UNIVERSITY OF CALIFORNIA, MERCED

The Role of Alternating Optimization in Learning Decision Tree Ensembles

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering & Computer Science

by

Magzhan Gabidolla

Committee in charge:

Professor Miguel Á. Carreira-Perpiñán, Chair
Professor Alberto Cerpa
Professor Xiaoyi Lu

2025

The dissertation of Magzhan Gabidolla is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

(Professor Alberto Cerpa)

(Professor Xiaoyi Lu)

(Professor Miguel Á. Carreira-Perpiñán, Chair)

University of California, Merced

2025

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisor M. Á. Carreira-Perpiñán for his great teaching, expertise, guidance and help throughout my PhD years. He has been a role model for the value of hard work, persistence and critical thinking. Special thanks to my committee members, Alberto Cerpa and Xiaoyi Lu, for their feedback and assistance.

I am deeply grateful to my undergraduate professors – Martin Lukac, Ben Tyler, Mona Rizvi, M. Fatih Demirci and Kenneth Nordstrom – for their excellent teaching and inspiring me to pursue research. Special thanks to my high school math teacher, Baibolov Zhanat Seitkazyuly, for teaching me the value of hard work and the importance of math from early years.

I would like to thank all my labmates, Yerlan, Suryabhan, Arman, Rasul and Kuat, for productive discussions and collaboration. Special acknowledgment to Arman and his family in helping us settle in Merced. I want to thank my dear friends, Arman, Daryn, Nurzhan, Alisher and Bakhtiyar for their support and sharing important moments of my life.

Most importantly, I would like to thank my parents Gabidolla and Makpal, my sister Tana, and my brother Olzhas for raising me and for the happiest childhood that I had and for always supporting me at every stage of my life. Lastly, I would like to express my gratitude to my dear wife, Assem for her unconditional love and care, and to our daughter, Meruert for bringing joy and meaning to our life.

VITA

2019	Bachelor of Science in Computer Science with *Summa Cum Laude*, Nazarbayev University, Astana, Kazakhstan
2025	Ph.D. in Electrical Engineering and Computer Science, University of California, Merced, CA, USA

PUBLICATIONS

M. Gabidolla and M. Á. Carreira-Perpiñán. A new class of interpretable models: the GAM tree. *in submission*, 2025.

M. Gabidolla and M. Á. Carreira-Perpiñán. Generalized additive models via direct optimization of regularized decision stump forests. *International Conf. on Machine Learning (ICML)*, 2025.

M. Gabidolla, A. Zharmagambetov and M. Á. Carreira-Perpiñán. Beyond the ROC curve: classification trees using cost-optimal curves, with application to imbalanced datasets. *International Conf. on Machine Learning (ICML)*, 2024.

M. Gabidolla and M. Á. Carreira-Perpiñán. Optimal interpretable clustering using oblique decision trees. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

M. Gabidolla and M. Á. Carreira-Perpiñán. Pushing the envelope of gradient boosting forests via globally-optimized oblique trees. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

M. Gabidolla, A. Zharmagambetov and M. Á. Carreira-Perpiñán. Improved multiclass AdaBoost using sparse oblique decision trees. *IEEE International Joint Conf. on Neural Networks (IJCNN)*, 2022.

M. Gabidolla, A. Zharmagambetov and M. Á. Carreira-Perpiñán. Cost-sensitive learning of classification trees, with application to imbalanced datasets. *Bay Area Machine Learning Symposium (BayLearn)*, 2023.

M. Gabidolla, A. Zharmagambetov and M. Á. Carreira-Perpiñán. Boosted Sparse Oblique Decision Trees. *Bay Area Machine Learning Symposium (BayLearn)*, 2020.

M. Á. Carreira-Perpiñán, M. Gabidolla and A. Zharmagambetov. Towards better decision forests: Forest Alternating Optimization. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

R. Kairgeldin, M. Gabidolla and M. Á. Carreira-Perpiñán. Adaptive softmax trees for many-class classification. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2024.

A. Zharmagambetov, M. Gabidolla and M. Á. Carreira-Perpiñán. Softmax tree: an accurate, fast classifier when the number of classes is large. *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

A. Zharmagambetov, M. Gabidolla and M. Á. Carreira-Perpiñán. Improved boosted regression forests through non-greedy tree optimization. *IEEE International Joint Conf. on Neural Networks (IJCNN)*, 2021.

A. Zharmagambetov, S. S. Hada, M. Gabidolla and M. Á. Carreira-Perpiñán. Non-greedy algorithms for decision tree optimization: an experimental comparison. *IEEE International Joint Conf. on Neural Networks (IJCNN)*, 2021.

A. Zharmagambetov and M. Gabidolla and M. Á. Carreira-Perpiñán. Improved multiclass AdaBoost for image classification: the role of tree optimization. *IEEE International Conf. on Image Processing (ICIP)*, 2021.

E. Chan, M. Gabidolla and M. Á. Carreira-Perpiñán. Oblique decision trees as an image model for cubist image restyling. *IEEE International Conf. on Image Processing (ICIP)*, 2025.

E. Chan, M. Gabidolla and M. Á. Carreira-Perpiñán. Cubist-style image effects with oblique decision trees. *Bay Area Machine Learning Symposium (BayLearn)*, 2024.

J. P. S. Sundaram, M. Gabidolla, M. Á. Carreira-Perpiñán and A. Cerpa. COMNETS: COst-sensitive decision trees approach to throughput optimization for Multi-radio IoT NETworkS. *IEEE International Conf. on Emerging Technologies and Factory Automation*, 2025.

J. P. S. Sundaram, A. Zharmagambetov, M. Gabidolla, M. Á. Carreira-Perpiñán and A. Cerpa. MARS: Multi-radio Architecture with Radio Selection using decision trees for emerging mesoscale CPS/IoT applications. *IEEE International Conference on Emerging Technologies and Factory Automation*, 2025.

Y. Idelbayev, A. Zharmagambetov, M. Gabidolla and M. Á. Carreira-Perpiñán. Faster neural net inference via forests of sparse oblique decision trees. *unpublished manuscript*, 2021.

ABSTRACT OF THE DISSERTATION

The Role of Alternating Optimization in Learning Decision Tree Ensembles

by

Magzhan Gabidolla
Doctor of Philosophy in Electrical Engineering & Computer Science
University of California Merced, 2025
Professor Miguel Á. Carreira-Perpiñán, Chair

Decision forests, also known as tree ensembles, have become an important model class in machine learning, particularly for tabular data, due to their excellent predictive accuracy, robustness, and minimal requirements for hyperparameter tuning. Remarkably, their widespread success has been achieved despite traditional training methods being largely heuristic, neither the individual trees nor the forest as a whole explicitly optimize a clearly defined loss function. Typically, these ensembles rely on methods such as bagging and boosting, where base learners are generated through greedy recursive partitioning.

This dissertation proposes an alternative, optimization-driven framework for learning decision forests. Building upon the recently introduced Tree Alternating Optimization (TAO) algorithm, we first show significant improvements when explicitly optimized base learners replace the standard, heuristically constructed trees within two of the most popular boosting algorithms: AdaBoost and Gradient Boosting. Taking this idea further, we define an objective function over a fixed-size decision forest and extend alternating optimization to optimize the ensemble jointly, eliminating the greedy incremental approach of traditional boosting. The resulting method, Forest Alternating Optimization (FAO), is very effective at minimizing training error. Though they are more susceptible to overfitting, averaging multiple FAO forests achieves best generalization performance.

Finally, we show the application of FAO to the special case of ensembles of decision stumps, which can naturally be viewed as generalized additive models (GAMs). By introducing targeted regularization strategies specifically designed for stump forests, we effectively mitigate overfitting, while consistently achieving state-of-the-art GAM performance.

Chapter 1

Introduction

Over the past two decades, decision tree ensembles, also known as decision forests, have consistently ranked among the most accurate machine learning models for tasks such as regression, classification, and other predictive problems, especially when working with tabular data. Their effectiveness is reflected in their widespread adoption across real-world applications, including fraud detection and ranking systems, as well as their frequent top rankings in machine learning competitions and surveys. For instance, the 2021 Kaggle survey on the state of machine learning reported that nearly 75% of respondents used decision trees and random forests, making them the second most popular technique after linear models, while gradient boosting machines followed at around 60% [66]. Further evidence of the superiority of tree-based methods on tabular data comes from several recent studies that explore why they often outperform deep neural networks in these contexts [119, 51, 90], despite the dominance of deep learning in fields such as computer vision and natural language processing. A key practical advantage of decision forests is their minimal need for extensive hyperparameter tuning; unlike deep neural networks, which require careful design choices regarding architecture, size, optimizer, learning rates, etc., tree ensembles often deliver strong performance with minimal tuning, making them reliable, off-the-shelf solutions.

To illustrate the concept of a decision forest and provide intuition for its strong modeling capacity, Fig. 1.1 presents a 2D regression example: predicting grayscale intensities of the *Cameraman* image from pixel coordinates (x_1, x_2). A decision for-

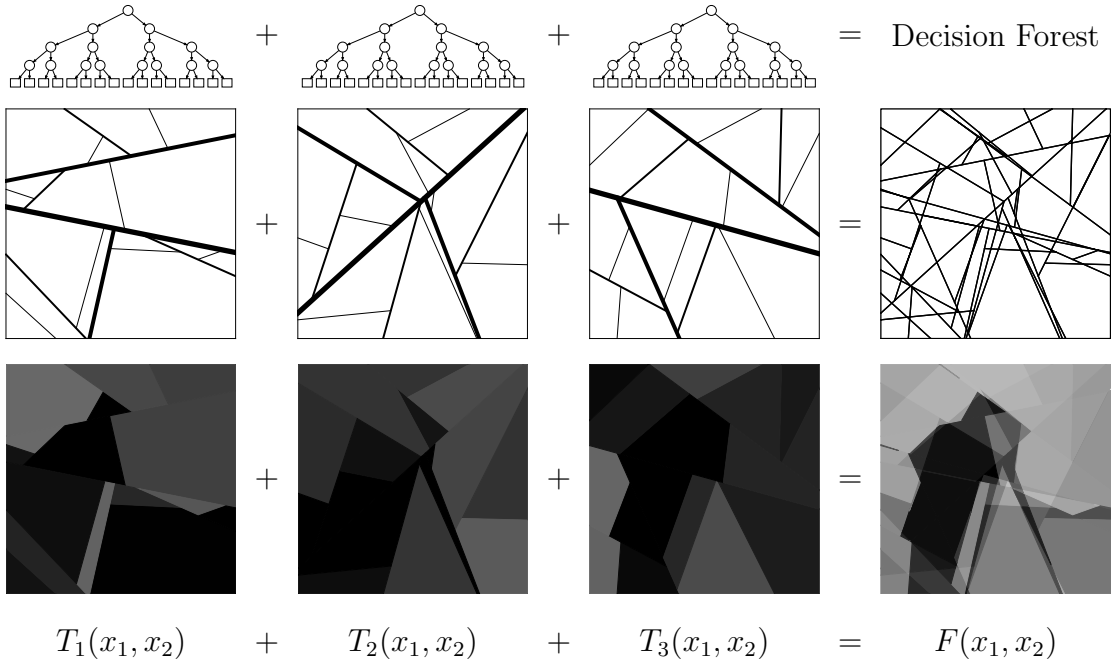

Figure 1.1: Illustration of a decision forest trained on the *Cameraman* image as a regression task: predicting grayscale values from pixel coordinates. Line segment thickness in the tree decision boundaries indicates depth: thicker lines correspond to nodes closer to the root, while thinner lines indicate deeper levels. Each tree is constrained to produce valid pixel intensities. The learning algorithm is presented in Chapter 5.

est $F(x_1, x_2)$, defined as a sum of three decision trees $F(x_1, x_2) = \sum_{t=1}^{3} T_t(x_1, x_2)$, is trained on this task using the algorithm described in Chapter 5. The figure shows both the forest output (rightmost column) and the individual outputs of each tree (left three columns).

Each decision tree defines hierarchical partition of the input space: at each decision node, the input (pixel coordinates) is directed to the left or right child based on the learned decision boundary, until a leaf node is reached that outputs a constant pixel intensity. These hierarchical decision boundaries are visualized in Fig. 1.1 using line segments whose thickness indicate the tree depth: thicker lines represent higher (earlier) levels in the tree, while thinner lines correspond deeper nodes. Each tree has a depth of 4 and a complete tree structure, producing 16 constant regions. When combined into a forest, the intersection of these three piece-wise constant models produces 163 distinct regions, highlighting a substan-

tial gain in model expressiveness without a significant increase in the number of parameters. The resulting piecewise-constant approximation of the original image gives rise to a distinctive, cubist-style visual effect. The idea of modeling an image with oblique trees/forests is first proposed in [26, 27].

Having introduced decision forests as an important model class, we now turn to their learning algorithms, where the approach differs significantly from the modern machine learning paradigm. In most established models, such as neural networks and kernel machines, learning involves defining an objective function over a model with fixed structure, then optimizing its parameters using an algorithm such as gradient descent. Decision forests, however, follow a fundamentally different strategy, relying on ensemble learning techniques of bagging and boosting. Bagging (short for bootstrap aggregation) involves training multiple base learners on different bootstrap samples of the training data and aggregating their predictions, commonly by averaging in regression or voting in classification. Boosting, by contrast, builds a strong predictor by sequentially combining weak learners, where each learner in the sequence focuses on correcting the errors made by its predecessors.

A fundamental component of these tree ensembles with both bagging and boosting is the method used to train the individual decision trees. Almost all existing implementations rely on a traditional approach based on greedy recursive partitioning: starting at the root, the input space is partitioned recursively by selecting axis-aligned splits (i.e., splits based on a single feature) that locally optimize a chosen criterion, continuing until a stopping condition is met. This approach dates back to at least the 1960s and underlies widely used decision tree algorithms such as CART [21] and C4.5 [103]. As is evident, this tree learning method contrasts sharply with the dominant modern machine learning paradigm, where one typically defines an objective function over a fixed model architecture and optimizes its parameters via an iterative learning algorithm. A more recent and principled alternative is **Tree Alternating Optimization (TAO)**, a scalable algorithm designed to learn tree-based models using alternating optimization [24]. Unlike greedy methods, TAO assumes an initial tree structure and parameters, and then refines them by optimizing one node (or a subset of nodes) at a time. Crucially,

this process guarantees a monotonic decrease in the objective function at each iteration. Another key advantage of TAO is its ability to effectively train more expressive oblique trees, which use linear combinations of features at decision nodes, unlike traditional axis-aligned trees that rely on a single feature per split. The TAO algorithm will be reviewed in detail in Chapter 2.

Using TAO as a building block, this dissertation demonstrates how two of the most widely used boosting algorithms, AdaBoost [37] and Gradient Boosting [39], can significantly benefit from properly optimized base learners. Traditional implementations of boosted decision trees typically rely on axis-aligned, suboptimally trained CART trees. Even in highly optimized frameworks such as XGBoost, LightGBM, and CatBoost, the base learners often remain restricted to axis-aligned and greedily induced trees. In contrast, this work is the first to show that leveraging more powerful oblique decision trees, trained via TAO to optimize the explicit objective functions of these boosting algorithms, leads to substantially more accurate tree ensembles, requiring fewer trees and fewer overall parameters. Central to this result is a C/C++ implementation of the TAO algorithm, developed to accelerate training within the inherently sequential nature of boosting.

After showing the benefits of alternating optimization in the base learner step of existing boosting algorithms, this dissertation takes a significant step further by introducing a method to *jointly optimize the entire decision forest* using alternating optimization. Specifically, by fixing all trees except one, the parameters of decision nodes of each tree are updated in turn with TAO, and all the leaf parameters are jointly optimized by solving a convex problem that is solved exactly and efficiently. However, due to the high flexibility of decision forests, there is a strong risk of overfitting, i.e., achieving low training error but poor generalization to unseen data. This dissertation analyzes such behavior and explores various regularization techniques to prevent overfitting. Nonetheless, controlling the flexibility and smoothness of the forest remains a challenging task. We empirically find that simple averaging of multiple optimized small forests, each trained from a different random initialization, consistently improves generalization performance, achieving accuracy that surpasses that of gradient boosting.

A particularly interesting special case of decision forests, arguably their simplest form, is a **forest of decision stumps**, where each tree is an axis-aligned split of depth one. Stump forests are functionally equivalent to **generalized additive models (GAMs)**, a class of models in which the predictive function can be expressed as a sum of univariate functions over individual features: $F(\mathbf{x}) = \sum_{d=1}^{D} F_d(x_d)$. This equivalence arises by grouping all stumps that split on the same feature dimension d, thereby defining the univariate shape function $F_d(x_d)$ for that feature. GAMs are widely regarded as an important class of interpretable models, as their structure allows for direct visualization and analysis of the individual shape functions. In this work, we apply the alternating optimization algorithm to efficiently train accurate stump forests. Unlike more general forests, the additive nature of stump forests enables explicit control over model smoothness via regularization of the complexity of individual shape functions. The resulting optimized and regularized stump forests achieve state-of-the-art test accuracy while requiring fewer stumps, showcasing the effectiveness of alternating optimization.

1.1 Contributions

In this dissertation, we study the role of optimization in learning decision forests. Since this type of models are highly non-differentiable, standard gradient-based numerical optimization methods are not applicable. Instead, we rely on alternating optimization (i.e. coordinate descent) which proves to be very effective for training tree ensembles. In the following chapters, we present the algorithms and the results of different applications of alternating optimization for this model class. We demonstrate that this approach produces more accurate and more compact decision forests than the traditional methods such as bagging and boosting.

- Chapter 2 reviews related work on decision trees and forests, and presents an overview of the Tree Alternating Optimization (TAO) algorithm.
- Chapter 3 shows how optimizing the base learner objective in the AdaBoost algorithm with TAO for oblique decision trees produces much more accurate tree ensembles while using fewer boosting iterations.

- Chapter 4 presents the improved gradient boosting decision trees with oblique splits jointly optimized over all their parameters.
- Chapter 5 extends the alternating optimization to jointly learn the whole decision forest and discusses the challenges with overfitting, and presents one effective solution.
- Chapter 6 applies alternating optimization to train regularized decision stump forests and its equivalence to the inherently interpretable generalized additive models (GAMs).
- Chapter 7 concludes this dissertation and outlines potential directions for future research.

While this dissertation focuses on the works about tree ensembles, other research projects that I worked on during my PhD include interpretable clustering [44], cost-sensitive learning [46], softmax trees [139] and its adaptive learning [67].

Chapter 2

Related work and overview of Tree Alternating Optimization

We first review the literature on traditional axis-aligned decision trees, and then on relatively less explored oblique trees. This will be followed by the literature review on ensemble learning with a focus on bagging, boosting and other existing approaches. After reviewing literature on GAMs, we then provide an overview of the TAO algorithm.

2.1 Axis Aligned Decision Trees

2.1.1 Greedy recursive partitioning

Decision trees have long history. According to the review by [87], the first published paper on this subject dates back to 1960s [95], where a recursive partitioning algorithm, named Automatic Interaction Detection (AID), was presented to learn an axis-aligned tree for regression problems. Obviously, during those times the modern view and applications of machine learning had not been developed, and the main motivation of the paper is on the more traditional statistics field of data analysis (the paper focused on survey data), in particular, a novel way of finding variable interactions. Indeed, unlike linear models which consider only additive interaction of variables, in the root-to-leaf of a decision tree the thresholded features

interact non-linearly, giving rise to a completely different way of looking at the data. We will review in detail the recursive partitioning algorithms for decision tree induction shortly with reference to the CART book [21].

Continuing on the history, independently and not so long after the [95] paper, Henrichon and Fu [56] propose a computational procedure to recursively produce axis-aligned regions for classification problems. At each split level, instead of considering only binary splits, their approach could potentially split the input space into multiple regions. They consider an arbitrary number of regions per feature, and based on heuristics and random perturbations, they select and merge to produce final regions. For multidimensional problems they check each dimension separately, and select the best one among them based on classification accuracy. And as in traditional algorithms, the procedure is repeated recursively until some predetermined minimum number of points per region is reached.

One of the earliest works of learning decision trees for classification problems in the statistics community is perhaps [93]. The core algorithm is still based on greedy recursive partitioning. Importantly, the paper introduces several types of splitting criteria, some of which (Gini index and cross entropy) have stood the test of time and are still in use today by current established implementations of trees. One recognized problem with traditionally non-parametric models as decision trees which recursively partition the input space based on the training data is the risk of over-fitting. Over-fitting is one of the fundamental problems in machine learning and statistics, where the model captures the training data with fine detail along with all the noise, so that it does not generalize well to the future unseen data. Indeed, as was indicated by the review in [87], and emphasized in the CART book [21], this problem of over-fitting by the greedy recursive partitioning algorithm might had been the main culprit in the widespread of adoption of decision trees in the statistics community before 1980s.

Classification and Regression Trees (CART) [21] is the name of the book on decision trees and of the corresponding software. While it builds on the already existing ideas of greedy recursive partitioning, the book provides several important contributions on the subject. By stating that the preliminary ideas appeared

in [18], Chapter 3 of the CART book argues for an approach to grow the tree maximally (most likely over-fitted), and then to perform pruning of the nodes based on some cost function. The authors state that this two step procedure produces in general more accurate trees on unseen data than inducing a small tree from the beginning. Before giving the details of this procedure, I will first outline how the tree growing works in the first place with reference to the CART book.

