0.00	91k	1k	21	31
	GB-TAO	2.23±0.02	31k	50	6	48
CT-slice (43k,384)	LightGBM	1.53±0.00	153k	100	107	512
	LightGBM	1.52±0.00	91k	1k	23	31
	GB-sklearn	1.43±0.02	192k	100	10	641
	XGBoost	1.50±0.00	107k	100	10	357
	GB-TAO	1.28±0.02	28k	1	8	223
	GB-sklearn	1.26±0.03	900k	1k	10	301
	XGBoost	1.26±0.00	767k	1k	10	256
	GBDT-PL [35]	1.24±0.00	-	-	-	-
	GB-TAO	0.90±0.02	81k	30	4	16
	GB-TAO	0.45±0.01	1.2M	100	6	64
casp (45k,9)	**GB-TAO**	4.38±0.03	4k	1	12	430
	XGBoost	3.66±0.00	119k	100	10	397
	GB-sklearn	3.65±0.02	727k	100	14	2424
	XGBoost	3.58±0.00	793k	1k	10	265
	GB-sklearn	3.58±0.01	854k	1k	10	285
	LightGBM	3.54±0.00	153k	100	114	512
	GB-TAO	3.49±0.01	256k	50	12	645
	LightGBM	3.48±0.00	766k	1k	109	256
	GBDT-PL [35]	3.46±0.00	-	-	-	-
	GB-TAO	3.43±0.00	481k	100	12	603
	GB-TAO	3.39±0.01	887k	200	12	552
superconduct (17k,81)	**GB-TAO**	11.02±0.10	8k	1	19	466
	GB-sklearn	9.38±0.01	139k	1k	6	47
	XGBoost	9.20±0.00	130k	1k	6	44
	GB-sklearn	9.14±0.03	128k	100	10	430
	XGBoost	8.98±0.00	132k	100	10	441
	GBDT-PL [35]	8.80±0.00	-	-	-	-
	LightGBM	8.77±0.00	38k	100	45	128
	GB-TAO	8.76±0.02	573k	50	6	216
	LightGBM	8.73±0.00	190k	1k	39	64
	GB-TAO	8.68±0.02	1M	100	6	218
year (450k,90)	**GB-TAO**	9.17±0.01	19k	1	8	252
	XGBoost	9.05±0.00	153k	100	10	511
	LightGBM	9.03±0.00	153k	100	37	512
	GB-sklearn	9.03±0.02	248k	100	10	827
	GB-sklearn	8.96±0.02	171k	1k	6	58
	LightGBM	8.92±0.00	1.5M	1k	43	512
	XGBoost	8.91±0.00	1.8M	1k	10	608
	GB-TAO	8.88±0.02	78k	20	6	62
	GB-TAO	8.73±0.01	402k	100	6	63

Table 3. Similar to Table 1, but for regression datasets. E_{test} is a root mean squared error. The output in all datasets is one dimensional, so one tree is used at each boosting step.

Examining the overall performance of forests in Table 3, we can observe a consistent improvement (often by a large margin) of GB TAO trees over other methods. LightGBM tend to perform as well on couple of datasets, but it usually produces deep and imbalanced trees. In one competing method, GBDT-PL [35], the trees use linear models at the leaves, but are still axis-aligned and greedily induced. Though linear leaves can model variables of higher order interaction, it is still important optimize such trees properly. We believe that oblique trees with linear leaf models optimized by TAO can further boost the performance of GB forests.