Given a training set as in standard machine learning problems (let's focus on classification), the goal is produce a binary decision tree. From the outset, we do not assume that we are given some initial tree structure or parameters, and the idea is to induce a tree from the dataset (this is why decision trees have been traditionally referred to as non-parametric models). Starting from the root node, we need to induce its two children (leaves). To do this, we define some purity criterion for a leaf node: it measures how "pure" the node is with respect to the class distribution. The CART book advocates for the Gini index and twoing criterion as a measure of purity, but states that the choice of this criterion is of secondary importance compared to how pruning is done. With some well-defined criterion of leaf purity, the algorithm then considers all possible ways each feature could be split, and selects the most optimal one among them. Given the sorted feature values, this operation could be done in linear time. The algorithm then recursively continues this procedure for the new formed leaves. It stops when the leaves contain points of only one class (i.e. pure), or contain some minimum predetermined number of points.

One of the natural measures of purity is a misclassification rate, which basically counts how many points in the leaf will be misclassified given the label from the majority class. The authors in CART provide several reasons why using this measure might not be favorable. Firstly, when splitting a node there might be cases where all possible splits result in no improvement of missclassification rate. This happens when there is class imbalance at the node to be split, and the major class dominates in both leaves for all possible splits, which will result in the same original 0-1 error of the node. No improvement in purity for all splits will prevent the tree from growing any further. Second, and more important reason against

using missclassification rate as a splitting criterion is based on the capacity of new formed leaves for better further expansion. Two different splits might have the same 0-1 error rate, but one could result in leaves which are better suited for further expansion. To illustrate this, the authors in CART provide an example (the same example appears in the seminal Elements of Statistical Learning textbook [54]), where it might be better to have a split with one pure leaf and then expand the other not-pure leaf, as opposed to a split which gives two non-pure leaves both requiring further expansion. They attribute this problem to the greedy nature of the algorithm, whereby more continuous measures of impurity can provide leaves better suited for further growing. However, apart from this example and intuitive reasoning (and perhaps empirical results), it is not firmly known why one shouldn't use a 0-1 loss as a splitting criterion.

The CART book elegantly formulates and develops the pruning problem for a given tree (fully grown in their case). In accordance with the established machine learning paradigm, it defines an objective function based on the training set loss plus hyperparameter α times regularization. For the loss they use a 0-1 misclassification rate (not Gini index!), and for regularization they use the number of leaves, which is very intuitive. Because of the discrete structure of the objective function, only some ranges of hyperparameter α values must be considered, and the resulting pruned trees can be found efficiently, and they form a nested subsequence (i.e. smaller trees are subtrees of larger trees). Then the optimal α and the corresponding pruned tree can be selected using standard cross validation techniques.

Ross Quinlan developed a series of decision tree algorithms [103], [104]: ID3, C4.5 and C5.0. The book of C4.5 does not seem to be freely available, and C5.0 is not documented, but based on the review papers and textbooks [87, 54, 96], we can state C5.0 is the most improved version among them, and turns out to be very similar to CART. The difference is in the splitting criteria (it uses information gain), and possibly on how the categorical variables are handled. It also has the additional feature of extracting simplified rules from the trained decision tree.

Kim and Loh [73], Loh [86] developed a series of works on axis-aligned decision

trees. As opposed to CART and C5.0, which perform exact search through all the features and intervals to find the optimal feature index and threshold pair, Loh [86] argues that there might be statistical bias towards some features with more frequent values. Instead they propose to use traditional statistical tests to detect feature interactions such as F and chi-squared tests, and select the feature to split based on these more complicated tools from traditional statistics. Their empirical performance does not seem to improve upon CART or C5.0 [140], and their software has not been widely used or recognized in the machine learning community.

To summarize, the greedy recursive partitioning approach for learning axis-aligned decision trees is still considered a leading algorithm to this day. Its main advantage are the acceptable performance on some problems and the vast empirical development and validation by the statistics and machine learning community as evidenced by this rich literature. Another benefit of the greedy approach is the speed of training, which is especially useful in the context of boosting: one needs to train hundreds of trees sequentially and having a fast algorithm in each boosting step is essential. However, an important drawback of the greedy recursive partitioning is its suboptimality: except for the pruning step, a global objective function is never considered, and the decision node splits would be left in their suboptimal states with respect to the overall tree objective. Another important disadvantage of greedy recursive partitioning is its poor performance or even inapplicability on trees beyond axis-aligned splits: as will be discussed later, this approach gives subpar accuracy on oblique (linear) splits; and is not straightforward to expand on other tree-based models such neural trees, kernel trees, etc.

2.1.2 Brute force search

Obviously, learning a globally optimal axis-aligned decision tree is NP-hard. [60] has a proof of NP-hardness for the problem of learning a perfect tree, one having 0% on the train set, but with a minimum number of expected features tests, and their proof can be adapted to the standard problem of minimizing a loss under the constraint of tree size (e.g. a limit on the tree depth). In spite of the

NP-hardness, however, recently there have been a series of works that attempt to learn globally optimal trees. Bertsimas and Dunn [11] propose to encode a tree learning problem as a mixed integer optimization (MIO) problem, and then to use commercial, highly optimized implementations of MIO solvers. But because the problem has exponential complexity, unsurprisingly, their experiments have been limited to toy size datasets (at most 5000 train instances) and to very small trees of up to depth 4. Verwer and Zhang [129] propose a different formulation of the MIO problem, and claims to improve the runtime. However, their experiments are also limited to very small datasets and to very shallow trees. [59], [83], [2] are another set of works addressing the globally optimal tree learning problem. Unlike [11], they do not rely on generic MIO solvers, but instead develop their own custom algorithm. However its main ideas are still based on the branch-and-bound search pruning methods (also used in MIO solvers), and their experiments are also limited to small scale problems. In summary, these lines of work attempting to learn globally optimal trees face the exponential complexity of the search space, and are not scalable even to moderately sized problems (not even close to MNIST scale dataset).

2.2 Oblique Decision Trees

Oblique trees use a linear combination of features at a decision node instead of just thresholding a single feature. Obviously, they have more modeling capacity and predictive power. However, the literature on this type of trees is not as rich as for axis-aligned trees, and their practical use has been quite limited. The main reason for this is the fundamental difficulty of learning such trees. While greedy recursive partitioning produces adequate results for axis-aligned trees, the same could not be said for oblique trees. Indeed, in the conclusion of the review paper Loh [87] states the problem of learning linear combination splits is still “elusive”.

2.2.1 Greedy recursive partitioning

Section 5.2 of the CART book [21] motivates the use of linear splits at a decision node, and adapts the split finding procedure of the greedy recursive partitioning of axis-aligned trees to search through the hyperplane coefficients. While we can exactly enumerate all the possible axis-aligned splits and choose the best one, a similar procedure cannot be made for oblique nodes. The search space is vastly complex, and one has to resort for approximate heuristics. The CART book proposes to use some coordinate descent based method: we cycle through each feature, increment or decrement the weight value of that feature by either -0.25 or 0.25 or 0.0, and select the one that gives a better value of the purity criterion, and repeat this cycle. The authors admit that this is "the result of considerable experimentation", and that "there is still room for improvement".

[97] is a work very similar to the CART approach for linear combination splits, but with the small addition of randomization. After the learned weights are found in the spirit similar to CART, in order to avoid a local optima, a random direction is chosen and a line search is performed in this direction. If this step improves the purity criterion, then another stage of local search is performed. The number of times to do this jump to random direction is left as a hyperparameter. The GUIDE software [86] has a linear split as one of the options. But the form of the linear split is very limited: the linear combination considers only two features. It choose these two features based on some clever statistical interaction measures.

2.2.2 Other approaches

Bennett [8], Bennett and Blue [9] formulate the problem of oblique tree learning as a linear program. Instead of inducing a tree, it takes some initial tree structure and parameters. The basic idea is to express the root-to-leaf path as a set of linear inequalities and form the objective function to minimize some form of the 0-1 loss over the leaves. The formulated problem is NP-hard, but it is solved approximately using the Frank-Wolfe algorithm and the extreme point method similar to the one used in the Simplex algorithm. In another paper, Bennett and Blue [10] present an SVM formulation for a 3 decision node tree problem, and adopt techniques from

the SVM literature to solve it. All these works are limited only to very small trees because of the exponential search space.

Norouzi et al. [98] formulate the problem of globally optimizing an oblique tree as a structured prediction problem with latent variables. Unlike tree growing methods, the paper assumes some initial tree structure and parameters as given. But instead of optimizing the tree objective function directly, they resort to an upper bound surrogate objective amenable to gradient based optimization. It is not very clear to what extent the upper bound is tight or loose, and the experimental results do not investigate this. Good et al. [49] obtain oblique trees using the composition of linear feature transformation and axis-aligned trees trained on these transformed features. Both these steps are learned jointly using alternating optimization. The axis-aligned tree learning step is still based on greedy recursive partitioning approach. Unfortunately, the source code for both [98] and [49] are not publicly available, and so we could not analyze and validate its results.

Carreira-Perpiñán and Tavallali [24], Zharmagambetov and Carreira-Perpiñán [135] proposed an algorithmic framework to learn oblique trees and other tree based models, called Tree Alternating Optimization (TAO). The algorithm has a guarantee of a monotonic decrease of an objective function over tree parameters and is adaptable to different objective functions and models. The empirical comparison shows that it produces state-of-the-art accuracy among different decision tree methods [136, 140]. We will provide the overview of TAO at the end of this Chapter.

2.2.3 Soft trees

Soft decision trees are one type oblique trees where each decision node routes an instance to both children with some nonzero probability. The final prediction by a soft tree is given by a weighted average of all the leaves. Hierarchical mixture of experts is a representative model of this type, first presented in [65]. Fixing the structure of the tree, they learn parameters of the nodes using an expectation maximization algorithm, although traditional gradient-based methods are also applicable. A major issue with these types of trees is slower inference because all

root-to-leaf paths must be followed, and a lack of interpretability inherent to traditional hard trees. And "hardening" the trained soft tree by only following the child with highest probability leads to significant accuracy degradation [47]. Multiple papers explore different variations of soft decision trees [99, 126, 42, 68] and its applications in language modeling [50, 94, 7].

2.3 Decision forests

The general goal of ensemble learning is to combine multiple base learners to produce a more accurate final model [77, 141]. By far the most widely established methods for this purpose are bagging and boosting.

2.3.1 Bagging

Bagging stands for bootstrap aggregation [19]. The general idea is to instill diversity into the base learners by training them on different samples of the train set. In bagging the sample is usually a bootstrap sample, i.e. a random sample with replacement from the train set, but other variations are possible such as sampling smaller sets without replacement. For each bagged sample we train our base model independently, and combine their output by averaging for regression or by majority vote or averaged probabilities for classification. The general motivation and theoretical explanation for bagging is the reduced variance of the final model: combining multiple unstable models can result in a smoother, more stable model. Apart from inducing diversity through the sampling mechanism, one could also rely on random behaviors intrinsic to the base learner. For decision tree learners Amit and Geman [4], Ho [58], Breiman [15] propose to use random subsets of features during the split finding step of the greedy recursive partitioning instead of enumerating through all the features. This additional step along with bagging helps to instill more diversity among the trees, and as evidenced by its empirical performance, the resulting model, called Random Forest, has been widely regarded as one of the best-off-the-shelf method in the machine learning and statistics community. Unlike learning a single tree with CART, where one grows a large tree

and prunes it back, the approach in random forests is to just grow a large tree without pruning on bootstrap samples, and rely on the averaging mechanism to reduce variance and overfitting. Geurts et al. [48] propose to use yet another way to induce diversity: on top randomly chosen subsets of features randomly sample a subset of threshold values and find the best one among them.

A theoretical analysis of random forests is still elusive. Apart from the intuitive understanding in terms of diversity and variance reduction, there has been quite limited success toward gaining insight into theoretical properties of random forests. Biau and Scornet [12] provide a comprehensive overview of the subject with a focus on "the mathematical forces driving the algorithm". As noted in the review, most of work assessing the theoretical properties of random forests focus not on the actual algorithm, but on a very simplified splitting procedure. One such example presented in [16] is as follows: assuming the input space is in hypercube $[0, 1]^D$, the simplified procedure randomly chooses a coordinate and performs the split at the center of the region independently of the training data and repeats this until it reaches a complete tree up to a given depth. Such simplified models are clearly quite far from the real algorithm, and even with those, one can obtain only asymptotic statistical consistency results. Another lines of work [5], [3] note the link and similarity between random forests and kernel estimates. Mentch and Zhou [92] empirically shows that the randomness in feature selection at each split step acts as a form of regularization in random forests. Biau and Scornet [12] concludes their recent review on random forests by stating that "the present results are insufficient to explain in full generality the remarkable behavior of random forests".

2.3.2 AdaBoost

AdaBoost The idea of boosting originated in the computational learning theory, where in 1989 Kearns and Valiant [72] posed a question of the possibility of converting a weak learning algorithm (slightly better than random guessing) into the one with high accuracy in the theoretical framework of probably approximately correct (PAC) learning. To answer this question, Schapire [112] described a constructive proof demonstrating the possibility of "boosting" a weak learning algorithm to

arbitrarily high accuracy, although the presented method was not practical. In the seminal paper of [37], the first practical boosting algorithm, AdaBoost was presented, immediately attracting much attention in the statistics and machine learning community. The main idea of the algorithm consist of sequentially training and combining weak learners where each learner focuses on the instances which were hard to classify by the previous learner. This is accomplished by maintaining weights per instance, and at each boosting step these weights readjust based on their difficulty for the past learner. We will provide the details of the algorithm in Chapter 3.

Statistical view of boosting The theoretical views on boosting is a fascinating subject. We delay its treatment for a later section, and review the widely influential statistical view on boosting. As observed by Breiman [14] and comprehensively presented in [38], the sequence of boosting iterations can be considered as forward stagewise optimization steps for some well-defined loss function. Given an additive model of decision trees and the exponential loss for classification, AdaBoost can also be derived as an algorithm which greedily adds trees to minimize the global exponential loss, surprisingly very different view on how the boosting was originally developed. Besides being simple and elegant, this statistical view based on optimization has been very influential in the machine learning and statistics field in that it gave rise to many different variations of boosting, one prominent example being gradient boosting.

Variations of AdaBoost Many variations of AdaBoost have been proposed. I will review some of the notable ones, but more comprehensive treatment of the subject can be found in the book by Schapire and Freund [113].The first AdaBoost algorithm was designed only for binary classification, and the most straightforward extension to multiclass is AdaBoost.M1. However, AdaBoost.M1 has more stringent requirement for base learners: their accuracy must be better than 50%. Clearly, truly weak learners better than just random guessing might not satisfy that requirement. Zhu et al. [142] addresses this by special encoding of the ground truth output and using the statistical view of boosting to derive SAMME algo-

rithm, which truly treats base learners to be weak, i.e. they must be better than just random guessing.

The original AdaBoost limits the base learner predictions to be in the range $[-1, 1]$, but the Schapire and Singer [114] extend this to base learners with arbitrary output range, and presents AdaBoost.MH, designed to optimize a Hamming Loss, and applicable to multiclass, multilabel problems. According to [38], the empirical performance indicates that AdaBoost.MH tends to be the most accurate variation of AdaBoost, although it is difficult to make firm conclusions. AdaBoost.MR is yet another version, derived for ranking problems. Base learners in both AdaBoost.MH and AdaBoost.MR are expected to output vectors as opposed to scalar values in the original AdaBoost, but in practice they resort to scalar-output trees trained in one-vs-all or one-vs-one binary reductions.

By simply using a logistic loss function in place of exponential, Friedman et al. [38] also presented a LogitBoost algorithm, separately for binary and multiclass problems. The multiclass LogisBoost has flexibility in how it defines the base class or whether it uses one-vs-all or one-vs-one binary reductions, and Li [80], Sun et al. [122] investigate these variations and propose improvements for LogisBoost. Saberian and Vasconcelos [110] propose a new formulation of multiclass boosting using margin enforcing loss functions and optimal set of codewords, and using a gradient boosting framework present two generic boosting procedures: CD-MCBoost and GD-MCBoost.

Theory of AdaBoost Multiple perspectives exist on how and why boosting works. Original developments in the PAC framework in the field of computational learning theory give bounds only on the train error with no considerations for generalization performance. With decision tree base learners, given that we can arbitrarily decrease train error by growing trees deeper or by adding more of those, the theoretical bound on the train error is not a very helpful explanation. Another theoretical view on boosting can be provided in terms of Vapnik–Chervonenkis (VC) dimension, but it predicts that AdaBoost must overfit, once more and more trees are added to the ensemble, which doesn't occur in practice (Chapter 4 in [113]).

Another perspective on AdaBoost is based on the large margins of a classifier. Popularized in the support vector machine literature, the concept of the margin specifies how far the boundary of the decision surface is from training instances. Intuitively, we expect larger margins of the classifier to correspond to better generalization performance. The work by Schapire et al. [115] tries to give a margin-based explanation for the high test accuracy of boosting. By noting the final form of the classifier in AdaBoost is given by the weighted majority vote of base learners, they define the margin for a point as the difference between the weighted vote for the correct class minus the maximal weight given to any incorrect class. With some theory and mostly based on empirical results, Schapire et al. [115] illustrate that as more and more boosting iterations are performed, the distribution of margins of the training instances increase and even reach maximum. However, approaches that directly modify AdaBoost to increase the margin ([14, 106]) do not produce better test error, indicating the major insufficiency of the margin-based theory of AdaBoost.

Breiman [20] attempts to explain the success of boosting from the perspective of variance and bias decomposition. In analogy with bagging, the paper views instance weight modifications of AdaBoost as a special form of resampling and develops an algorithm called arcing, short for adaptive resampling and combining. Empirically, arcing seems to produce more accurate ensembles than simple bagging, and based on synthetic datasets where some measures of bias and variance can be estimated, Breiman [20] concludes that arcing, and thus AdaBoost, might have better variance reduction properties than bagging. This might be simple and quite helpful view on AdaBoost, however, Schapire et al. [115] illustrates several counterexamples where variance reduction is not enough to explain the low test error of boosting.

The statistical view considers AdaBoost as forward stagewise additive modeling to minimize exponential loss. It too however, does not explain several mysterious properties of AdaBoost. Through a series of simulated experiments, [91] provide several examples where the behavior of boosting does not correspond to what the statistical view would have predicted. Some of the examples illustrate how decision

stumps (a tree with only 2 leaves) overfit, whereas an 8-node tree does not or how early stopping might not help with overfitting, among others.

The theory of boosting is an unsettled question, as are other mysterious behaviors of many of the more complex machine learning models such neural nets. One major drawback of almost all the perspectives on boosting is that they do not take into account the properties of decision tree base learners. The intersection of multiple hierarchical piece-wise constant models gives rise to a very complex partition of the input space, and boosting can be viewed as one way of achieving it with good generalization performance. Andreas Buja elegantly summarizes the multiple perspectives of AdaBoost: "There is no single 'true' interpretation of anything; interpretation is a vehicle in the service of human comprehension."

2.3.3 Gradient Boosting

Friedman [39] introduced the gradient boosting 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 [39], ranking [23], and density estimation [108], [22]. At each boosting iteration, stochastic gradient boosting [40] uses random subsampling of training points with the goal to inject more diversity into the ensemble and faster training time. While the original gradient boosting formulation [39] uses only the functional gradient information, one can also incorporate second order information, and derive LogitBoost [38] 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 gradient boosting forest construction such as penalizing a tree structure or the value of a leaf prediction [64]. 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 gradient boosting software. Finally, XGBoost [28], LightGBM [71] and CatBoost [102] are some of the popular, quite recent and highly optimized implementations of gradient boosting forests. At a more fundamental algorithmic level, they implement the same core gradient boosting procedure, but they dif-

fer in specific practical implementation details such as usage of histograms, clever subsampling, special handling of categorical features, etc. We will give the details of the gradient boosting framework in Chapter 4.

2.3.4 The choice of base learners

By far the most widespread and successful base learners used in boosting are decision trees. Although other models are possible to be boosted, decades of empirical knowledge support that ensembles of decision trees produced by bagging or boosting to be the best off-the-shelf model in data scientist's toolkit.

The de facto choice to optimize individual base learners in boosting are axis-aligned decision trees with a greedy algorithm. This widely established paradigm of learning trees dates back to the 1960s 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 [54]), though in the gradient boosting 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 gradient boosting framework. Traditional algorithms on decision tree learning, such as CART [21], C5.0 [104], ID3 [103], 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 [54]. Works based on greedy recursive partitioning for oblique trees in isolation [21, 97] or in a bagged forest [128, 69] produce little improvement and considerably increase the model size. Axis-aligned trees with linear models at the leaves have been used in GB [118], but such trees are still learned greedily. GB regression forests have been used for a face alignment problem [70], 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. Owing to the TAO algorithm, [45, 137, 138, 43] effectively use sparse oblique trees in boosting and produce more accurate ensembles. We will present in more detail these results in Chapters 3 and 4.

2.3.5 Other approaches

Few works have explored the learning of decision forests without explicitly using the ensembling techniques of bagging and boosting. Sorokina et al. [121] introduced the Grove, an additive model consisting of a small, fixed number of regression trees. In this approach, each tree is iteratively grown from the root using greedy recursive partitioning on the residuals of the other trees, following a round-robin schedule. Their algorithm uses complex heuristics to grow the trees in a layered fashion, aiming to control overfitting. Ultimately, however, the authors resort to bagging multiple Groves to achieve competitive test accuracy. More recently, Tan et al. [125] proposed Fast Interpretable Greedy-Tree Sums (FIGS), a method for simultaneously growing a small set of additive decision trees with a focus on interpretability. The main goal is to construct a compact and explainable decision forest as opposed to competing with XGBoost in accuracy. Importantly, both Grove and FIGS do not define an explicit objective function over a parametric decision forest of fixed size. Instead, they extend the traditional greedy recursive partitioning approach to multi-tree settings, without adopting a principled optimization framework unlike in [25].

Bayesian additive regression trees (BART) [29, 30] is a purely Bayesian approach for a sum of independent trees model (i.e. a decision forest). It defines a prior distribution over the tree structures and leaf values, and uses Bayesian inference techniques to learn the posterior distribution. Although the mathematical formulation of BART is quite complicated, one should note that the final model prediction is given as an average over individual tree predictions, making it quite similar to the ensembling techniques. Hill et al. [57] provides a more recent, comprehensive overview of BART.

Several works exist that take an existing decision forest and refine the leaf

parameters or perform some form of pruning. One notable example is RuleFit [41], which aims to extract a compact, interpretable set of rules from a pre-trained tree ensemble. It does so by enumerating all possible paths from root to leaf (i.e., decision rules), treating them as basis functions, and then fitting their coefficients using LASSO regression to promote sparsity, and select only a few rules. The idea of fitting all the leaf values jointly is very clever, and can be used to refine any other existing tree ensemble. For example, Ren et al. [107] apply this method for Random Forests, and achieves improved accuracy. Liu and Mazumder [84] propose an approach to post-process tree ensembles by pruning depth layers from individual trees with the goal reducing model size. However, none of these works change the decision nodes parameters which typically come from the greedy recursive partitioning approach.

Finally, multiple papers propose to learn soft tree ensembles [75, 55, 61]. As these are differentiable, end-to-end gradient-based optimization methods are used to learn them. However, soft tree ensembles, just like soft decision trees, have exponential complexity with the tree depth in both during training and inference, and their empirical performance against modern tree boosting frameworks such XGBoost and LightGBM is questionable.

2.4 Generalized Additive Models

Although this dissertation focuses on decision forests, generalized additive models (GAMs) have some relations, particularly because tree ensembles provide one of the most effective methods for training GAMs. Accordingly, we provide a brief review of relevant literature in this area to establish connection and context. GAMs have a rich history in statistics [52]. As shape functions, splines are used almost exclusively, and with many different variations, including cubic splines, penalized B-splines [34], and thin plate regression splines [131]. Backfitting [17, 52], a form of alternating optimization, and penalized (iteratively reweighted) least squares are commonly used to learn these GAMs. Selecting the placement and number of knots (a point where two spline segments meet) has been an open problem with many

different methods proposed on this [13, 109, 35]. In our approach with alternating optimization in Chapter 6, the placement of knots are determined on-the-fly during optimization where stumps choose an optimal feature/threshold pair. Apart from splines, non-parametric methods based on trend filtering have been explored as shape functions [101, 111, 85]. We refer the reader to the books [53, 132] for a more comprehensive treatment of GAMs in statistics literature.

More recently, other models for GAMs have been explored in the machine learning community. Lou et al. [88] performed experimental comparison of different algorithms and shape functions for GAMs, and found out that ensembles of trees based on boosting and bagging produce most accurate results. Follow-up work [89] extended this tree-based approach to model pairwise terms, though the idea of modeling interactions existed long before (e.g. section 9.5 in the GAM book [53]). Agarwal et al. [1] used neural networks with special activation functions to model each of the 1D shapes in GAMs. Radenovic et al. [105] proposed to use a single shared basis neural network to model all the shape functions and further extended them to pairwise interactions. Ibrahim et al. [62] used soft decision trees to model both univariate and bivariate shapes, and imposed structural sparsity constraints on the number of terms. Given the differentiability of neural networks and soft decision trees, end-to-end SGD-based optimization methods were used to learn all these GAMs.

2.5 Overview of Tree Alternating Optimization

In this section we provide an overview of the TAO algorithm as it is an important building block for the subsequent chapters. We first define formally the notation for decision trees and then explain the algorithm.

Let $\{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^D \times \mathbb{R}$ be our training set of size N of D-dimensional input features and y_n is a target. For 1D regression tasks, $y_n \in \mathbb{R}$, for binary classification $y_n \in \{0, 1\}$, for multiclass classification with $K > 2$ number of classes, $y_n \in \{1, 2, \dots, K\}$. We denote the *decision tree* as $\boldsymbol{\tau}(\mathbf{x}; \boldsymbol{\Theta})$, a rooted binary tree with a set of decision (internal) nodes $\mathcal{N}_{\text{dec}}$ and a set of leaf nodes $\mathcal{N}_{\text{leaf}}$. Each decision node

$i \in \mathcal{N}_{\text{dec}}$ has a decision function $g_i(\mathbf{x};\boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \{\texttt{left}_i, \texttt{right}_i\} \subset \{\mathcal{N}_{\text{dec}} \cup \mathcal{N}_{\text{leaf}}\}$ that sends an instance $\mathbf{x}$ to its left or to its right child. For oblique (linear) decision nodes: "if $\mathbf{w}_i^T\mathbf{x} + w_{i0} \geq 0$ then $g_i(\mathbf{x}) = \texttt{right}_i$, otherwise $g_i(\mathbf{x}) = \texttt{left}_i$" where the learnable parameters are $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$. Axis-aligned splits are a special case of this, where $\mathbf{w}_i$ is a one-hot encoded representation of the feature index, and w_{i0} is a splitting threshold. Each leaf $j \in \mathcal{N}_{\text{leaf}}$ contains a predictive function $\mathbf{f}_j(\mathbf{x};\boldsymbol{\theta}_j)$ that produces the actual output of the tree $\boldsymbol{\tau}(\mathbf{x};\boldsymbol{\Theta})$ for an instance $\mathbf{x}$. In this dissertation, we mostly consider constant leaves $\mathbf{f}_j(\mathbf{x};\boldsymbol{\theta}_j) = \boldsymbol{\theta}_j$ that output either a constant scalar $\theta_j \in \mathbb{R}$ or a constant vector $\boldsymbol{\theta}_j \in \mathbb{R}^K$. The predictive function of the whole tree $\boldsymbol{\tau}(\mathbf{x};\boldsymbol{\Theta})$ then works by routing an instance $\mathbf{x}$ to exactly one leaf through a root-leaf path of decision nodes and outputting that leaf's prediction.

Now we describe the TAO algorithm. TAO is a general method for optimizing a given objective function over a given decision tree model. Unlike CART-type methods, TAO works similarly to how one would optimize a neural network: by taking an initial tree structure (network architecture) and parameters (network weights) it performs alternating optimization over the nodes (gradient descent in a neural net) to monotonically decrease the objective function. Given an initial tree $\boldsymbol{\tau}(\mathbf{x};\boldsymbol{\Theta})$ of fixed structure (e.g. a complete tree of depth Δ) and initial parameters (e.g. random), the goal of TAO is to minimize the following objective:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(y_n, \boldsymbol{\tau}(\mathbf{x}_n;\boldsymbol{\Theta})) + \lambda \sum_{i \in \mathcal{N}_{\text{dec}}} \|\mathbf{w}_i\|_1 \tag{2.1}$$

where $L(\cdot,\cdot)$ is the loss function, $\boldsymbol{\Theta} = \{\mathbf{w}_i, w_{i0}\}_{i \in \mathcal{N}_{\text{dec}}} \cup \{\boldsymbol{\theta}_j\}_{j \in \mathcal{N}_{\text{leaf}}}$ are the set of all learnable model parameters, and there is an ℓ_1 penalty over the weight vectors to promote sparsity via hyperparameter $\lambda \geq 0$.

The TAO algorithm is based on two theorems. First, the *separability condition* states that eq. (2.1) separates over a set of non-descendant nodes, e.g. all the nodes at a given depth. This is a consequence of the tree making hard decisions. All such non-descendant nodes can be optimized independently and in parallel. Second, the *reduced problem over a node* states that optimizing the top-level problem of eq. (2.1) over the parameters of a given node $i \in \{\mathcal{N}_{\text{dec}} \cup \mathcal{N}_{\text{leaf}}\}$ reduces to a simpler, well-defined problem involving only the training instances that currently reach that

node i (the *reduced set* $\mathcal{R}_i \subset \{1, \ldots, N\}$). The exact form of the reduced problem differs for leaves and for decision nodes:

- For a decision node $i \in \mathcal{N}_{\text{dec}}$, the top-level problem of eq. (2.1) reduces to a *weighted 0/1 loss binary classification problem*:

$$E_i(\mathbf{w}_i, w_{i0}) = \sum_{n \in \mathcal{R}_i} c_n \, \overline{L}(\overline{y}_n, g_i(\mathbf{x}_n)) + \lambda \left\| \mathbf{w}_i \right\|_1 \tag{2.2}$$

  where $\overline{L}(\cdot, \cdot)$ is the 0/1 loss, $\overline{y}_n \in \{\texttt{left}_i, \texttt{right}_i\}$ is a pseudolabel indicating the "best" child (i.e., the child that gives the lower value of the loss between the left and right subtrees) for the instance $\mathbf{x}_n$, and $c_n \geq 0$ is the loss difference between the "other" child and the "best" child for the instance $\mathbf{x}_n$. The points for which $c_n = 0$ are referred to as "don't care" points, as it does not matter where they end up with since both child subtrees produce the same value of the loss for these points. This reduced problem over an oblique node is in general NP-hard, but it can be approximated well with a surrogate loss such as the cross-entropy (i.e., solving a logistic regression). We can ensure a monotonic decrease of the top-level objective (2.1) by accepting the update only if it improves (2.2).

- For leaf node $j \in \mathcal{N}_{\text{leaf}}$, the top-level problem of eq. (2.1) reduces to a form involving the original loss but only over the parameters of the leaf predictor function $\mathbf{f}_j(\cdot)$ and its reduced set $\mathcal{R}_j$:

$$E_j(\boldsymbol{\theta}_j) = \sum_{n \in \mathcal{R}_j} L(y_n, \mathbf{f}_j(\mathbf{x}_n, \boldsymbol{\theta}_j)) \tag{2.3}$$

  where $L(\cdot, \cdot)$ is the same loss function of eq. (2.1). For constant leaf predictors $\mathbf{f}_j(\mathbf{x}_n, \boldsymbol{\theta}_j) = \boldsymbol{\theta}_j$, the solution of this reduced problem is typically simple: the majority class label for 0/1 classification loss and the average over $\{y_n\}_{n \in \mathcal{R}_j}$ for ℓ_2 regression loss.

While these theorems do not prescribe the order in which the nodes should be optimized, most commonly used order is a reverse breadth-first search order: all the nodes at a given depth are optimized in parallel, starting from the deepest ones until the root. Each optimization subproblem over decision nodes involves

input initial tree $\boldsymbol{\tau}(\cdot, \boldsymbol{\Theta})$ of depth Δ, training set $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, regularization parameter $\lambda \geq 0$, number of iterations I
repeat for I iterations
  for $d = \Delta$ **to** 0 **do**
    for all nodes *node* at depth d
      if *node* is a leaf
        solve the leaf reduced problem of eq. (2.2)
        e.g. find the majority class label or find the average
      else
        fit an ℓ_1-regularized logistic regression for eq. (2.3)
      end if
    end for
  end for
return tree $\boldsymbol{\tau}(\cdot; \boldsymbol{\Theta})$

Figure 2.1: Pseudocode of TAO for oblique decision trees

solving an ℓ_1-regularized logistic regression, and over leaves involves finding the optimal constant predictor. One optimization pass through all the nodes defines one TAO iteration I. The algorithm typically converges within 20 TAO iterations. As an initial tree, a complete tree of a given depth Δ is used with initial parameters set randomly. Because of the ℓ_1-penalty, some decision node weights can be entirely zero making them irrelevant and thus pruned, leading to a smaller final tree structure. The hyperparameters of the model are the depth Δ of the tree and the ℓ_1 regularization parameter λ. Fig. 2.1 provides the pseudocode of the TAO algorithm. By ensuring that the (approximate) solution of the reduced problem of a decision node improves upon the previous node parameter values, TAO is guaranteed to decrease the objective function (2.1) monotonically.

We illustrate the steps of the TAO algorithm on a simple 2D classification problem using 0/1 loss, as shown in fig. 2.2. The initial decision tree has depth $\Delta = 3$, resulting in 8 leaf nodes with constant class labels. The first row of the

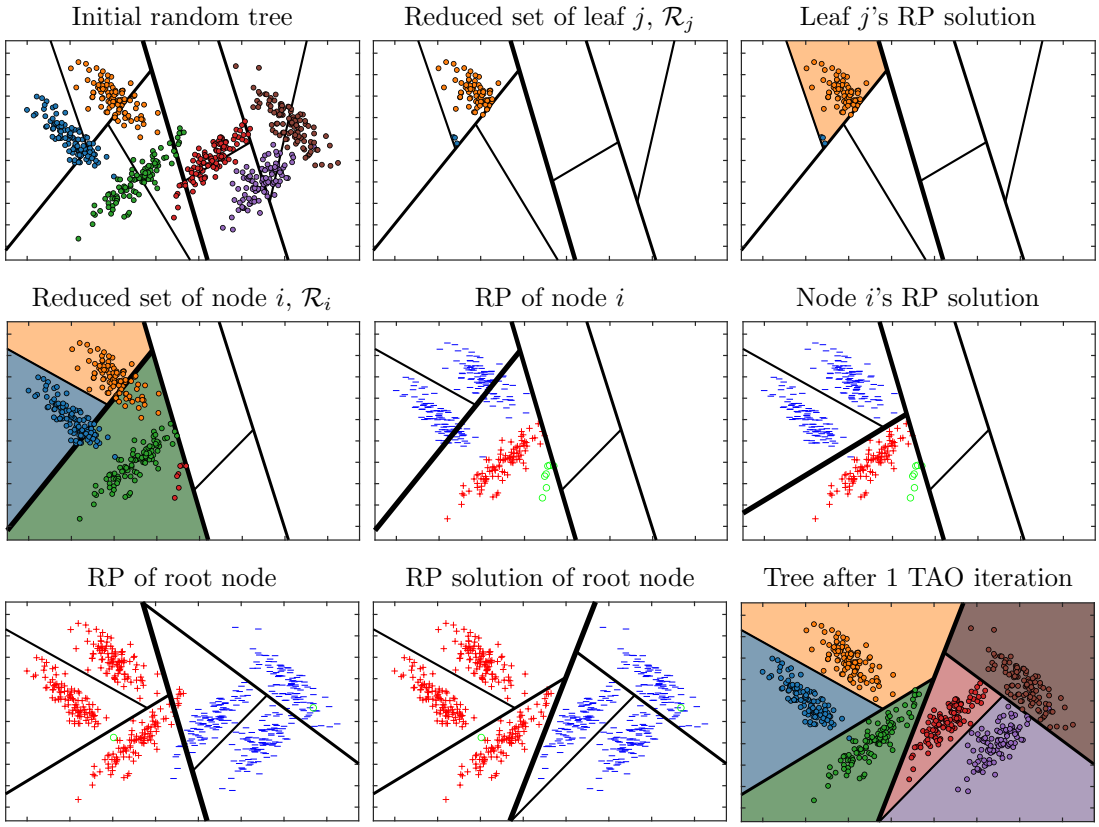

Figure 2.2: Illustration of the TAO algorithm on a toy 2D classification problem. "RP" stands for Reduced Problem.

figure shows, from left to right, the randomly initialized tree, the reduced set corresponding to one of its leaf nodes, and the solution to its reduced problem (which simply assigns the majority class label). The second row illustrates the update of a specific internal decision node: it displays the node's reduced set, the reduced problem, and its solution. Notably, in the reduced problem, certain points from the red class are marked as "don't care" and are visualized as green circles. This occurs because both of the node's child subtrees misclassify these points, making them irrelevant for optimizing this specific decision node. The third row shows the reduced problem and solution at the root node, followed by the updated tree after one complete iteration of TAO. Note how the final tree has only 6 leaves thanks to implicit pruning by the ℓ_1 regularization.

Chapter 3

AdaBoost with optimized base learners

The previous chapter reviewed the literature on learning tree ensembles, highlighting the significance of AdaBoost as the first widely successful boosting algorithm. It also noted that, in practice, AdaBoost most commonly employs greedily induced, axis-aligned decision trees as base learners. In this chapter, we investigate how replacing these with more expressive oblique decision trees, trained with alternating optimization, the TAO algorithm, improves the performance of AdaBoost. This chapter is based on papers [45, 137, 138], and more details can be found there.

3.1 Overview of AdaBoost

As reviewed in the previous chapter, there are numerous variations of the AdaBoost algorithm. While we believe that all of them could benefit from better-optimized oblique decision trees, this chapter focuses on evaluating only a select few. Specifically, we concentrate on two of the most widely used variants: AdaBoost.M1 and SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function).

The pseudocode of SAMME is presented in Algorithm 3.1. In both AdaBoost.M1 and SAMME, each boosting step involves training a base learner (an oblique tree in our case) to minimize the weighted misclassification loss (I is an indicator func-

input training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where $y_n \in \{1, \ldots, K\}$,
base learner $\boldsymbol{\tau}(\cdot)$, number of boosting steps T
initial instance weights: $\{w_n = \frac{1}{N}\}_{n=1}^N$, learning rate η
for $t = 1$ **to** T **do**
  Train a base learner ($\boldsymbol{\tau}_t$) to minimize eq (3.1)
  Obtain predictions: $\{\hat{y}_n\}_{n=1}^N \leftarrow \mathbf{T}_t(\{\mathbf{x}_n\}_{n=1}^N)$
  Compute weighted misclassification loss E in eq. (3.1)
  if $E \geq 1 - \frac{1}{K}$
    set $T = t - 1$, exit loop
  end if
  Compute $\alpha_t = \eta \cdot (\log \dfrac{1 - E}{E} + \log (K - 1))$
  Set $w_n \leftarrow w_n \cdot \exp(\alpha_t \cdot I(y_n \neq \hat{y}_n))$
  renormalize $w_1, \ldots, w_N$ to sum to 1
end for
return $F(\mathbf{x}) = \arg\max_k \sum_{t=1}^T \alpha_t \cdot I(\boldsymbol{\tau}_t(\mathbf{x}) = k)$

Figure 3.1: AdaBoost SAMME pseudocode. The pseudocode for AdaBoost.M1 is very similar.

tion):

$$E = \sum_{n=1}^{N} w_n \, I(y_n \neq \boldsymbol{\tau}_t(\mathbf{x}_n)) \tag{3.1}$$

where $\boldsymbol{\tau}_t(\cdot)$ is the prediction of the current (t-th) tree and w_n is the weight per input instance (initially equal to $w_n = \frac{1}{N}$ for $n = 1, \ldots, N$). For boosting to proceed, the training loss E (eq. (3.1)) must be lower than some threshold, which differs in these two versions: in AdaBoost.M1 the threshold is $\frac{1}{2}$, and in SAMME it is $1 - \frac{1}{K}$ (where K is the number of classes). If this weak learning condition is not met, the boosting procedure will terminate. In the multiclass case ($K > 2$), AdaBoost.M1 has more stringent requirement on base learners than SAMME, but with fairly strong base learners this should not be an issue [113]. Once the base learner is trained, both algorithms estimate the weight α_t (see Algorithm 3.1) of the current base learner. AdaBoost.M1 differs from SAMME here in that there

is no $\log(K-1)$ term in the sum. The final step in a single boosting iteration is to change the distribution of the instance weights for the next base learner. Both algorithms increase the weights of the misclassified instances and decrease the weights of the correctly classified instances. The predictions from all base learners are combined through a weighted majority vote to produce the final prediction.

Adaptation of TAO for AdaBoost. The objective function of AdaBoost (Eq. (3.1)) is a weighted misclassification error that is separable across training instances, making it directly compatible with TAO. To encourage sparsity, we augment the objective with an additional ℓ_1 regularization term on the decision node weights. The reduced problem for a decision node remains unchanged: it is still a weighted 0/1-loss binary classification task. However, the reduced problem for a leaf node $j \in \mathcal{N}_{\text{leaf}}$ must now minimize the original weighted 0/1 loss from Eq. (3.1) over its reduced set $\mathcal{R}_j$. In the case of constant-leaf models, this simply involves assigning the class label corresponding to the *weighted* majority class within its reduced set.

3.2 Experiments: comparison with baselines

3.2.1 Implementation of TAO and boosting

The TAO is implemented in C++. As an initial tree, we take an oblique complete tree of depth Δ with random weights. For solving a reduced problem in a decision (internal) node, we use an ℓ_1-regularized logistic regression solver in LIBLINEAR (v2.30 with support for instance weights) [36]. We parallelize the training of nodes at a given depth. We also implemented a Python interface for TAO. We implemented both AdaBoost.M1 and SAMME in Python.