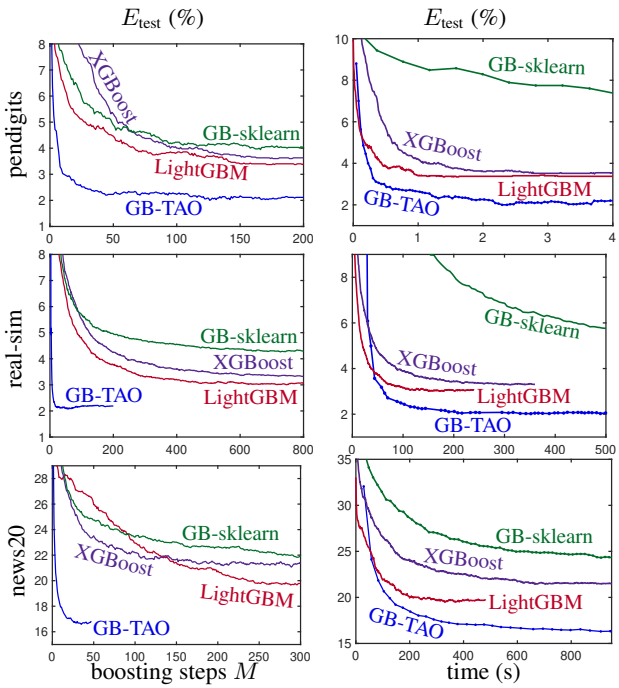

Figure 3. Comparison of different GB forests as a function of the number of boosting steps M (left column) and time (right column). All methods except GB-sklearn are trained using parallel processing on a shared memory system with 8 threads.

5.3. Training time and the number of boosting steps

In analogy with numerical optimization, steps of GB have been motivated as following functional gradient (or Newton) steps, where the steps are represented by parametric models, such as decision trees. Further expanding this analogy, one would expect that if these step directions are more accurate representations of the true gradient (or Newton), then it should lead to faster convergence and possibly, to a better optima for a given number of boosting steps M. In the left column of fig. 3 we compare how the test error changes for classification datasets as a function GB steps M. The plots clearly show that stronger TAO oblique trees require fewer steps M to converge, and to converge to a better test error than greedily induced axis-aligned trees.

One notable disadvantage of TAO over greedy tree induction algorithms is slower training time. TAO performs an iterative optimization over the nodes, and at each iteration it must solve multiple logistic regressions. However, it is possible to accelerate GB TAO in multiple ways. At the algorithmic level, we do not usually need to train GB-TAO for many steps M as is commonly done with traditional axis-aligned trees. Also, we can apply approximations, such as solving the reduced problem at a decision node inexactly. In the right column of fig. 3, we show that by limiting the number of TAO iterations to I=3, we can obtain accurate GB-TAO forests in a comparable time. In suppl. mat. we further explore the interplay between runtime, TAO iterations I and test error.

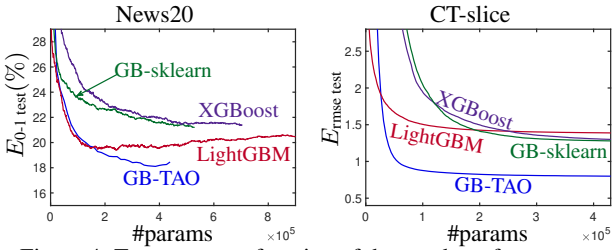

Figure 4. Test error as a function of the number of parameters

5.4. Test error vs model size

Forests of oblique trees can be potentially huge in terms of the number of parameters. For a given number of trees T of depth Δ and feature dimension D, the number of parameters can be up to $TD2^{\Delta}$. This might especially be problematic with high dimensional data. Luckily, ℓ_1 penalty used by TAO helps to produce forests of comparable number of parameters. Fig. 4 shows that, for the selected datasets, GB-TAO consistently obtains more accurate forests than GB with axis-aligned trees for a given number of parameters. In suppl. mat. we explore how the number of trees at each boosting step affects the model size and accuracy.

6. Discussion and limitations of our work

Off-the-shelf models Our GB oblique forests can be considered off-the-shelf models even more so than axis-aligned forests. This is because, once the optimization of an individual tree is well defined, many heuristics become unnecessary. For example, besides `max_depth`, `learning_rate` and the number of trees, XGBoost includes hyperparameters such as `max_delta_step`, `min_split_loss`, `min_child_weight`, the `lambda` and `alpha` regularization hyperparameters, etc. (refer to the XGBoost manual for details). These control whether to split or prune a node based on the number of training instances it receives, the gain in split purity, an ℓ_2 penalty in the leaves, which thresholds to check in the splits, etc. In TAO, the tree structure and parameters automatically result from optimizing a well-defined objective function.

Runtime In terms of training runtime, our oblique forests are quite slower than XGBoost or LightGBM at present, but our implementation is far from optimized compared to these heavily developed toolkits. Training a single oblique tree of depth Δ with TAO has a cost per iteration comparable to training Δ logistic regressions on the entire dataset (for which efficient code such as LIBLINEAR exists). While this is much slower than training a single axis-aligned tree with CART, the total number of trees is also much smaller in the oblique forests. All things considered, the oblique forests will probably be slower than the axis-aligned ones, but the gain in accuracy and possibly smaller model size compensates for that.

In terms of inference time, the oblique forests may be faster than axis-aligned ones in that 1) they may make a better use of GPUs because they use vector products, and 2) they have a more regular memory access pattern because the trees are shallower. The number of trees is also much smaller, while still providing ample room for parallelization. At present it is hard to do an apples-to-apples comparison because of the lack of an optimized implementation.

Limitations We have argued and empirically confirmed that oblique forests are a less restrictive model for feature interactions than axis-aligned forests. That said, oblique forests still take a particular functional form (inductive bias), and there may be even better types of trees. Also, our results hold within the GB framework, although we expect them to apply to some extent to other frameworks.

Although we have provided a reasonable intuition behind the success of oblique GB forests, we have no rigorous theory to explain it. The theoretical basis of (gradient) boosting itself remains hotly debated. One leading theory [14] that seems consistent with our results states that AdaBoost.M1 performs forward stagewise additive modeling using the exponential loss. This suggests that, if base learners at each stage are well optimized (as we do with TAO), this should result in a faster minimization of the exponential loss.

In any case, the accuracy and forest size performance is in the end an empirical question. While our paper is necessarily limited to a small set of specific experiments, the strong baselines we compare with on several popular benchmarks and the consistent improvement shown by our GB forests gives us confidence that our results hold robustly.

7. Conclusion

While GB forests are among the most accurate ML models for classification and regression, until now all widespread implementations of them use axis-aligned trees, which are a restricted type of base learner. We have motivated the use of a significantly more powerful tree type having hyperplane splits, which are able to learn many-feature interactions effectively. Key to this is the ability to optimize the GB loss over such trees, a difficult problem which we address using a variation of tree alternating optimization. Far from decreasing the ensemble diversity, the resulting oblique forests considerably improve over axis-aligned forests, even those of highly sophisticated implementations such as XGBoost or LightGBM. In raw accuracy, the oblique forests consistently improve over all competitors, sometimes by a surprisingly large margin, using few, shallow trees, often having fewer parameters overall. Given the widespread use of GB forests in computer vision and other areas, this could have a considerable practical impact. Our work also suggests that exploring other types of trees or loss functions, properly optimized, may result in even better GB forests.

Acknowledgments. Work supported by NSF award IIS–2007147.

References

[1] *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021. 10

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Łukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. TensorFlow WhitePaper. 3

[3] Ethem Alpaydın. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, third edition, 2014. 3

[4] S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors. *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31. MIT Press, Cambridge, MA, 2018. 9, 10

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001. 1, 3

[6] Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984. 1, 2, 3, 4

[7] Peter Bühlmann and Torsten Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, 22(4):477–505 (with discussion, pp. 506–522), 2007. 2

[8] Christopher J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR–TR–2010–82, Microsoft Research, 2010. 2

[9] Miguel Á. Carreira-Perpiñán. The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models. arXiv, 2021. 2, 4

[10] Miguel Á. Carreira-Perpiñán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In Bengio et al. [4], pages 1211–1221. 2, 3, 4, 5

[11] Miguel Á. Carreira-Perpiñán and Arman 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. 2