We conduct an empirical comparison across standard machine learning benchmarks to evaluate the performance of AdaBoost with TAO-trained base learners. A diverse set of datasets was selected (see Table 3.1) from various domains, including computer vision and natural language processing, covering a wide range of number of classes (K), feature types (dense, sparse), and feature dimensions (D).

Dataset	N_{train}	N_{test}	D	K
Letter	16 000	4 000	16	26
MNIST	60 000	10 000	784	10
Char74k	66 707	7 400	64	62
R8	5 485	2 189	400	8
RCV1	15 564	518 571	47 236	53

Table 3.1: Datasets used in these experiments: number of training and test instances (N_{train}, N_{test}), number of features D, number of classes K.

We compare our method against several forest-based baselines: Random Forests (RF), Gradient Boosting (using the XGBoost implementation), and conventional AdaBoost (with the SAMME variant). For clarity, we denote AdaBoost.M1 with axis-aligned trees as M1-CART, and the SAMME variant as S-CART. In addition, we include published results from several tree ensemble variants proposed in the literature: Alternating Decision Forests (ADF) [116], shallow Neural Decision Forests (sNDF) [75], and refined Random Forests (rRF) [107]. As a point of reference, we also report the performance of a single CART tree [21]. For boosted TAO trees, S-TAO refers to the use of SAMME with sparse oblique TAO-trained trees as base learners, while M1-TAO employs the AdaBoost.M1 variant.

Tables 3.2 and 3.3 present the results, sorted by decreasing test accuracy. We begin by evaluating the case of a single TAO tree (T=1) to highlight the strength of TAO's optimization. On datasets such as MNIST and Letter, a single TAO-trained tree significantly outperforms a standard CART tree in terms of accuracy. Remarkably, a single TAO tree even matches full ensemble methods like Random Forests (RF) on certain datasets, most notably R8 and RCV1. These results suggest that TAO is capable of finding more optimal solutions to the decision tree optimization problem, making even individual trees strong learners.

We now turn to results for tree ensembles ($T > 1$). The performance of the baseline methods (RF, conventional AdaBoost, and XGBoost) aligns well with previously reported results in the literature [116, 107]. Among them, XGBoost generally achieves the best accuracy, although in some cases, such as the Char74 dataset, well-tuned Random Forests can perform comparably. It is also important

Table 3.2: Comparison of AdaBoost-TAO trees with baselines

	Forest	E_{test} (%)	size	T	Δ
MNIST	CART	12.11±0.04	6k	1	50
	TAO	5.25±0.20	24k	1	8
	RF	3.05±0.06	1M	100	46
	S-CART	2.96±0.05	6M	1k	30
	RF	2.84±0.06	10M	1k	48
	sNDF[75]	2.80±0.12	22M	80	10
	ADF[116]	2.71±0.10	(3.6M)	100	25
	XGBoost	2.67±0.00	0.3M	1k	8
	S-CART	2.28±0.02	13M	1k	16
	M1-TAO	2.09±0.04	0.8M	30	8
	rRF[107]	2.05±0.02	(160k)	100	25
	XGBoost	1.94±0.00	0.6M	10k	8
	S-TAO	1.93±0.02	0.8M	30	8
	M1-TAO	1.74±0.02	2.6M	100	8
	S-TAO	1.67±0.04	2.6M	100	8

	Forest	E_{test} (%)	size	T	Δ
Letter	CART	13.06±0.15	3k	1	27
	TAO	9.59±0.31	10k	1	11
	XGBoost	4.30±0.00	0.4M	2.6k	10
	RF	3.77±0.06	0.4M	100	34
	ADF[116]	3.52±0.12	(1M)	100	25
	RF	3.44±0.09	4.2M	1k	36
	XGBoost	3.35±0.00	0.8M	26k	6
	S-CART	2.83±0.15	0.7M	100	16
	rRF[107]	2.98±0.15	(180k)	100	25
	sNDF[75]	2.92±0.17	2.4M	70	10
	S-CART	2.58±0.09	6.7M	1k	16
	M1-TAO	1.85±0.09	0.2M	30	11
	S-TAO	1.79±0.07	0.2M	30	11
	M1-TAO	1.40±0.09	0.6M	100	11
	S-TAO	1.38±0.03	0.6M	100	11

Table 3.3: As Table 3.2, but for different datasets

	Forest	E_{test} (%)	size	T	Δ
R8	TAO	6.64±1.04	3k	1	7
	RF	6.16±0.35	93k	100	27
	S-CART	5.85±0.07	61k	100	20
	RF	5.57±0.56	0.9M	1k	27
	XGBoost	5.34±0.00	51k	800	6
	S-CART	5.11±0.09	0.6M	1k	20
	XGBoost	4.89±0.00	83k	8k	6
	S-TAO	3.12±0.10	0.6M	100	7
	M1-TAO	3.06±0.14	0.6M	100	7
RCV1	S-CART	out of time		1k	16
	RF	19.84±0.42	1M	100	233
	RF	18.78±0.37	10M	1k	233
	TAO	17.96±0.03	3M	1	12
	S-CART	16.81±0.38	0.4M	100	100
	XGBoost	13.97±0.00	0.3M	5.1k	30
	XGBoost	13.06±0.00	0.7M	51k	8
	M1-TAO	11.81±0.02	32M	100	9
	S-TAO	11.74±0.03	18M	100	9

	Forest	E_{test} (%)	size	T	Δ
Char74k	TAO	23.94±0.37	46k	1	12
	XGBoost	18.08±0.00	1.7M	6.2k	50
	S-CART	17.86±0.15	2.5M	100	60
	RF	17.33±0.14	2.5M	100	65
	S-CART	16.77±0.08	14M	1k	16
	XGBoost	16.70±0.00	6M	62k	12
	ADF[116]	16.67±0.21	(4M)	100	25
	RF	16.61±0.14	26M	1k	65
	sNDF[75]	16.04±0.20	58M	200	12
	rRF[107]	15.40±0.10	(1.1M)	100	25
	S-TAO	13.14±0.19	3.6M	100	12
	M1-TAO	12.81±0.11	4.7M	100	12

to note that XGBoost and other gradient boosting frameworks typically add K trees per iteration (one for each class), resulting in significantly larger ensemble sizes [39]. Focusing now on boosted TAO trees, we observe that both variants, M1-TAO and S-TAO, consistently outperform all baselines, including RF, conventional AdaBoost, XGBoost, and previously proposed methods from the literature. This improvement is especially pronounced on the Letter and R8 datasets, where the performance gap is significant. Moreover, boosted TAO ensembles tend to require fewer trees and shallower depths to achieve these results. While internal node in an oblique tree contains $D + 1$ parameters (a weight vector and a bias) due to the use of linear splits (as opposed to only two parameters of feature index and threshold in axis-aligned trees), we apply an ℓ_1-regularization penalty during training to promote sparsity in the node parameters. As a result, the overall number of effective parameters is often reduced, and as shown in the tables, boosted TAO models are typically smaller.

3.3 Experiments: Analysis

The strong performance of our proposed boosting method motivates a deeper analysis. In this section, we investigate the impact of using stronger base learners compared to weaker ones, assess the robustness of TAO-boosted trees to overfitting, and analyze the effect of key hyperparameters.

3.3.1 TAO vs CART trees as base learners

The top row of Fig. 3.2 illustrates the effect of replacing CART trees with TAO trees as base learners in AdaBoost.M1 and SAMME. In the left column, the number of trees is capped at 10^4, as a result, Random Forest (RF) terminates early on the Letter and MNIST datasets due to this limit. Across the three datasets shown, AdaBoost with TAO trees produce a substantial reduction in test error compared to CART trees. Importantly, hyperparameters for both CART and TAO models are selected via cross-validation, and we observe that TAO generally prefers shallower trees than CART. The top row of fig. 3.2 also shows the results of

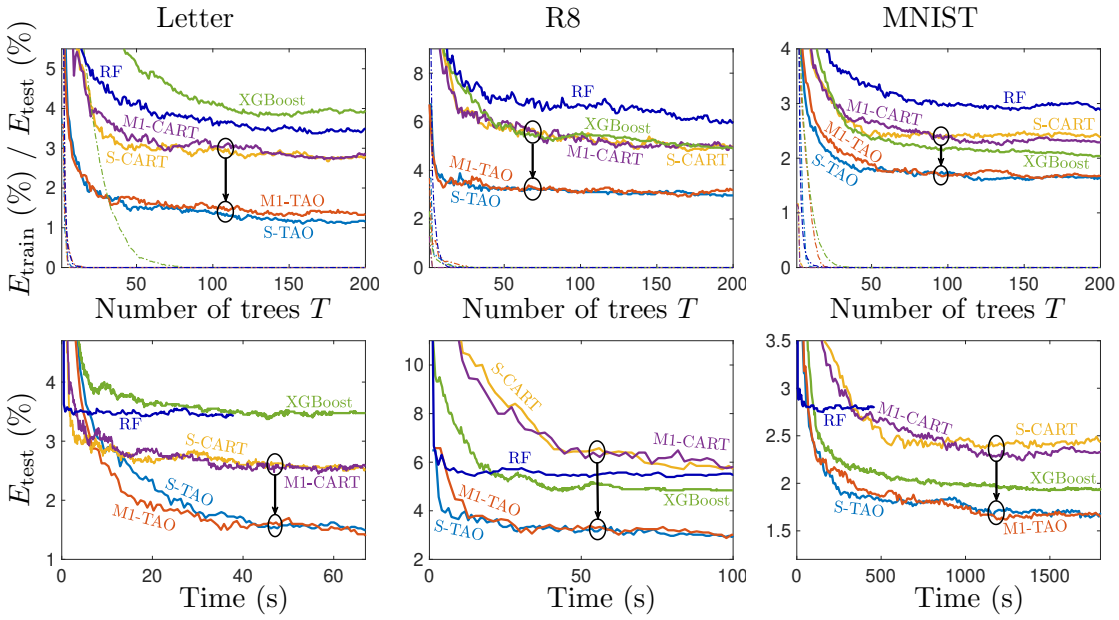

Figure 3.2: Comparison between different tree ensembles as a function of the number of trees T (top) and training time (bottom). Solid lines—test errors, dashed lines—train errors.

two different tree ensembles: Random Forests and gradient boosting (XGBoost). Clearly, for the fixed T, boosted TAO trees achieve considerably better test error than all the other forest methods. This again demonstrates the benefit of better optimization of base learners in ensemble models.

3.3.2 Comparison of runtime and number of trees

One could argue that because of the iterative nature of TAO and the sequential learning of boosting, the time to train T number of TAO trees will be much slower than inducing T number of CART trees. Though TAO is in general slower than CART, it has the flexibility to control the tradeoff between training time and optimization in the number of TAO iterations I. By setting I to a lower value, we compare boosted TAO trees against different tree ensembles as a function of training time (bottom row of fig. 3.2). All methods, with the exception of the "*-CART" variants, use parallel training with 8 threads. Even with smaller I ($I =$ 5 for Letter and MNIST, $I = 10$ for R8), boosted TAO trees achieve significantly

lower test error than other forest methods almost from the start or in a reasonably short time. This is because TAO needs fewer boosting steps to reach better test error than the other tree ensembles.

3.3.3 Hyperparameter search

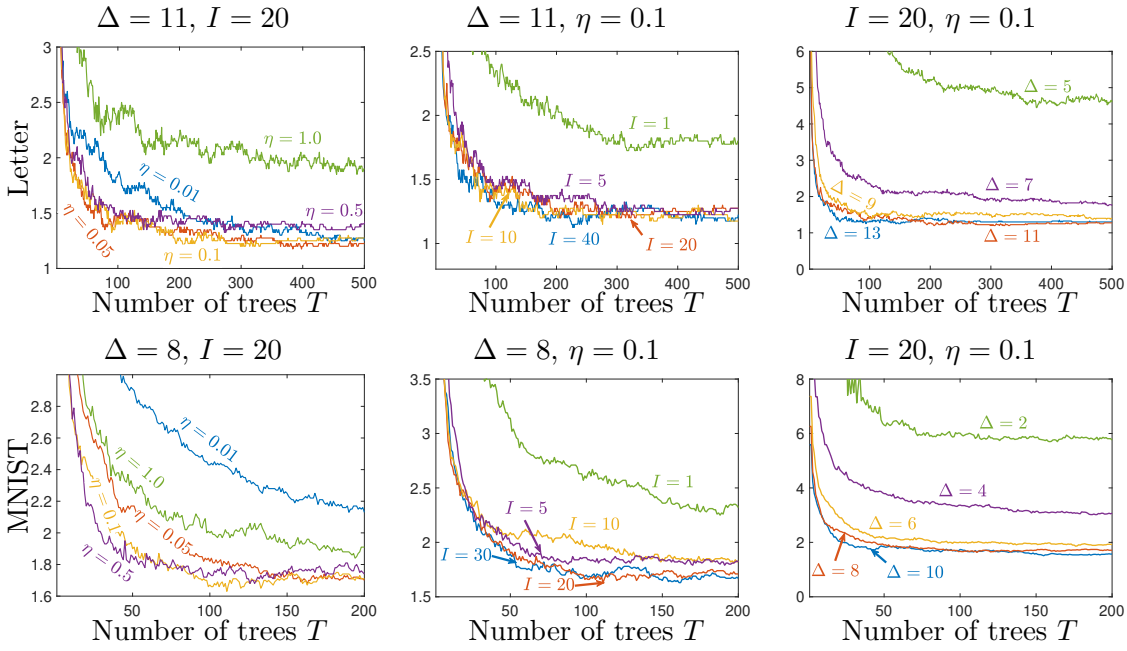

Figure 3.3: Test errors of S-TAO on Letter (top) and MNIST (bottom) as a function of 4 hyperparameters: tree depth Δ, number of TAO iterations I, shrinkage factor η and number of trees T. Each column fixes the two, and varies the other two.

Most important hyperparameters in TAO are the number of iterations I and tree depth Δ. In fig. 3.3 we explore how these two hyperparameters affect the performance of S-TAO along with the other two hyperparameters of boosting: the shrinkage factor η and the number of trees T. In the suppl. mat. we provide similar exploration for M1-TAO.

The shrinkage factor η has a significant effect on the test error. Larger values of η tend to result in noisy and less accurate ensembles, while smaller values tend to require more boosting steps T to reach the minimum test error. This is in accordance with the empirical findings in statistical literature. For the default

shrinkage factor we use $\eta = 0.1$.

Comparison of different I suggests that more TAO iterations result in more accurate ensembles. This is expected, because higher values of I correspond to better optimization. Comparison of different Δ indicate that deeper trees produce better models, though overly deep trees are more likely to overfit.

3.3.4 Does overfitting occur?

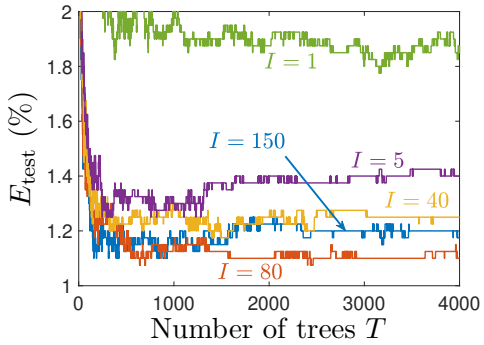

Figure 3.4: SAMME-TAO with large number of trees to detect whether overfitting happens on Letter dataset with different number of TAO iterations (I).

Traditional statistical wisdom states that overfitting is known to occur if large enough model is fit well to the training set. In case with AdaBoost, the situation is different. Owing to the shrinkage (i.e. learning rate) parameter $\eta < 1.0$, the overfitting rarely occurs in practice, and if it does, its effect is benign. In fig. 3.4 we explore AdaBoost with TAO trees using higher values of TAO iterations I and number of trees T for the Letter dataset (with a relatively large depth $\Delta = 11$). We see that overfitting eventually occurs, for example the test error curve for $I = 150$ touches bottom at around $T = 150$ and then increases (surprisingly, this also happens for $I = 5$). However, the increase is small (from 1.1% to 1.2%) and all curves continue to oscillate slightly as T increases. While hard to get, overfitting does occur, but in the form of oscillations around the minimum test error value, rather than a steady error increase.

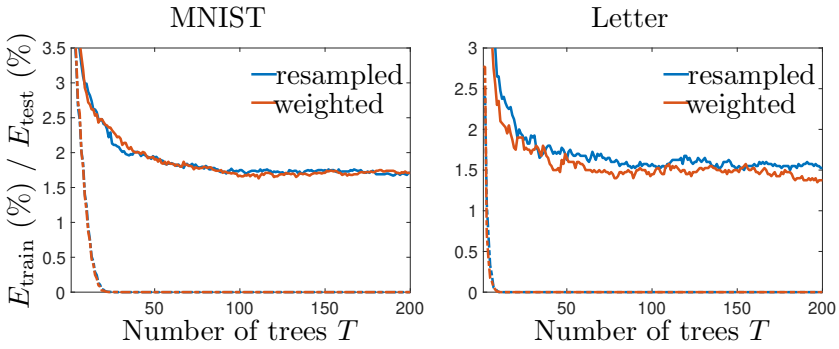

Figure 3.5: Different approaches to solve weighted classification problem for trees (eq. (3.1)) on MNIST and Letter. Solid lines—test err., dashed lines—train err.

3.3.5 Weighting or resampling?

Each boosting step involves training a tree to minimize *weighted* misclassification error (eq. (3.1)) and depending on how tree optimization works, some algorithms cannot directly handle such losses. Therefore, some implementations use so called "resampling" technique as an approximation which samples N instances with replacement and with probabilities assigned for each data point. $w_1, w_2, ..., w_n$ play role of that probabilities and indeed, they can be a good approximation since each weight is interpreted as an "importance" of that instance at the current boosting step. Moreover, $\sum_n w_n = 1$. On the other hand, our oblique trees can directly handle weighted losses and thus, we do not need any "resampling". But we still investigate these two approaches applied to our boosting algorithm and present our findings in fig. 3.5. Results show that directly solving the optimization problem gives better performance. However, sometimes the difference might be marginal as it can be observed on MNIST.

Chapter 4

Gradient Boosting with globally optimized sparse oblique trees

The previous chapter demonstrated that AdaBoost can benefit significantly from TAO-optimized base learners, both in terms of accuracy and model size. While AdaBoost has a long and influential history, another boosting algorithm, Gradient Boosting (GB), has become the dominant approach in recent years. GB provides a more general framework for greedily learning additive models with respect to any differentiable loss function. Thanks to its flexibility, GB has been successfully applied to a wide range of tasks, including regression, binary and multiclass classification, ranking, and density estimation. In contrast, AdaBoost is primarily designed for classification problems.

However, just like AdaBoost, widely used and highly optimized implementations of GB, such as XGBoost, LightGBM, and CatBoost, continue to rely on traditional, greedily induced axis-aligned decision trees as base learners. In this chapter, we demonstrate how the use of more powerful, oblique decision trees, trained via alternating optimization, improves the performance of gradient boosting. This chapter is based on [43], and more details can be found there.

4.1 Overview of Gradient Boosting (GB)

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

$$\mathbf{F}(\mathbf{x}) = \sum_{t=1}^{T} \boldsymbol{\tau}_t(\mathbf{x}) \tag{4.1}$$

where $\boldsymbol{\tau}(\mathbf{x})$ can in general be any base learner, but in this dissertation we focus on oblique 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:

$$\min_{\boldsymbol{\tau}_1,\ldots,\boldsymbol{\tau}_T} \sum_{n=1}^{N} L\big(\mathbf{y}_n, \sum_{t=1}^{T} \boldsymbol{\tau}_t(\mathbf{x}_n)\big). \tag{4.2}$$

Because this is a difficult optimization problem with nondifferentiable base learners $\boldsymbol{\tau}(\mathbf{x})$, GB resorts to greedy forward stagewise modeling (in the next chapter, we show how we directly optimize eq. (4.2) without using this greedy approach). Starting with some initial prediction $\mathbf{F}_0(\mathbf{x})$, at each step t we want to add a new tree to minimize the global loss:

$$\min_{\boldsymbol{\tau}_t} \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{F}_{t-1}(\mathbf{x}_n) + \boldsymbol{\tau}_t(\mathbf{x}_n)) \tag{4.3}$$

input training set; twice diff. loss function $L(\mathbf{y}, \hat{\mathbf{y}})$;
number of boosting steps T; learning rate η
$\mathbf{F}_0(\mathbf{x}) = \arg\min_{\boldsymbol{\rho}} \sum_{n=1}^{N} L(\mathbf{y}_n, \boldsymbol{\rho})$
for $t = 1$ **to** T
  $\mathbf{g}_n = \frac{\partial L(\mathbf{y}_n, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}\Big|_{\hat{\mathbf{y}}=\mathbf{F}_{t-1}(\mathbf{x}_n)}, n = 1, \ldots, N$
  $\mathbf{H}_n = \frac{\partial^2 L(\mathbf{y}_n, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}^2}\Big|_{\hat{\mathbf{y}}=\mathbf{F}_{t-1}(\mathbf{x}_n)}, n = 1, \ldots, N$
  Train a base learner $\boldsymbol{\tau}_t$ to minimize eq. (4.5)
  $\mathbf{F}_t(\mathbf{x}_n) = \mathbf{F}_{t-1}(\mathbf{x}_n) + \eta\boldsymbol{\tau}_t(\mathbf{x}_n), n = 1, \ldots, N$
return $\mathbf{F}(\cdot) = \mathbf{F}_0(\cdot) + \sum_{t=1}^{T} \eta\boldsymbol{\tau}_t(\cdot)$

Figure 4.1: Pseudocode of Gradient Boosting (GB).

Following the established approach in [38, 28], we perform a second order Taylor approximation:

$$\min_{\boldsymbol{\tau}_t} \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{F}_{t-1}(\mathbf{x}_n)) + \mathbf{g}_n^T \hat{\mathbf{y}}_n + \frac{1}{2} \hat{\mathbf{y}}_n^T \mathbf{H}_n \hat{\mathbf{y}}_n \tag{4.4}$$

where $\hat{\mathbf{y}}_n = \boldsymbol{\tau}_t(\mathbf{x}_n)$, 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. 4.1). Ignoring the constant term in eq. (4.4), we obtain the following objective function of a base learner at boosting step t:

$$\min_{\boldsymbol{\tau}_t} \sum_{n=1}^{N} \mathbf{g}_n^T \boldsymbol{\tau}_t(\mathbf{x}_n) + \frac{1}{2} \boldsymbol{\tau}_t(\mathbf{x}_n)^T \mathbf{H}_n \boldsymbol{\tau}_t(\mathbf{x}_n) \tag{4.5}$$

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 $\boldsymbol{\tau}_t(\mathbf{x})$ is trained at boosting step t, 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 T or one might early stop based on the validation metric. The pseudocode in fig. 4.1 summarizes the GB algorithm.

Adaptation of TAO for Gradient Boosting. As described above, Gradient Boosting defines a specific, somewhat unusual at first glance, loss function to optimize for each new tree added to the ensemble (Eq. (4.5)). Similar to the AdaBoost objective, this loss is expressed as a separable sum over individual training instances, making it directly compatible with the TAO algorithm. To promote sparsity, we again include an ℓ_1-regularization term on the decision node weights, controlled by a hyperparameter λ. The reduced problem at each decision node remains unchanged: it is a weighted 0/1-loss binary classification task over oblique hyperplane parameters. From an implementation perspective, the only modification required at decision nodes is the computation of the losses incurred when routing an instance to the left or right subtree, used in constructing the reduced problem. This computation is handled at the leaf level, as leaves are responsible for producing the final predictions of the tree.

As regards with a reduced problem over a leaf $j \in \mathcal{N}_{\text{leaf}}$ with a reduced set $\mathcal{R}_j$, it takes this form:

$$\min_{\boldsymbol{\theta}_j} \sum_{n \in \mathcal{R}_j} \mathbf{g}_n^T \boldsymbol{\theta}_j + \frac{1}{2} \boldsymbol{\theta}_j^T \mathbf{H}_n \boldsymbol{\theta}_j. \tag{4.6}$$

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

4.2 Experiments

Implementation We implement everything in C++ (having a Python interface makes debugging difficult). For all the experiments, we start with a complete tree of depth Δ and random initial parameters (Gaussian (0,1)). We parallelize the optimization over the nodes at a given depth using OpenMP. When solving a reduced problem at a decision node, we use an ℓ_1 regularized logistic regression in LIBLINEAR solver of version 2.43. λ is a hyperparameter in GB TAO so ideally it should be cross-validated. To save training time, instead of cross-validating it for the whole forest, we cross-validate λ only for a single tree. This simple choice already results in leading performance in our experiments.

Experiment settings We compare GB oblique trees trained with TAO against established state-of-the-art implementations of GB: XGBoost [28] and LightGBM [71]. For reference, we also compare with a scikit-learn [100] implementation of GB, which closely follows the original formulation[39]. We could not find any existing implementation of GB oblique trees, but we include SPORF [128] which uses bagging to construct an ensemble of sparse oblique trees. We additionally cite the published results of GB with piecewise linear trees, GBDT-PL [118].

In order to compare models of different size, for each comparison baseline we choose and fix some number of boosting steps T (or number of trees T in SPORF), and for the given 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 λ 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 $I = 30$.

4.2.1 Classification tasks

Image classification Table 4.1 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 T=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 demonstrates the superiority of an oblique tree over an axis-aligned one for these image datasets.

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 [81, 120]. To analyze how GB oblique trees perform with these types of problems, in Table 4.2 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

	Forest	E_{test} (%)	#pars.	T	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
	SPORF	2.89±0.04	-	1k	1	50	5k
	GB-TAO	2.33±0.00	200k	10	1	10	209
	XGBoost	2.20±0.00	107k	100	10	6	36
	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 4.1: Comparison of GB-TAO with baselines for classification, sorted by decreasing test error. T refers to the number of boosting steps, k is the number of trees used at each boosting step. The total number of trees is Tk. Δ is the max depth of the forest.

	Forest	E_{test} (%)	#pars.	T	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 4.2: Similar to Table 4.1, but for sparse high-dimensional document classification datasets

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.

4.2.2 Regression tasks

Table 4.3 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

	Forest	E_{test}	#pars.	T	Δ
cpuact (5k,21)	**GB-TAO**	2.67±0.03	440	1	34
	XGBoost	2.60±0.00	60k	100	201
	XGBoost	2.51±0.00	42k	1k	15
	GB-sklearn	2.51±0.06	14k	100	49
	GB-sklearn	2.41±0.04	43k	1k	15
	GB-TAO	2.42±0.02	17k	30	46
	LightGBM	2.27±0.00	19k	100	64
	LightGBM	2.25±0.00	91k	1k	31
	GB-TAO	2.23±0.02	31k	50	48
CT-slice (43k,384)	LightGBM	1.53±0.00	153k	100	512
	LightGBM	1.52±0.00	91k	1k	31
	GB-sklearn	1.43±0.02	192k	100	641
	XGBoost	1.50±0.00	107k	100	357
	GB-TAO	1.28±0.02	28k	1	223
	GB-sklearn	1.26±0.03	900k	1k	301
	XGBoost	1.26±0.00	767k	1k	256
	GBDT-PL	1.24±0.00	-	-	-
	GB-TAO	0.90±0.02	81k	30	16
	GB-TAO	0.45±0.01	1.2M	100	64
year (450k,90)	**GB-TAO**	9.17±0.01	19k	1	252
	XGBoost	9.05±0.00	153k	100	511
	LightGBM	9.03±0.00	153k	100	512
	GB-sklearn	9.03±0.02	248k	100	827
	GB-sklearn	8.96±0.02	171k	1k	58
	LightGBM	8.92±0.00	1.5M	1k	512
	XGBoost	8.91±0.00	1.8M	1k	608
	GB-TAO	8.88±0.02	78k	20	62
	GB-TAO	8.73±0.01	402k	100	63

	Forest	E_{test}	#pars.	T	Δ
casp (45k,9)	**GB-TAO**	4.38±0.03	4k	1	430
	XGBoost	3.66±0.00	119k	100	397
	GB-sklearn	3.65±0.02	727k	100	2424
	XGBoost	3.58±0.00	793k	1k	265
	GB-sklearn	3.58±0.01	854k	1k	285
	LightGBM	3.54±0.00	153k	100	512
	GB-TAO	3.49±0.01	256k	50	645
	LightGBM	3.48±0.00	766k	1k	256
	GBDT-PL	3.46±0.00	-	-	-
	GB-TAO	3.43±0.00	481k	100	603
	GB-TAO	3.39±0.01	887k	200	552
superconduct (17k,81)	**GB-TAO**	11.02±0.10	8k	1	466
	GB-sklearn	9.38±0.01	139k	1k	47
	XGBoost	9.20±0.00	130k	1k	44
	GB-sklearn	9.14±0.03	128k	100	430
	XGBoost	8.98±0.00	132k	100	441
	GBDT-PL	8.80±0.00	-	-	-
	LightGBM	8.77±0.00	38k	100	128
	GB-TAO	8.76±0.02	573k	50	216
	LightGBM	8.73±0.00	190k	1k	64
	GB-TAO	8.68±0.02	1M	100	218

Table 4.3: Similar to Table 4.1, 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.

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.

Examining the overall performance of forests in Table 4.3, 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 [118], 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.

4.2.3 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 T. In the left column of fig. 4.2 we compare how the test error changes for classification datasets as a function GB steps T. The plots clearly show that stronger TAO oblique trees require fewer steps T 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 T 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. 4.2

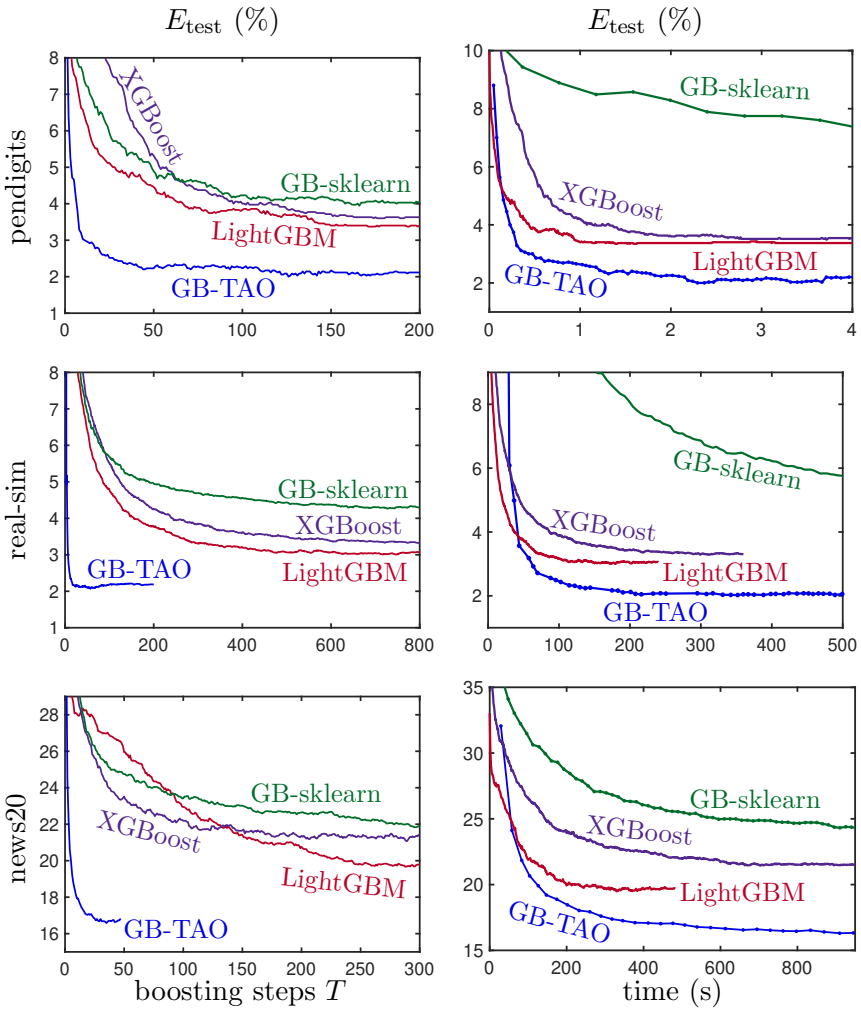

Figure 4.2: Comparison of different GB forests as a function of the number of boosting steps T (left column) and time (right column).

(all methods except GB-sklearn are trained using parallel processing on a shared memory system with 8 threads), we show that by limiting the number of TAO iterations to I=3, we can obtain accurate GB-TAO forests in a comparable time.

4.2.4 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

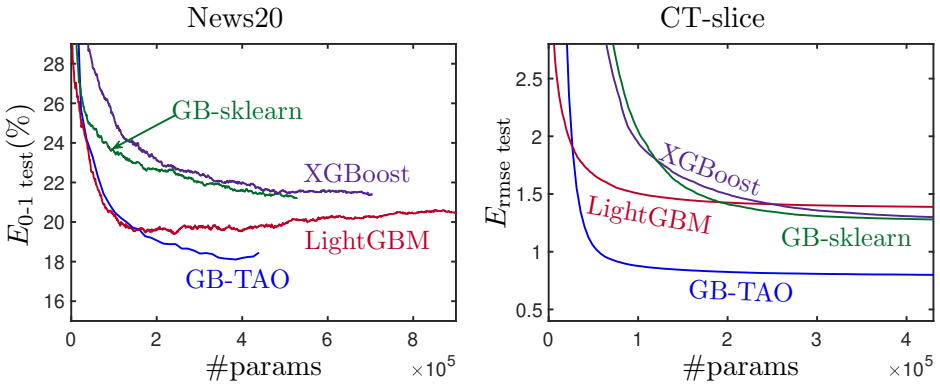

Figure 4.3: Test error as a function of the number of parameters

forests of comparable number of parameters. Fig. 4.3 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.

4.3 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.

Chapter 5

Forest Alternating Optimization

The previous two chapters showed that simply replacing the base learners in AdaBoost and Gradient Boosting with more powerful, TAO-optimized trees leads to more accurate ensembles that require fewer boosting iterations to achieve competitive performance. However, both AdaBoost and Gradient Boosting are fundamentally *greedy algorithms*, they construct the ensemble by sequentially adding base learners that minimize the current residuals, without revisiting or updating the previously added models. In fact, traditional implementations of these methods can be viewed as introducing an additional layer of greediness by relying on CART-style recursive partitioning to train the individual trees. The preceding chapters showed that removing this inner-level greediness, through alternating optimization, improves model accuracy.

This naturally raises the question: can we also address the outer-level greediness of boosting? Rather than following the forward stagewise additive modeling approach, can we directly define and jointly optimize a fixed-size decision forest by iteratively refining each tree's parameters? This chapter explores exactly that question, introducing an alternating optimization algorithm designed to jointly learn all trees in a decision forest, without relying on greedy stagewise procedures. The resulting algorithm, Forest Alternating Optimization (FAO), achieves state-of-the-art results in learning accurate decision forests.

5.1 Forest Alternating Optimization (FAO)

We consider a decision forest $\mathbf{F}(\mathbf{x};\boldsymbol{\Theta})$ of T trees $\{\boldsymbol{\tau}_t(\mathbf{x};\boldsymbol{\Theta}_t)\}_{t=1}^T$ (oblique, with constant leaves), where the forest output is defined by the sum of individual tree outputs:

$$\mathbf{F}(\mathbf{x};\boldsymbol{\Theta}) = \sum_{t=1}^{T} \boldsymbol{\tau}_t(\mathbf{x};\boldsymbol{\Theta}_t). \tag{5.1}$$

For 1D regression, $F(\mathbf{x};\boldsymbol{\Theta}) \in \mathbb{R}$ and is the final model prediction. For binary classification, $F(\mathbf{x};\boldsymbol{\Theta}) \in \mathbb{R}$ is a logit that will be passed through a logistic function to output a probability. For $K > 2$ class classification, $\mathbf{F}(\mathbf{x};\boldsymbol{\Theta}) \in \mathbb{R}^K$ is a vector of logits that will be passed through a softmax to output class probabilities. We do not consider a coefficient for each tree (e.g. the the learning rate η in Gradient Boosting) because it can be absorbed within each $\boldsymbol{\tau}_t$ (more precisely, within each leaf label).

Given a training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of (instance,label) pairs, we want to learn a forest by optimizing the following objective function:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(y_n, \mathbf{F}(\mathbf{x}_n;\boldsymbol{\Theta})) + \lambda \sum_{t=1}^{T} \sum_{i \in \mathcal{N}_{\text{dec}}^t} \|\mathbf{w}_{ti}\|_1 + \mu \sum_{t=1}^{T} \sum_{j \in \mathcal{N}_{\text{leaf}}^t} \|\boldsymbol{\theta}_{tj}\|_2^2. \tag{5.2}$$

This formulation consists of a loss function that is additive over training instances, along with regularization terms on the parameters of the decision nodes and leaves across all trees with hyperparameters $\lambda, \mu \geq 0$. Note that we also include ℓ_2-regularization on the leaf parameters, which is not present in the objective function for a single tree of eq. (2.1). As will become clear later, this regularization helps mitigate overfitting. The parameters $\boldsymbol{\Theta}_t$ of tree t include the decision node parameters $\{\mathbf{w}_{ti}, w_{ti0}\}_{i \in \mathcal{N}_{\text{dec}}^t}$ and the leaf labels $\{\boldsymbol{\theta}_{tj}\}_{j \in \mathcal{N}_{\text{leaf}}^t}$. For simplicity, we assume each tree has the same structure, specifically, complete of depth Δ. This formulation of forest learning follows the standard approach used in most machine learning models, such as SVMs or neural networks, by casting learning as regularized empirical risk minimization. In contrast, traditional methods for training forests (e.g., Random Forests, AdaBoost, and Gradient Boosting) typically use greedy procedures that sequentially add one tree at a time, freezing the parameters of previously

added trees. Here, by contrast, we define learning as a joint optimization problem over all parameters of a well-defined parametric model, including the loss and regularization terms. The hyperparameters of the forest (number of trees T, depth Δ, regularization λ, μ) can be determined by standard model selection techniques such as cross validation.

We now describe how to optimize Eq. (5.2). Since the forest function $\mathbf{F}$ is non-differentiable, gradient-based methods cannot be applied directly. Instead, we use alternating optimization, which we implement in the following two forms.

Alternating optimization over individual trees In this type of step, we optimize one tree at a time given the remaining trees are fixed. The objective function (5.2) over a single tree $\boldsymbol{\tau}_t(\mathbf{x}; \boldsymbol{\Theta}_t)$ is the following:

$$\begin{aligned}\min_{\boldsymbol{\Theta}_t} E(\cdots \boldsymbol{\Theta}_t \cdots) = \sum_{n=1}^{N} L\left(y_n, \sum_{t=1}^{T} \boldsymbol{\tau}_t(\mathbf{x}_n; \boldsymbol{\Theta}_t)\right) \\ + \lambda \sum_{t=1}^{T} \sum_{i \in \mathcal{N}^t_{\text{dec}}} \|\mathbf{w}_{ti}\|_1 + \mu \sum_{t=1}^{T} \sum_{j \in \mathcal{N}^t_{\text{leaf}}} \|\boldsymbol{\theta}_{tj}\|_2^2. \Leftrightarrow\end{aligned} \tag{5.3}$$

$$\begin{aligned}\min_{\boldsymbol{\Theta}_t} E(\boldsymbol{\Theta}_t) = \sum_{n=1}^{N} L\left(y_n, \boldsymbol{\tau}_t(\mathbf{x}_n; \boldsymbol{\Theta}_t) + \sum_{t' \neq t} \boldsymbol{\tau}_{t'}(\mathbf{x}; \boldsymbol{\Theta}_{t'})\right) \\ + \lambda \sum_{i \in \mathcal{N}^t_{\text{dec}}} \|\mathbf{w}_{ti}\|_1 + \mu \sum_{j \in \mathcal{N}^t_{\text{leaf}}} \|\boldsymbol{\theta}_{tj}\|_2^2.\end{aligned} \tag{5.4}$$

Because regularization terms are separable over individual trees, only the ones related to the current $\boldsymbol{\tau}_t(\mathbf{x}; \boldsymbol{\Theta}_t)$ remains. However, the loss function $L(\cdot, \cdot)$ for the individual tree, at first glance, takes a more complex form. Because the other trees are fixed, $\sum_{t' \neq t} \boldsymbol{\tau}_{t'}(\mathbf{x}; \boldsymbol{\Theta}_{t'})$ is constant with respect to the current tree. The meaning of this objective function over the individual tree $\boldsymbol{\tau}_t(\mathbf{x}; \boldsymbol{\Theta}_t)$ is clear: by adding its contribution to the current forest prediction we want to minimize the original loss. This objective function is still amenable to the TAO algorithm:

Decision node The reduced problem over a decision node does not change. It still takes the form of a weighted 0/1 binary classification problem over an oblique hyperplane weights.

Leaf node The reduced problem of a leaf $j \in \mathcal{N}^t_{\text{leaf}}$ of a tree $\boldsymbol{\tau}_t(\mathbf{x}; \boldsymbol{\Theta}_t)$ with its reduced set $\mathcal{R}_{tj}$ takes the following form:

$$\min_{\boldsymbol{\theta}_{tj}} \; E(\boldsymbol{\theta}_{tj}) = \sum_{n \in \mathcal{R}_{tj}} L\left(y_n, \boldsymbol{\theta}_{tj} + \sum_{t' \neq t} \boldsymbol{\tau}_{t'}(\mathbf{x}_n; \boldsymbol{\Theta}_{t'})\right) + \mu \left\|\boldsymbol{\theta}_{tj}\right\|_2^2. \tag{5.5}$$

For regression problems, with squared error, the solution is simply the average of residuals. For the cross entropy loss of classification, there is no closed form solution though the problem is convex. In our implementation for classification, we skip the optimization step over the individual leaves, and instead use only the joint optimization over all the leaf parameters, presented next.