[12] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 785–794, San Francisco, CA, Aug. 13–17 2016. 1, 2, 3, 4, 5

[13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, Aug. 2008. 5

[14] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–407, Apr. 2000. 2, 4, 8

[15] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. 2, 5

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr. 2006. 3

[17] Trevor J. Hastie and Robert J. Tibshirani. *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1990. 3

[18] Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009. 2, 3

[19] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976. 3

[20] Rie Johnson and Tong Zhang. Learning nonlinear functions using regularized greedy forest. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 36(5):942–954, May 2013. 2

[21] Rakesh Katuwal, P. N. Suganthan, and Le Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078, Mar. 2020. 2

[22] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'14)*, pages 1867–1874, Columbus, OH, June 23–28 2014. 2

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 3146–3154. MIT Press, Cambridge, MA, 2017. 1, 2, 3, 5

[24] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Buló. Deep neural decision forests. In *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*, pages 1467–1475, Santiago, Chile, Dec. 11–18 2015. 5, 6

[25] Pan Li, Zhen Qin, Xuanhui Wang, and Donald Metzler. Combining decision trees and neural networks for learning-to-rank in personal search. In *Proc. of the 25rd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2019)*, pages 2032–2040, Anchorage, AK, USA, Aug. 4–8 2019. 6

[26] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *J. Artificial Intelligence Research*, 2:1–32, 1994. 2

[27] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learn-*

ing Research, 12:2825–2830, Oct. 2011. Available online at https://scikit-learn.org. 3, 5

[28] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: Unbiased boosting with categorical features. In Bengio et al. [4], pages 6638–6648. 1, 2, 3

[29] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. 2

[30] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. 1, 2, 4

[31] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Global refinement of random forest. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 723–730, Boston, MA, June 7–12 2015. 5, 6

[32] Greg Ridgeway. Looking for lumps: Boosting and bagging for density estimation. *Computational Statistics and Data Analysis*, 38(4):379–392, Feb. 28 2002. 2

[33] Robert E. Schapire and Yoav Freund. *Boosting. Foundations and Algorithms*. Adaptive Computation and Machine Learning Series. MIT Press, 2012. 1, 2

[34] Samuel Schulter, Paul Wohlhart, Christian Leistner, Amir Saffari, Peter M. Roth, and Horst Bischof. Alternating decision forests. In *Proc. of the 2013 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'13)*, pages 508–515, Portland, OR, June 23–28 2013. 5, 6

[35] Yu Shi, Jian Li, and Zhize Li. Gradient boosting with piecewise linear regression trees. In *Proc. of the 28th Int. Joint Conf. Artificial Intelligence (IJCAI'19)*, pages 3432–3438, Macao, China, Aug. 10–16 2019. 2, 5, 7

[36] Si Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In Doina Precup and Yee Whye Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 3182–3190, Sydney, Australia, Aug. 6–11 2017. 6

[37] Tyler M. Tomita, James Browne, Cencheng Shen, Jaewon Chung, Jesse L. Patsolic, Benjamin Falk, Carey E. Priebe, Jason Yim, Randal Burns, Mauro Maggioni, and Joshua T. Vogelstein. Sparse projection oblique randomer forests. *J. Machine Learning Research*, 21:1–39, 2020. 2, 3, 5

[38] Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In Hal Daumé III and Aarti Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 11398–11408, Online, July 13–18 2020. 2

[39] Arman Zharmagambetov, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán. Improved boosted regression forests through non-greedy tree optimization. In *Int. J. Conf. Neural Networks (IJCNN'21)* [1]. 2

[40] Arman Zharmagambetov, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán. Improved multiclass AdaBoost for image classification: The role of tree optimization. In *IEEE Int. Conf. Image Processing (ICIP 2021)*, pages 424–428, Online, Sept. 19–22 2021. 2

[41] Arman Zharmagambetov, Suryabhan Singh Hada, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán. Non-greedy algorithms for decision tree optimization: An experimental comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)* [1]. 2, 3