Alternating optimization over all leaves In this type of step, we optimize over the labels of *all* leaves of *all* trees given the decision nodes' parameters are fixed. First, write the predictive function of a tree in the following form:

$$\begin{aligned} &\boldsymbol{\tau}\left(\mathbf{x}; \{\mathbf{w}_i, w_{i0}\}_{i \in \mathcal{N}_{\text{dec}}}, \{\boldsymbol{\theta}_j\}_{j \in \mathcal{N}_{\text{leaf}}}\right) = \sum_{j \in \mathcal{N}_{\text{leaf}}} \boldsymbol{\theta}_j \, \varphi_j(\mathbf{x}), \\ &\text{with} \quad \varphi_j(\mathbf{x}) = I\left[\rho\left(\mathbf{x}; \{\mathbf{w}_i, w_{i0}\}_{i \in \mathcal{N}_{\text{dec}}}\right) = j\right] \end{aligned} \tag{5.6}$$

where I is an indicator function and ρ: $\mathbb{R}^D \rightarrow \mathcal{N}_{\text{leaf}}$ is the routing function of the tree, which maps an input instance to the leaf it reaches under the tree. In other words, $\varphi_j(\mathbf{x})$ is 1 in the input space region of leaf j and 0 elsewhere. Note ρ depends only on the decision nodes' parameters. In the above form, the tree can be seen as a sum of nonlinear basis functions $\{\varphi_j\}_{j \in \mathcal{N}_{\text{leaf}}}$ (whose support is given by the decision nodes) with coefficients $\{\boldsymbol{\theta}_j\}_{j \in \mathcal{N}_{\text{leaf}}}$ given by the leaf labels. With that notation, we have that optimizing (5.2) over all the leaf labels is equivalent to:

$$\min_{\{\{\boldsymbol{\theta}_{tj}\}_{j \in \mathcal{N}^t_{\text{leaf}}}\}_{t=1}^T} \; \sum_{n=1}^N L\left(\mathbf{y}_n, \sum_{t=1}^T \sum_{j \in \mathcal{N}^t_{\text{leaf}}} \boldsymbol{\theta}_{tj} \, \varphi_{tj}(\mathbf{x}_n)\right) + \mu \sum_{t=1}^T \sum_{j \in \mathcal{N}^t_{\text{leaf}}} \left\|\boldsymbol{\theta}_{tj}\right\|_2^2. \tag{5.7}$$

Since φ_{tj} does not depend on any leaf labels and thus constant with respect to this problem, the second argument of the loss L is a linear function of the leaf labels. Hence, eq. (5.7) is a ridge regression problem or ℓ_2-regularized logistic regression which can be solved jointly over all the leaf labels of all the trees exactly

with efficient algorithms. Thus, this step also monotonically decreases the overall objective function (5.2).

Note that optimizing over all the leaves of a single tree is fully separable across leaves, due to the separability condition in TAO. In contrast, jointly optimizing the leaves across all trees in the forest is not separable. Nonetheless, as noted earlier, this optimization can still be performed effectively and often leads to substantial progress, particularly since leaves account for half of all nodes in a decision forest.

5.1.1 Overall FAO algorithm

input training set;
initial forest $\mathbf{F}(\cdot; \boldsymbol{\Theta})$ of T trees
number of FAO iterations FI
repeat
  Optimize all leaves jointly by solving
  a regularized linear regression problem
  for $t = 1$ **to** T
    Optimize decision nodes of t's tree $\boldsymbol{\tau}_t(\cdot; \boldsymbol{\Theta}_t)$ using TAO
until FI iterations
return F

Figure 5.1: Pseudocode of FAO for regression.

At each iteration we optimize over tree $t = 1, \dots, T$ (using TAO) and then optimize over all leaves jointly. One such optimization path defines one FAO iteration. We experimentally observe the algorithm converges in about 20 FAO iterations. The initial parameters are just as in TAO: random for each node of each tree.

Averaging multiple FAO forests As noted in section 5.2, FAO is very effective at optimizing equation (5.2) over a training set, so much so that it can easily

overfit. While this can be corrected to some extent by model selection (cross-validation over the forest hyperparameters), we find we obtain better generalization by averaging several FAO forests. Specifically, we train Q forests $\mathbf{F}_1(\mathbf{x}), \dots, \mathbf{F}_Q(\mathbf{x})$ (each with T trees) independently on the entire training set, each with a different random initialization. The result is a forest with QT trees $\mathbf{F}(\mathbf{x}) = \frac{1}{Q}(\mathbf{F}_1(\mathbf{x}) + \cdots + \mathbf{F}_Q(\mathbf{x})) = \sum_{q=1}^{Q} \sum_{t=1}^{T} \boldsymbol{\tau}_{qt}(\mathbf{x})$ (where the $\frac{1}{Q}$ factor has been absorbed into each $\boldsymbol{\tau}_{qt}$). As discussed in the next section, this version of FAO forests achieves the most accurate forests, and we evaluate it in different datasets in section 5.3.

5.2 Optimization vs generalization, and smoothness of the forest

With the FAO algorithm, we can effectively select the number and size of trees, define a loss function on the training data, and incorporate regularization terms to efficiently find a better local optimum of (5.2). However, while this approach is powerful, it can introduce local regions of nonsmoothness, referred to as "islands", in the forest's predictive function. These nonsmooth regions can negatively impact generalization performance. We present these through illustrative examples.

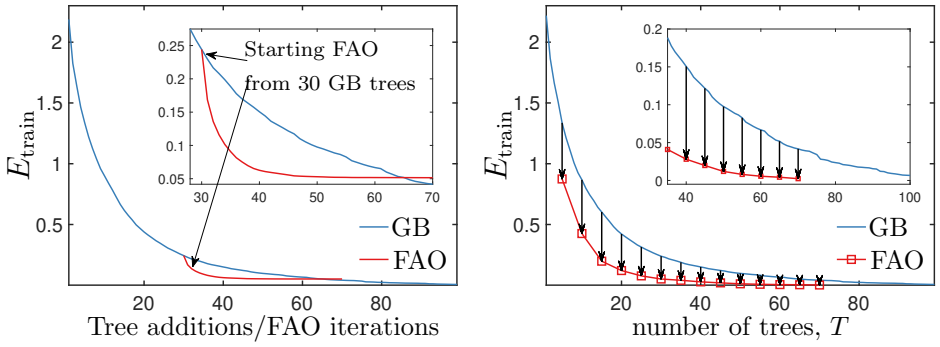

Figure 5.2: Optimization ability and overfitting problem of FAO on the cpuact dataset. The errors are RMSE. All trees are oblique and complete of depth $\Delta = 6$. *Left:* optimizing 30 trees (initialized from GB) with FAO can exceed the performance of 60 GB trees with no shrinkage. *Right:* similar behavior for a range of forest sizes.

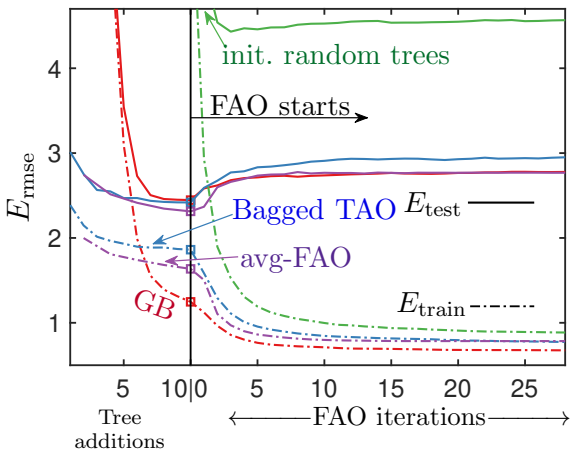

Figure 5.3: Illustration of overfitting: starting FAO with 10 trees initialized in any of various ways (from GB, bagging, averaged small FAO forests or random initialization) results in a significant increase in test error.

FAO is very effective at optimizing the training problem Fig. 5.2 shows, as a function of the number of FAO iterations, the training and test error for FAO forests under various initialization conditions: random parameters, or the first trees from a GB forest, or bagged TAO trees. *On the training set (left/middle plots), we see the objective function monotonically decreases*, achieving in very few iterations a much lower error than a GB forest of the same size. This confirms that a proper optimization can make a much better use of an available parameter budget. *However, on the test set shown in fig. 5.3, the error increases as soon as FAO starts optimizing, indicating overfitting.* While the test error does not always increase, particularly if we use a strong leaf regularization, it generally does, even with noiseless data.

Nonsmooth forests: the "islands" phenomenon The strong overfitting we saw earlier happens because forests are both extremely flexible and locally controlled models. First is flexibility. Decision trees and forests make predictions that are constant within specific regions of the input space. In a single tree, each leaf defines one such region, so a tree with L leaves has L regions. In a forest with T trees, each also with L leaves, a prediction is made by combining one leaf from each tree. The result is up to L^T regions but many may be empty because some leaf combinations do not intersect. This shows how much more flexible a forest is

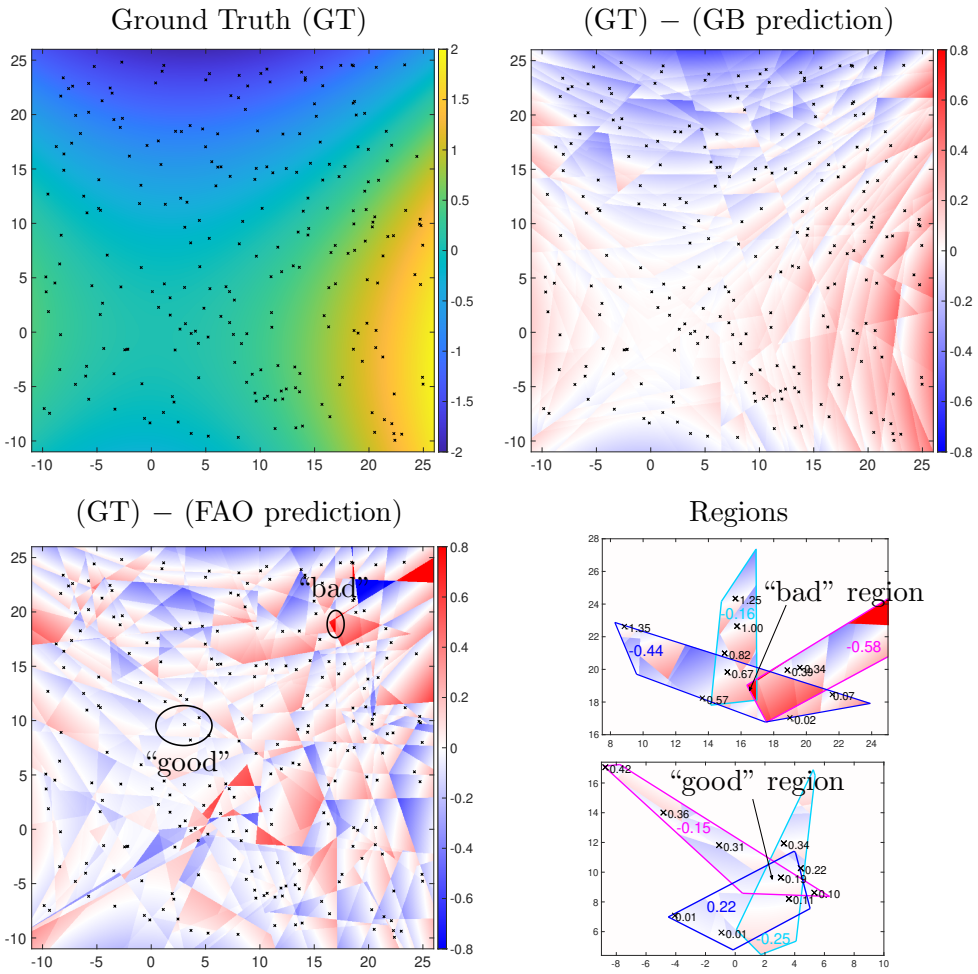

Figure 5.4: Illustrative example of the "islands phenomenon" in 2D with $T = 3$ trees of depth $\Delta = 6$ fitted on $N = 250$ points. *Top-left:* training set (black dots) and contours of the ground truth function. *Top-right:* plot of the signed error of the GB forest on the entire input space. It is smoother and has overall a lower test error. *Bottom-left:* signed error plot of the FAO forest on the entire input space. Dark-red and dark-blue regions indicate a larger error. *Bottom-right:* example of a "bad" region, i.e., an "island" of large error (top), and of a "good" region (bottom), showing the tree leaves that make up each region.

than a single tree. Even with a training set of N examples, we can only create N labeled regions with a single tree, but a forest can generate exponentially more regions using just TL leaves. Second is locality. In a tree, if you change the value of one leaf, you only affect the prediction only in that leaf's region. In a forest, a region's value depends on the values of multiple leaves (one per tree). Changing

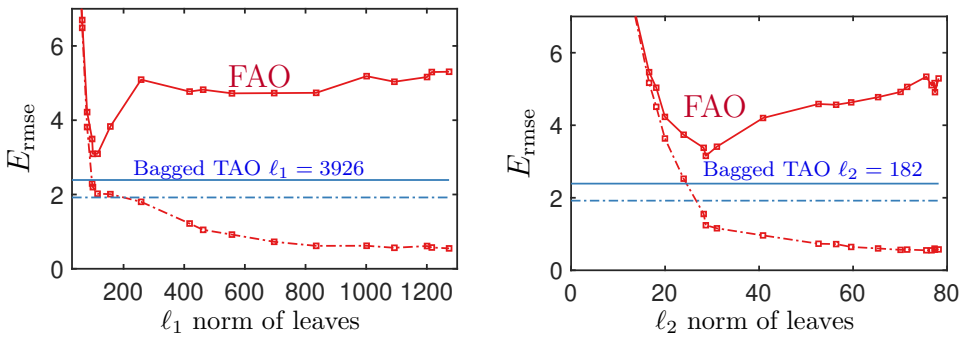

Figure 5.5: Test (solid lines) and train (dash-dotted lines) RMSE errors when training $T = 10$ trees of depth $\Delta = 6$ with FAO for different leaf regularization on the cpuact dataset.

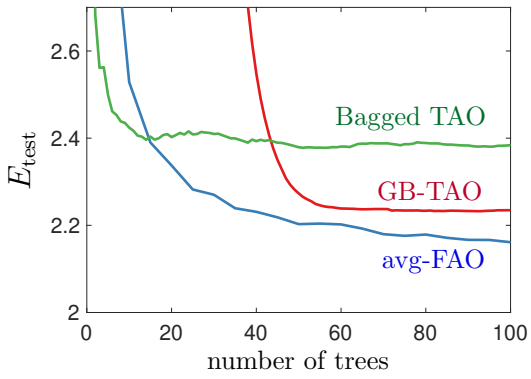

Figure 5.6: Generalization error for FAO, GB and bagging. Here, each FAO forest consists of 5 trees. Then, for FAO, 100 trees in the X axis means 20 FAO forests each consisting of 5 trees.

one of these leaves affects all the regions it touches, but the impact is still limited to those areas. This is very different from global models such as linear regression, where changing the model affects the entire input space. Figure 5.4 shows this effect in a 2D regression task with no noise. Some regions exhibit much higher error than their surroundings. This happens when "bad" parts of leaves, those far from training points, intersect. These poorly generalized regions become more prominent in the presence of outliers but also occur with clean data.

In summary, the overfitting seen with FAO is a natural consequence of the kind of functions that forests are capable of modeling. Using the proper optimization as FAO, this phenomenon becomes more pronounced. Similar overfitting behavior, especially in the presence of noise or outliers, has been observed with boosting

[33, 133]. In GB, this issue is typically managed by the shrinkage factor when adding new trees. However, this comes at the cost of requiring a much larger number of trees in the model.

Regularization helps but is not good enough As described, forests exhibit a particular kind of nonsmoothness. They are not alone in being able to overfit—many other flexible models, such as neural networks, also are. The standard way to control for this is to use a regularization term that smoothes the model and to tune its hyperparameter via cross-validation. Fig. 5.5 (left and middle) shows representative results of doing this using an ℓ_1 or ℓ_2 penalty on the label values, which encourages them to be small. Indeed, this helps to determine an optimal amount of regularization, but the resulting forests have higher error than simply bagging TAO trees. There may be other forms or regularization that work better, but here we consider a different approach, described next.

Averaged FAO forests achieve best results The above arguments show that a FAO forest is a very strong, low-bias, high-variance model. Returning to the spirit of ensemble learning, an average of such models (described in section 5.1) should remain low-bias but with lower variance, hence generalize better. This is indeed what we observe, consistently. As shown in fig. 5.5 (right), it now results in lower test error than a bagged or GB forest, while also using fewer trees.

5.3 Experiments

This section shows our experimental results on real-world data. We demonstrate that *the averaged FAO forests consistently dominate over other tree-based ensembling frameworks in accuracy while generating compact models.* In most cases, the accuracy gap is noticeably large compared to the established frameworks, such as XGBoost, which makes FAO a strong competitor with the state-of-the-art performance. To show that, we consider several regression and classification benchmarks of varying sizes and across different domains. We start with the main comparison results on machine learning benchmarks and report the training run-

	Forest	E_{test} (%)	#pars.	T	Δ
MNIST (60k,784,10)	SPORF	2.89±0.04	(143M)	1k	50
	XGBoost	2.20±0.00	107k	1k	6
	LightGBM	2.02±0.00	121k	1k	10
	XGBoost	1.91±0.00	505k	10k	6
	GB-TAO	1.65±0.02	3M	500	7
	LightGBM	1.62±0.00	642k	10k	21
	GB-TAO	1.55±0.02	7.2M	1.4k	7
	avg-FAO	1.48±0.06	658k	60	6
	avg-FAO	1.39±0.04	968k	90	6
	avg-FAO	1.33±0.04	4.9M	300	8
news20 (16k,62k,20)	SPORF	22.51±0.10	(110M)	100	569
	XGBoost	22.19±0.00	84k	2k	6
	SPORF	21.65±0.26	(1.1B)	1k	571
	XGBoost	21.39±0.00	705k	20k	6
	LightGBM	20.69±0.00	1.8M	20k	27
	LightGBM	20.01±0.00	182k	2k	28
	GB-TAO	18.76±0.01	746k	50	6
	GB-TAO	18.13±0.00	1.0M	100	6
	avg-FAO	17.45±0.21	500k	50	4
	GB-TAO	16.65±0.04	1.7M	800	4
	avg-FAO	16.59±0.13	998k	100	4

	Forest	E_{test} (%)	#pars.	T	Δ
CIFAR100 (50k,512,100)	LightGBM	30.32±0.00	1.0M	14.3k	18
	XGBoost	30.15±0.00	174k	100k	6
	GB-TAO	29.38±0.04	39k	1	12
	SPORF	28.63±0.07	26k	10	20
	SPORF	27.07±0.20	106k	100	10
	GB-TAO	26.98±0.04	1.2M	750	8
	avg-FAO	26.93±0.07	594k	50	8
	GB-TAO	26.86±0.02	2.0M	1.2k	8
	SPORF	26.71±0.05	530k	500	10
	GB-TAO	26.64±0.02	3.3M	400	6
	avg-FAO	26.55±0.12	725k	80	6
SUSY (4.5M,18)	SPORF	19.9124	(271M)	100	102
	SPORF	19.7344	(2.7B)	1k	109
	XGBoost	19.6282	151k	300	8
	XGBoost	19.6214	196k	100	10
	LightGBM	19.6164	153k	100	23
	LightGBM	19.5998	230k	300	21
	XGBoost	19.5902	2.0M	1k	10
	LightGBM	19.5748	1.5M	1k	23
	avg-FAO	19.5104	233k	50	8
	avg-FAO	19.4994	459k	100	8

Table 5.1: Comparison of FAO with other baselines for classification.

time of our algorithm. We then finalize the section by analyzing the impact of forest size and number of FAO iterations.

Setup We compare averaged FAO forests of oblique trees against GB with TAO. We include SPORF [128] forests, which also use sparse oblique trees but they are greedily induced. Additionally, we include highly optimized GB packages of axis-aligned trees: XGBoost [28] and LightGBM [71]. We tune important hyperparameters of the baselines such as the depth, number of leaves and a learning rate. For FAO, we tune the tree depth Δ and the number of trees T. We apply an ℓ_1 penalty for decision node weights, for which we find the optimal hyperparameter λ for a single tree, and use it for the whole FAO forest. We apply a small penalty on leaf weights with a fixed parameter $\mu = 0.01$: for regression we use an ℓ_2 penalty, for classification an ℓ_1 penalty. For an initial tree in the forest, we take a complete tree of depth Δ and random node parameters. More details on hyperparameters and datasets appear in the Appendix.

	Forest	E_{test}	#pars.	T	Δ
cpuact (5k,21)	XGBoost	2.60±0.00	60k	100	10
	XGBoost	2.51±0.00	42k	1k	4
	GB-TAO	2.42±0.02	18k	30	6
	LightGBM	2.27±0.00	19k	100	34
	LightGBM	2.25±0.00	91k	1k	21
	GB-TAO	2.23±0.02	31k	50	6
	avg-FAO	2.22±0.01	34k	50	6
	avg-FAO	2.20±0.01	68k	100	6
	avg-FAO	2.18±0.00	102k	150	6
CT-slice (43k,384)	LightGBM	6.57±0.00	9.1k	100	19
	XGBoost	6.40±0.00	419k	500	10
	XGBoost	6.39±0.00	739k	1k	10
	LightGBM	6.22±0.00	45.5k	500	20
	LightGBM	6.15±0.00	91k	1k	23
	GB-TAO	4.61±0.17	616k	50	8
	GB-TAO	4.60±0.17	1.7M	100	8
	avg-FAO	4.56±0.12	1.3M	100	8
	GB-TAO	4.47±0.24	780k	50	10
	avg-FAO	4.44±0.17	872k	50	10
	GB-TAO	4.47±0.24	2.3M	100	10
	avg-FAO	4.37±0.18	1.7M	100	10

	Forest	E_{test}	#pars.	T	Δ
casp (45k,9)	XGBoost	3.66±0.00	119k	100	10
	XGBoost	3.58±0.00	793k	1k	10
	LightGBM	3.54±0.00	153k	100	114
	GB-TAO	3.49±0.01	256k	50	12
	LightGBM	3.48±0.00	766k	1k	109
	avg-FAO	3.45±0.02	359k	50	12
	GB-TAO	3.43±0.00	481k	100	12
	avg-FAO	3.40±0.01	711k	100	12
	GB-TAO	3.39±0.01	887k	200	12
	avg-FAO	3.37±0.01	1.4M	200	12
year (450k,90)	XGBoost	9.05±0.00	153k	100	10
	LightGBM	9.03±0.00	153k	100	37
	XGBoost	9.00±0.00	568k	300	10
	LightGBM	8.92±0.00	460k	300	43
	LightGBM	8.92±0.00	1.5M	1k	43
	XGBoost	8.91±0.00	1.8M	1k	10
	GB-TAO	8.81±0.02	119k	30	6
	GB-TAO	8.77±0.01	200k	50	6
	GB-TAO	8.73±0.01	402k	100	6
	avg-FAO	8.72±0.00	392k	100	6
	avg-FAO	8.70±0.00	786k	200	6

Table 5.2: As Table 5.1 but for regression.

Main results Tables 5.1 and 5.2 report the performance of different forests on classification and regression benchmarks sorted by decreasing test error. The results are self-evident. FAO-avg has the lowest test error in all datasets, often by a considerable margin (e.g. MNIST, CT-slice). Even when the difference is small at the first side (e.g. SUSY), it is important to note that a slight improvement in those datasets is already hard to accomplish. Noticeably, we were able to achieve $\sim 1.3\%$ test error on MNIST which is about the same performance as was obtained using multipayer perceptrons. It worth to mention that we are not aware of any other tree ensembling framework that has shown similar results on MNIST using original pixel values. Among other baselines, GB-TAO is the second best approach followed by LightGBM. This again convincingly shows the power of optimization in ensemble training since GB-TAO also uses better optimized trees as base learners.

In terms of model size, our FAO-avg generates trees with total number of parameters that are similar or better than the baseline methods. This is surprising

given the fact that it uses oblique splits which generally have more parameters than axis-aligned ones. The reason is twofold: 1) we apply ℓ_1 penalty at each node that encourages sparsity; 2) FAO-avg was able to achieve such an outstanding performance by using small number of trees (by better optimizing them).

Training time Training $T = 10$ oblique trees of depth $\Delta = 6$ on MNIST for $FI = 20$ FAO iterations on a single processor takes about 12 minutes. For averaged FAO forests we train multiple of those independently, and so parallelization comes for free, and using just 6 processors on a typical machine we can obtain a forest with test accuracy of 1.5% in a matter of minutes.

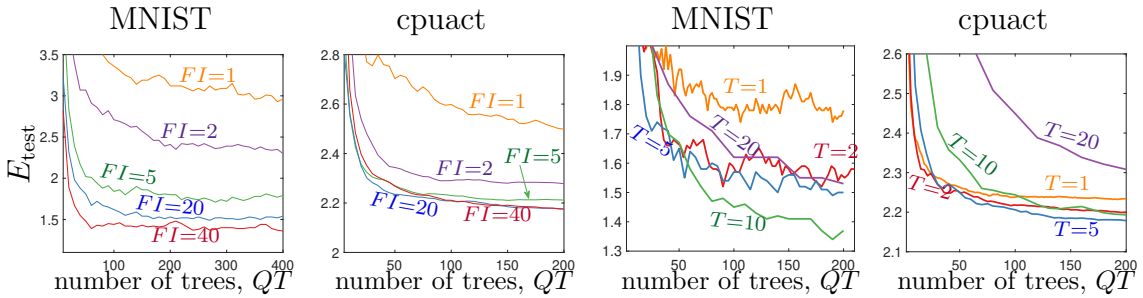

Figure 5.7: The effect of the number of FAO iterations FI and FAO forest size T for the overall ensemble of Q averaged FAO forests. We report 0-1 error on the MNIST dataset and RMSE on cpuact.

Exploring the number of FAO iterations and forest size The left two plots of fig. 5.7 show the effect of the number of FAO iterations FI on the generalization performance of the averaged FAO ensembles. The general observation is that better optimized individual small forests does indeed help to produce most accurate models. The right two plots show how the size of the individual FAO forests affect the model performance. We can observe that using very few trees gives inferior performance. While the curves with many trees ($T = 20$) still have the downward trend for the given budget of trees, it will result in a huge ensemble if we set the number of terms Q large.

Chapter 6

Generalized Additive Models with FAO on Stump Forests

Forest Alternating Optimization (FAO), introduced in the previous chapter, is a powerful method for optimizing tree ensembles. In this chapter, we focus on a special case: applying FAO to forests of decision stumps, depth-1, axis-aligned decision trees. These models are particularly interesting because they can be viewed as generalized additive models (GAMs), where the predictive function takes the form $F(\mathbf{x}) = \sum_{d=1}^{D} F_d(x_d)$, a sum of univariate functions over individual features. Because a stump (depth-one, axis-aligned decision tree) uses only a single feature, those using the same feature x_d can be grouped, and this forms a shape function $F_d(x_d)$ for that one feature x_d. As building blocks for GAMs, stumps have the advantage of being flexible in choosing the feature, and during learning they can adapt to the data distribution by using more of themselves for more complex shapes while choosing less the simpler ones. Unlike more general forests, the additive nature of stump forests enables explicit control over model smoothness via regularization of the complexity of individual shape functions. We analyze overfitting behavior of stump forests, and propose effective regularization methods based on the complexity of resulting GAM shape functions. The proposed algorithm achieves state-of-the-art accuracy in comparison with other GAM baselines while using fewer number of parameters.

6.1 Direct Optimization of Stump Forests

To keep notation simple we use regression to introduce the problem. Assume we have a training set of N (instance, target) pairs: $\{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^D \times \mathbb{R}$. Let $s(\mathbf{x}; \boldsymbol{\theta})$: $\mathbb{R}^D \to \mathbb{R}$ be a decision stump with 4 learnable parameters: $\boldsymbol{\theta} = \{\phi, \tau, \mu^l, \mu^r\}$. $\phi \in \{1, \ldots, D\}$ is a feature index to split, $\tau \in \mathbb{R}$ is a threshold value, $\mu^l, \mu^r \in \mathbb{R}$ are the left and right leaf prediction values. The predictive function of a stump $s(\mathbf{x}; \boldsymbol{\theta})$ works by comparing x_ϕ with τ, and outputting either μ^l if $x_\phi < \tau$, or μ^r otherwise:

$$s(\mathbf{x}; \boldsymbol{\theta}) = \begin{cases} \mu^l, & \text{if } x_\phi < \tau \\ \mu^r, & \text{if } x_\phi \geq \tau. \end{cases} \tag{6.1}$$

A stump forest $F(\mathbf{x}; \boldsymbol{\Theta})$ is defined as a sum of T stumps: $F(\mathbf{x}; \boldsymbol{\Theta}) = \mu + \sum_{t=1}^T s(\mathbf{x}; \boldsymbol{\theta}_t)$, with $\mu \in \mathbb{R}$ being a bias term. Since each stump uses only one feature, we can regroup the stumps by feature indices, and obtain an additive model:

$$\begin{aligned} F(\mathbf{x}; \boldsymbol{\Theta}) = \mu + \sum_{t=1}^T s(\mathbf{x}; \boldsymbol{\theta}_t) &= \mu + \sum_{d=1}^D \sum_{t:\ \phi_t = d} s(\mathbf{x}; \boldsymbol{\theta}_t) \\ &= \mu + \sum_{d=1}^D f_d(x_d) \end{aligned} \tag{6.2}$$

where a shape function $f_d(x_d)$ of d's feature is a sum of stumps that use feature d: $f_d(x_d) = \sum_{t:\ \phi_t = d} s(\mathbf{x}; \boldsymbol{\theta}_t)$. Since we are dealing with constant leaf predictor stumps, the shape function $f_d(x_d)$ is also a piecewise constant function with piece intervals corresponding to the stump thresholds. Viewing $F(\mathbf{x}; \boldsymbol{\Theta})$ as both a stump forest and an additive model will be important for our final learning method. Let us now first consider the learning problem of stumps forests without any regularization. For regression, we minimize a squared error:

$$\min_{\boldsymbol{\Theta}} \frac{1}{2} \sum_{n=1}^N [y_n - F(\mathbf{x}_n; \boldsymbol{\Theta})]^2. \tag{6.3}$$

Unlike in forward stagewise additive modeling of boosting, where stumps are added greedily to the forest, we want to directly optimize problem (6.3). Because stumps define a non-differentiable function, gradient-based methods are not applicable.

However, we can effectively apply alternating optimization: the parameters of one stump, while keeping the others fixed, can be optimized exactly; the parameters of all the leaf values $\{\mu_t^l, \mu_t^r\}_{t=1}^T$, while keeping the splits $\{\phi_t, \tau_t\}_{t=1}^T$ fixed, can also be optimized exactly. More specifically, alternating optimization consists of the following steps:

Individual stumps. The optimization problem (6.3) over a given stump $s(\cdot; \boldsymbol{\theta}_t)$ when others are fixed is $\min_{\boldsymbol{\theta}_t} \frac{1}{2} \sum_{n=1}^N [y_n - \sum_{u \neq t} s(\mathbf{x}_n; \boldsymbol{\theta}_u) - s(\mathbf{x}_n; \boldsymbol{\theta}_t)]^2$. This is a standard regression problem over a stump but with targets corresponding to the residuals. It can be solved exactly through enumeration over each (feature, threshold) pair as in traditional decision tree algorithms.

All leaf parameters. Once the splits $\{\phi_t, \tau_t\}_{t=1}^T$ are fixed, the problem (6.3) simplifies to a linear regression on features corresponding to stump partitions. To see this, we can rewrite the predictive function of a stump using an indicator function: $s(\mathbf{x}; \boldsymbol{\theta}) = \mu^l \; I(x_\phi < \tau) + \mu^r \; I(x_\phi \geq \tau)$. With this notation, the objective function over all leaves is:

$$\min_{\mu, \{\mu_t^l, \mu_t^r\}_{t=1}^T} \frac{1}{2} \sum_{n=1}^N \left[y_n - \sum_{t=1}^T \left(\mu_t^l \; I(x_{n\phi_t} < \tau_t) + \mu_t^r \; I(x_{n\phi_t} \geq \tau_t) \right) \right]^2 .$$

Since in this step all the splits $\{\phi_t, \tau_t\}_{t=1}^T$ are fixed, the indicator functions $I(\cdot)$ are just constants multiplying the unknowns $\{\mu_t^l, \mu_t^r\}_{t=1}^T$ that appear linearly inside the squared error. And so, eq. 6.4 is a simple linear regression problem on features induced by the stump splits, and can also be solved exactly and efficiently.

Starting with initial stump forest parameters (e.g. random), we can repeatedly optimize each stump individually, and all the leaves jointly. Because each such optimization step exactly solves the problem over its subset of parameters, this guarantees a monotonic decrease of the error. And since the total number of possible stump splits are finite, and since the algorithm either decreases the error or leaves it unchanged, this guarantees a convergence in a finite number of steps (assuming there are no cycles between states). While there are potentially many ways to order these alternating optimization steps, such as performing the step over

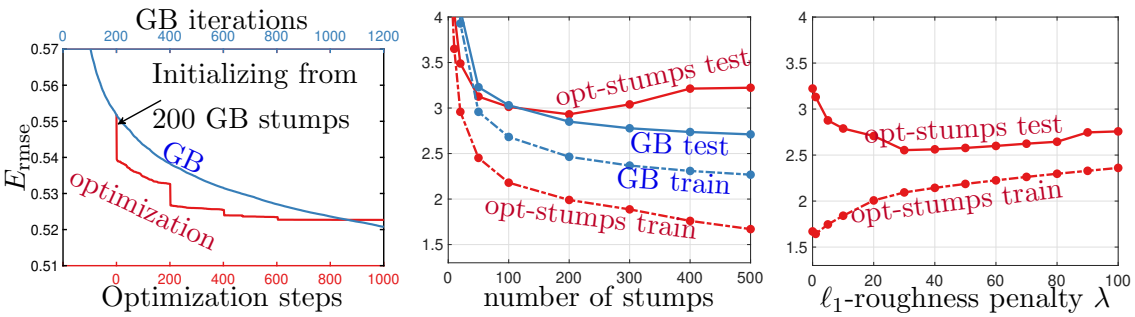

Figure 6.1: *Left:* Comparison of stump forest optimization against the greedy approach of gradient boosting (GB) in train error for the California Housing dataset. GB uses a learning rate of 1, i.e. no shrinkage. Optimization step corresponds to either the step over all leaves or over the individual stumps. *Middle:* Overfitting problem of optimized stump forests on the cpuact dataset. Here GB avoids overfitting by using a learning rate 0.3. *Right:* By penalizing the ℓ_1-discontinuity we obtain stump forests with better generalization (cpuact dataset).

all leaves after each individual stump step, or even doing everything in random order, in our experiments we use the following simple approach: first optimizing all leaves jointly, then optimizing the individual stumps in a fixed order (from 1 to T), and repeating this until convergence.

6.2 Overfitting and Regularization

The alternating optimization method just described for stump forests is very effective at minimizing the objective function. The left plot of fig. 6.1 shows how it is possible to take 200 greedily added decision stumps (in a gradient boosting (GB) way with a learning rate of 1), and optimize it significantly with the proposed algorithm so that it matches the performance of more than 1000 GB stumps (the red vs the blue curves in the plot). Because GB works by greedily adding stumps, without revisiting the previous parameters, it produces much suboptimal results. In the left plot of fig. 6.1, we can also observe the importance of jointly optimizing all the leaves: the sharp decreases in the red curve correspond to the optimization steps over all the leaf parameters. Since leaves account for half of the number of parameters, these significant drops in error should not be surprising (it would also be desirable to optimize multiple stump splits jointly, but since their

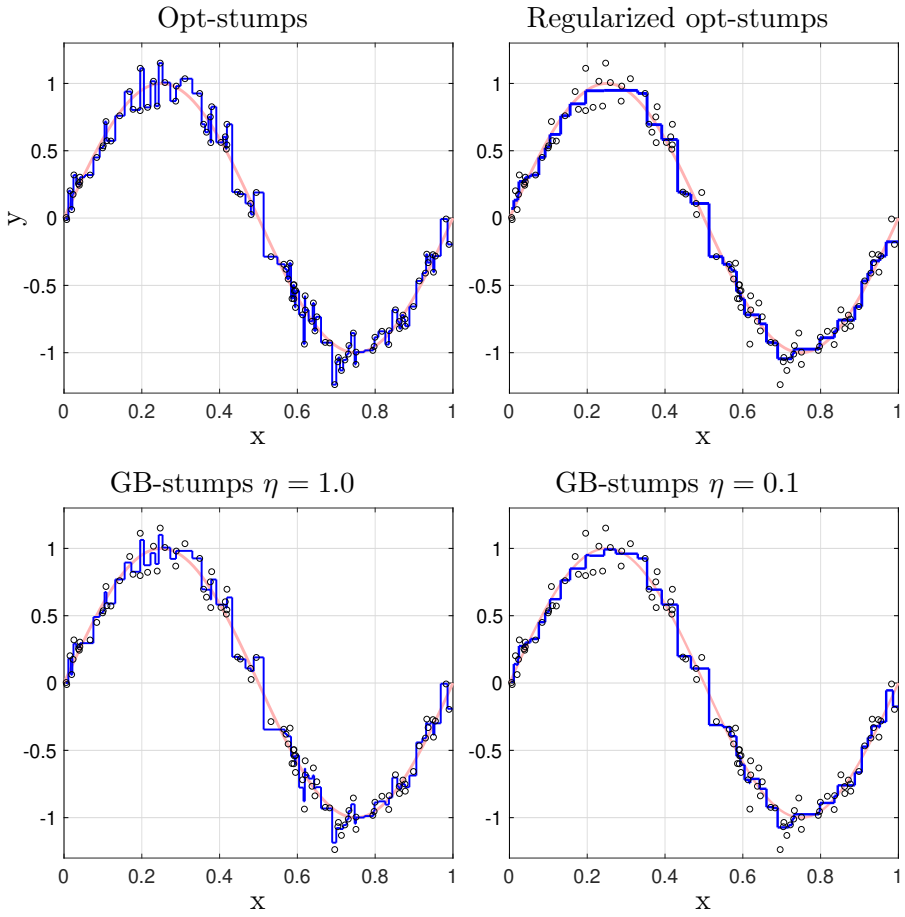

Figure 6.2: Illustration on a simple 1D regression problem. The learned shape functions are in blue. GB uses 1000 stumps with learning rates $\eta = 0.1$ and $\eta = 1.0$. Our optimization method uses 100 stumps. Ground truth targets are from a sine function plotted in light red.

search space grows exponentially with the number of stumps, this would make it computationally infeasible).

Despite being a simple model with relatively few parameters, an optimized stump forest tends to overfit the training data and exhibit poor generalization performance. The middle plot of fig. 6.1 illustrates that as the number of stumps increases, the test error also rises. Initially, this is not surprising, as a higher number of stumps results in a more flexible model, which can naturally lead to overfitting. However, gradient boosting with a small learning rate (i.e. a shrinkage parameter) can produce stump forests that are much larger, yet generalize much better. In the middle plot of fig. 6.1, for example, the GB with 500 stumps have

considerably better test error than any of the optimized stump forests, whose best test error is at around 200 stumps.

To understand better the overfitting problem, we use a simple 1D regression task where the resulting models can be easily visualized. The ground truth function is a $\sin(2\pi x)$. We generate 100 training points with added Gaussian noise to the targets. An optimized 100 stump forest achieves nearly 0 train error, and is visualized in the top plot in fig. 6.2. Clearly, the learned model exhibits poor generalization, as it overfits the training data by interpolating all the noise, resulting in a very wiggly and discontinuous function. This can be partly attributed to the model's high flexibility: with a sufficiently large number of stumps, the training set can be perfectly fit. Additionally, the model can operate very locally, where a few stumps can significantly alter the output in a localized region of the input while leaving the rest unchanged. Given the model's high flexibility and locality, such overfitting could be expected. (GB without shrinkage also suffers from overfitting, but with a small learning rate it does not. See the bottom two plots in fig. 6.2. This simple shrinkage method is very fundamental for the success of GB, and the theoretical understanding of it still remains unclear.)

Regularization is a fundamental technique to control overfitting. Since we know that stump forests define an additive model whose underlying geometry is a shape function per each feature, we can include regularization terms based on these shape functions. Such regularization will involve more global view of the model, thus can help in addressing the high flexibility and locality of stump forests. We define a roughness or discontinuity penalty function $r(\{\tau_t, \mu_t^l, \mu_t^r\}_{t:\ \phi_t=d})$: $\mathcal{P}_T(\mathbb{R}^3) \to \mathbb{R}_{\geq 0}$ that takes stumps grouped on the same feature d (specifically, their 3 parameters: threshold τ, left and right leaf predictions μ^l and μ^r), and outputs some measure of discontinuity of the shape function $f(d)$ resulting from the stumps. Its domain is the set of triplets of size at most T, as there are at most T stumps: $\mathcal{P}_T(\mathbb{R}^3) = \{S \subseteq \mathbb{R}^3 \mid |S| \leq T\}$. As a roughness penalty $r(\cdot)$ we use an ℓ_1 sum of discontinuities. To express it, let us first derive the explicit form of the shape function $f_d(x_d)$ from the set of stumps. Assume, without loss of generality, that the parameters $\{\tau_t, \mu_t^l, \mu_t^r\}_{t=1}^{T_d}$ are of the stumps that use feature d, and that their threshold values

are sorted $\tau_1 < \cdots < \tau_{T_d}$. Then the leftmost constant piece of the resulting shape function $f_d(x_d)$ is $\beta_0 = \sum_{t=1}^{T_d} \mu_t^l$, which corresponds to the left leaf predictions of all stumps (because for $x_d \in (-\infty, \tau)$, all stumps will route x_d to their left child). The next leftmost piece is then $\beta_1 = \mu_1^r + \sum_{t=2}^{T_d} \mu_t^l$. More generally, the value of the i's piece from the left is defined as $\beta_i = \sum_{t=1}^{i} \mu_t^r + \sum_{t=i+1}^{T_d} \mu_t^l$. The shape function $f_d(x_d)$ overall consists of the $T_d + 1$ constant pieces with values $\beta_0, \ldots, \beta_{T_d+1}$ from left to right, and with their starting intervals being $-\infty, \tau_1, \cdots, \tau_{T_d}$ (assuming all the stumps use a different threshold τ). With the shape function explicitly defined, we can now express the following roughness penalty:

$$r(\{\tau_t, \mu_t^l, \mu_t^r\}_{t=1}^{T_d}) = \sum_{t=0}^{T_d-1} |\beta_{t+1} - \beta_t|. \tag{6.4}$$

This penalizes the ℓ_1 difference between the two consecutive constant pieces of the shape function $f_d(x_d)$. It is exactly zero when the shape function $f_d(x_d)$ is constant, and is a large value if $f_d(x_d)$ is very wiggly and discontinuous. It can be viewed as a first order discrete derivative of the shape function $f_d(x_d)$. Another possibility is to penalize the ℓ_2 measure of discontinuity, but we experimentally observe the ℓ_1 difference producing better results. A similar type of smoothness penalty occurs in fused lasso [127] where the difference between neighboring coefficients is penalized, and in the ℓ_1 trend filtering approach for nonparametric regression [74].

Another regularization term that we propose is to penalize the deviation from the mean (or bias) for each of the leaf predictions: $\sum_{t=1}^{T} (\mu_t^l - \mu)^2 + (\mu_t^r - \mu)^2$. This encourages all stumps to be equally important, and can prevent stumps from being very dominant locally. This is conceptually similar to the idea of regularizing the leaf values proposed in [64], and currently implemented in popular gradient boosting decision tree libraries such as XGBoost and LightGBM. But instead of penalizing the actual leaf values we want to penalize their deviation from the bias. GB usually fits trees to the zero mean targets (by setting the initial constant prediction to the mean of the target), and so this type of similar regularization could already be happening in XGBoost.

With these two types of regularization, our final objective function is:

$$\min_{\boldsymbol{\Theta}} \frac{1}{2}\sum_{n=1}^{N}[y_n - F(\mathbf{x}_n;\boldsymbol{\Theta})]^2 + \lambda \sum_{d=1}^{D} r(\{\tau_t, \mu_t^l, \mu_t^r\}_{t:\ \phi_t=d}) + \alpha \sum_{t=1}^{T}\left((\mu_t^l - \mu)^2 + (\mu_t^r - \mu)^2\right) \tag{6.5}$$

The hyperparameter λ controls the strength of the roughness/discontinuity penalty, and we experimentally find that it is the more important one. The hyperparameter α controls the deviation from the bias, and plays less important role than λ. We do not tune it, and use a fixed value of $\alpha = 0.1$.

Now, for this regularized, more involved objective function, the alternating optimization steps from section 6.1 are still applicable, but with some changes:

Individual stumps. This is still solved exactly through enumeration over each (feature, threshold) pair but now the optimal value of the two leaves changes. The split finding algorithm will take into account the regularization terms when evaluating each possible split.

All leaf parameters. Now two regularization terms will be added to the problem (6.4):

$$\min_{\mu,\{\mu_t^l,\mu_t^r\}_{t=1}^T} \frac{1}{2}\sum_{n=1}^{N}\left[y_n - \sum_{t=1}^{T} \mu_t^l\ I(x_{n\phi_t} < \tau_t) + \mu_t^r\ I(x_{n\phi_t} \geq \tau_t)\right]^2 + \lambda \sum_{d=1}^{D} r(\{\tau_t, \mu_t^l, \mu_t^r\}_{t:\ \phi_t=d}) + \alpha \sum_{t=1}^{T}\left((\mu_t^l - \mu)^2 + (\mu_t^r - \mu)^2\right) \tag{6.6}$$

The optimization variables $\mu, \{\mu_t^l, \mu_t^r\}_{t=1}^T$ appear either linearly or quadratically in the regularization, and the whole problem still remains convex. Instead of developing a specialized algorithm for this problem, we use a generic convex solver. In our experiments, we model the problem using this high-level formulation, and rely on CVXPY [32] to translate it into a form acceptable to a solver. In our experiments we use the MOSEK solver inside CVXPY.

input training set
  initial forest $F(\cdot;\boldsymbol{\Theta})$ of T stumps
  roughness penalty λ, deviation from bias penalty α
repeat
  Optimize all leaves jointly by solving
  the convex problem (6.6)
  for $t = 1$ **to** T
    Optimize the individual stump $s(\cdot;\boldsymbol{\theta}_t)$
    through enumeration
until convergence
return optimized forest $F(\cdot;\boldsymbol{\Theta})$

Figure 6.3: Pseudocode for Optimized Regularized Stump Forests.

The proposed regularization terms control effectively the overfitting problem. In the right plot of fig. 6.1 we can observe how penalizing the ℓ_1-discontinuity helps in reducing the gap between training and test errors, and the resulting models are more accurate than the GB on the test set. In the second (from the left) plot in fig. 6.2 we see how these regularizations help to produce smoother stumps forests in the 1D example. We refer to our **O**ptimized **R**egularized **S**tump **F**orests as **ORSF**. We provide its pseudocode in fig. 6.3.

6.3 Experiments

Implementation of our algorithm We implement our algorithm in the combination of Python and C++. The optimization step over each stump is implemented in C++. The convex problem of joint leaf fitting is implemented in Python. We use the CVXPY open source Python package [32] to model the leaf fitting problem using high-level formulation, and rely on the library to convert/compile it into a specific solver form. As a convex solver inside CVXPY, we use MOSEK, a large scale optimization software. We use CVXPY version 1.4.3, and MOSEK version 10.1. We perform grid search on the following hyperparameter values: number of

stumps = {200, 400, 600, 800}, roughness penalty $\lambda = \{2.0, 4.0, 6.0\}$ for classification datasets, $\lambda = \{20.0, 40.0, 60.0\}$ for regression datasets. We do not tune the deviation from bias hyperparameter α, and use the fixed value $\alpha = 0.1$. As an initial stump forest we use stumps with random parameters.

6.3.1 Comparison on benchmarks

Our experimental results consistently demonstrate the improved accuracy of optimized stump forests across multiple benchmarks in both classification and regression tasks. More often than not, our optimized forests require fewer number of stumps, and thus fewer number of parameters overall.

We compare with the following established baselines for GAMs: explainable boosting machines (EBM) [89], gradient boosting (GB) with stumps, traditional cubic splines (implementation in Python [117]), neural additive models (NAM) [1] and fused lasso additive models (FLAM) [101]. We tune the important hyperparameters of all the baselines on a holdout set, and with the best found hyperparameters we repeat the experiment 5 times on different training/test splits to report mean and standard deviation. In our method, ORSF, for regression problems we use the squared error, and for binary classification we use the cross entropy loss with the logistic link function.

Table 6.1 shows the results on regression datasets. Across all the problems, regularized stump forests achieve competitive accuracy as some of the established state-of-the-art baselines for GAMs. Importantly, among the tree/stump based methods, our approach uses far fewer number of parameters than the GB, and especially, EBM. Boosting methods work by greedily adding trees/stumps with a small learning rate, typically requiring many boosting iterations to converge. This results in a large number of trees/stumps and corresponding GAM shape functions with many constant pieces. In contrast, our approach properly optimizes a stump forest and controls its complexity with regularization. As a result, hundreds of well-optimized stumps in our method match the performance of the thousands of stumps in GB and the tens of thousands of trees in EBM.

Table 6.2 presents results on classification problems. The general trend is qual-

Table 6.1: Train and test RMSE, model size (number of parameters) and training time (average ± standard deviation over 5 runs) for different GAMs. N refers to the dataset size, D is the feature dimension.

Dataset		ORSF	GB	EBM	Splines	NAM	FLAM
Cpuact	train	2.12±0.01	2.20±0.04	2.19±0.02	2.53±0.02	3.38±0.26	2.88±0.01
N=8.2k	test	2.37±0.03	2.43±0.06	2.50±0.05	2.69±0.06	3.41±0.28	2.99±0.05
D=21	size	642±0	3.4k±133	16.6k±36	271±3	134k±0	77.9k±123
	time (s)	189±25	46±17	39±2	37±0.03	99±1	85±2
Wine	train $\times 10^{-2}$	65.70±0.15	68.13±0.27	66.73±0.27	67.99±0.29	74.40±0.16	67.39±0.21
N=6.5k	test $\times 10^{-2}$	70.02±0.66	70.92±0.51	70.12±0.39	71.79±1.40	76.07±2.11	70.19±0.84
D=11	size	724±12	770±32	3.9k±11	197±7	70.1k±0	5041±11
	time (s)	64±3	2.87±0.58	4.44±1.33	56±16	64±0	53±3
Housing	train $\times 10^{-2}$	51.84±0.16	54.24±0.27	52.70±0.04	53.37±0.21	71.56±0.30	55.08±0.20
N=21k	test $\times 10^{-2}$	54.80±0.65	56.15±0.58	55.23±0.68	55.49±0.61	72.23±0.88	56.24±0.74
D=8	size	1.4k±20	2.4k±31	7.2k±8	528±2	51.0k±0	118k±101
	time (s)	600±140	42±8	36±2	37±2	175±2	73±2
Diamond	train $\times 10^{2}$	9.95±0.02	10.07±0.05	10.11±0.03	10.02±0.02	13.53±0.22	11.75±0.03
N=54k	test $\times 10^{2}$	10.15±0.08	10.19±0.08	10.23±0.06	10.96±1.45	13.59±0.25	11.70±0.12
D=26	size	934±16	1182±81	3.4k±7	273±24	86k±0	4139±12
	time (s)	648±120	140±58	20±2	42±0.4	708±2	805±11
Year	train	9.12±0.03	9.30±0.03	7.53±0.02	9.14±0.03	10.22±0.05	
N=423k	test	9.30±0.01	9.35±0.00	9.82±0.02	9.38±0.03	10.22±0.08	out of time
D=90	size	1379±0.8	1490±25	368k±0	2158±55	573k±0	$>$ 2 days
	time (s)	9681±205	4368±256	4262±437	3618±45	8858±88	
FPS	train	55.40±0.09	55.48±0.09	55.42±0.09	55.41±0.09	56.23±0.10	
N=401k	test	55.41±0.34	55.45±0.34	55.42±0.34	55.42±0.34	55.62±0.24	out of time
D=100	size	983±37	824±57	2372±12	411±1	288k±0	$>$ 2 days
	time (s)	6010±314	1803±466	655±84	2043±2	4397±10	

itatively similar to the regression case. Interestingly, traditional cubic splines show much better results here, often outperforming GB and EBM results. Splines have a rich history in statistics and possess many good theoretical properties. They also use much fewer number of parameters (degrees of freedom) while still being accurate. In the literature of GAMs, the importance of splines cannot be neglected.

6.3.2 Effect of different regularizations

In fig. 6.4, we examine the individual effects of three different types of regularizations on stump forests using the CPU activity dataset. As expected, all regularization methods exhibit typical behavior: increasing the regularization penalty initially results in high test error due to overfitting, followed by a decrease to the

Table 6.2: As in Table 6.1 but for classification datasets. The error is a 0/1 misclassification (%).

Dataset		ORSF	GB	EBM	Splines	NAM	FLAM
Letter	train	15.94±0.14	16.38±0.17	16.12±0.20	15.87±0.14	21.54±1.1	17.94±0.18
N=20k	test	16.40±0.52	16.88±0.41	16.63±0.42	16.55±0.70	22.53±1.88	17.95±0.51
D=16	size	403±13	420±15	502±2	224±1	68k±0	510±2
	time (s)	150±9	32±3	31±1	58±2	153±0	71±1
Churn	train	18.88±0.19	19.00±0.23	18.84±0.08	18.78±0.15	22.59±2.13	19.85±0.18
N=7.0k	test	19.28±0.29	19.32±0.37	19.47±0.51	19.32±0.48	21.69±2.02	20.30±0.88
D=45	size	129±5	644±48	7292±11	40±0.04	120k±0	13.7k±15
	time (s)	36±8	3±1	15±1	0.5±0.03	120±2	113±2
FICO	train	24.86±0.13	26.54±0.15	26.37±0.10	26.79±0.15	28.23±0.41	27.15±0.21
N=10k	test	27.33±0.04	27.62±0.30	27.43±0.31	27.35±0.17	28.08±0.61	27.64±0.52
D=23	size	550±28	1002±66	3680±9	83±1	130k±0	3791±11
	time (s)	231±18	1.6±0.6	7±0.2	1.96±0.10	180±1	61±1
IJCNN	train	4.42±0.05	4.56±0.07	4.51±0.03	4.44±0.04	7.51±0.44	6.86±0.08
N=50k	test	4.95±0.14	5.10±0.15	5.00±0.14	4.92±0.20	7.48±0.55	7.14±0.15
D=22	size	414±23	918±21	12.3k±0	266±0.5	101k±0	828k±242
	time (s)	1090±200	148±24	19±0	153±40	501±1	249±6
Covtype	train	22.50±0.03	22.56±0.02	22.46±0.02	22.48±0.02	26.16±0.50	
N=581k	test	22.71±0.11	22.77±0.10	22.68±0.12	22.72±0.10	26.08±0.54	out of time
D=54	size	504±4	1090±32	6402±4	403±1	170k±0	$>$ 2 days
	time (s)	4354±32	1202±49	325±5	15624±84	5373±16	
Bank	train	9.81±0.04	10.00±0.03	9.75±0.05	9.79±0.04	10.09±0.08	11.27±0.04
N=41k	test	9.83±0.17	9.99±0.13	9.91±0.17	9.88±0.12	9.87±0.26	11.23±0.15
D=62	size	231±4	530±15	1103±7	95±2	174k±0	1182±1
	time (s)	153±11	34±7	40±3	22±2	662±10	916±3

lowest test error indicating the best generalizable models, and finally, an increase in test error due to underfitting as the penalty becomes too strong. We can also observe the good ranges of the hyperparameter values, for example, the deviation from the bias penalty typically favors small values of the penalty parameter α. When comparing the importance of these 3 methods on the test error, the ℓ_1-roughness penalty usually produces models with the lowest test error than the other two. In our main experiments we use the combination ℓ_1-discontinuity and the deviation from the bias regularizations.

6.3.3 Case Study: Car Price Prediction

To showcase the interpretability of stump forest additive models, we use the problem of predicting the price of a used car. From the Kaggle platform we obtain

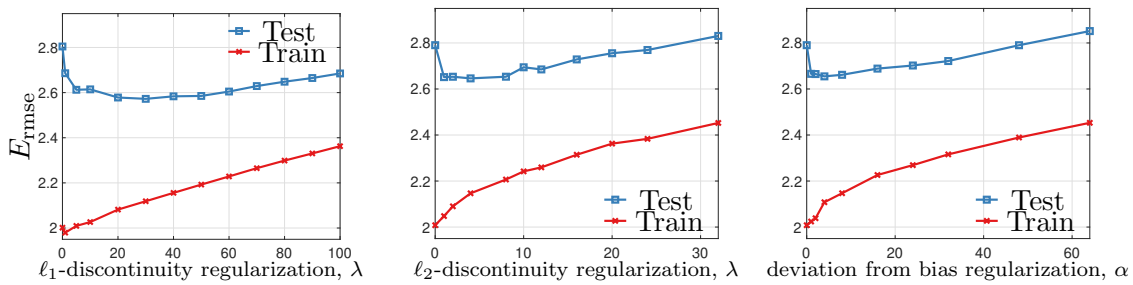

Figure 6.4: The effect of different regularization types in stump forests for the cpuact dataset. The number of stumps $T = 200$. For our final method we use the combination of ℓ_1-discontinuity and the deviation from the bias regularizations.

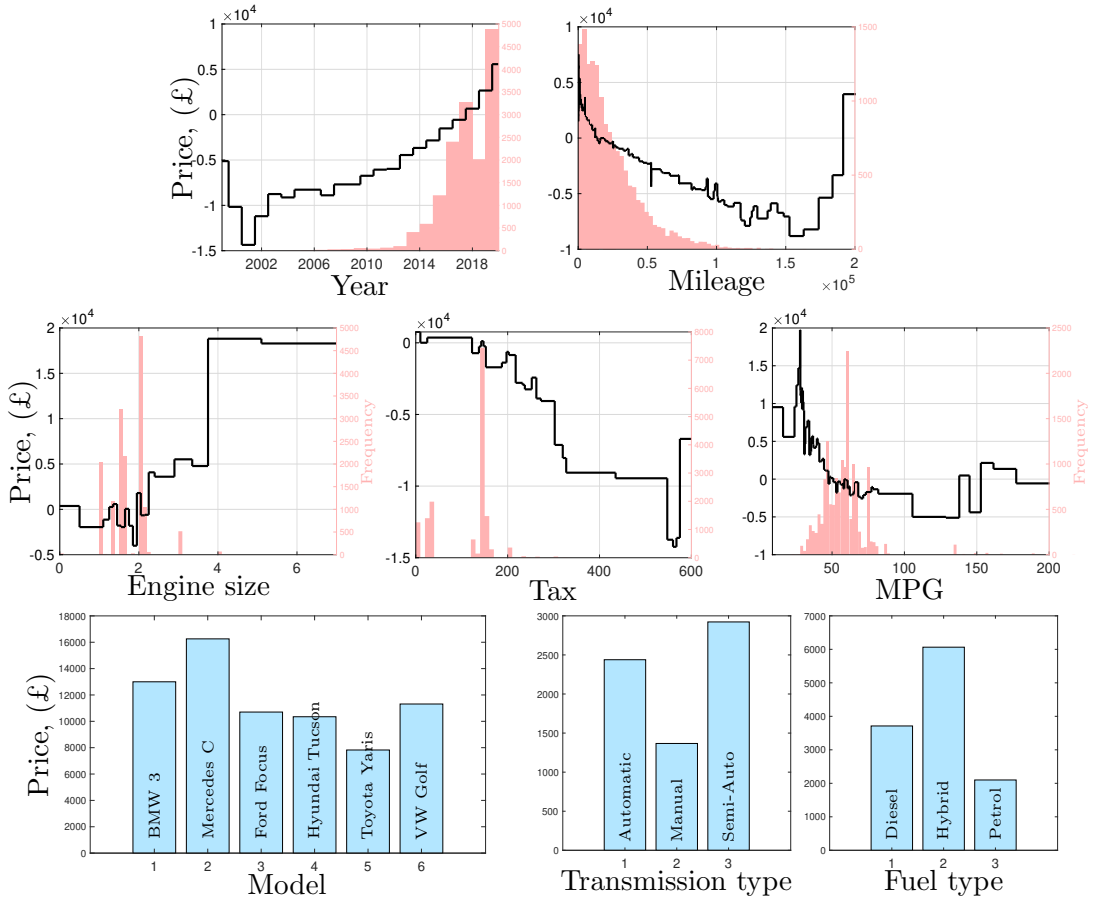

Figure 6.5: Visualization of the resulting additive model shape functions from our optimized stump forests for the UK used car dataset. For the numerical features, the light red bars show the histogram of the training points with the frequency values given on the right y-axis.

100,000 UK Used Car Data set, which features used car listings in UK, grouped by different car manufactures. To avoid the large number of car make/model

categories when visualizing, we focus on the following 6 popular models: BMW 3 series, Mercedes C Class, Volkswagen Golf, Ford Focus, Hyundai Tucson and Toyota Yaris. The total number of training points is 15,200 with additional 3,800 instances used for testing. The following 8 features describe a given car: the year of manufacture, mileage (a total number of miles traveled), amount of yearly tax, miles per gallon (MPG), engine size, a make/model, a transmission type and a fuel type. The first 5 features are numerical, and the last 3 are categorical.

We train our forest with 700 stumps and a roughness penalty of $\lambda = 60$. The model achieves a mean absolute error (MAE) of £1,600 on the test set. Considering the average price is £16,900 with a standard deviation of £7,600 on the whole dataset, this error margin is relatively acceptable. And the model is much more accurate than a linear regression which has a test MAE of £2,200. Most importantly, we can visualize the entire model as 8 plots for the 8 used features in fig. 6.5, and interpret it. There was also a constant bias term, but we eliminate it by distributing its value to the 3 categorical features.

We expect newer cars to be more expensive, and the shape function of the year feature generally confirms this expectation. There is an unexpected negative correlation with prices for the years 1999-2001, but this anomaly is due to having only six car listings from those years. Another crucial feature in predicting car prices is mileage; typically, cars with higher mileage are cheaper. The shape function for mileage supports this relationship. However, there is an unusual slight drop in price around 50,000 miles. A closer examination of the dataset reveals an outlier: a Mercedes C-Class with 52,700 miles listed at £2,140, whereas the normal price for this model is about £10,000. Miles per gallon (MPG), which characterizes fuel consumption, is another significant predictor variable. Its shape function generally shows a negative relationship with price. In the training set, we find newer Mercedes C-Class cars with engine sizes of 4.0 and 6.3 that have low MPG but are very expensive. Additionally, there are noisy points with mislabeled MPGs, such as eight newer BMW 3 Series cars with an MPG of 8.8, which should have been 30.0. Regarding categorical features, the distribution of prices within each category appears to be reasonable: luxury brands like Mercedes and BMW are

more expensive, cars with manual transmissions tend to be cheaper, and hybrid fuel types are usually more expensive. Overall, by visualizing and analyzing the model, we can understand it, and possibly even debug and correct the parameters and/or the dataset. This example on car price prediction clearly demonstrates the inherent interpretability of additive models.

Chapter 7

Conclusion and Future Work

In this dissertation, we studied the problem of learning ensembles of decision trees using alternating optimization. We showed that simply using more powerful oblique decision trees optimized with the TAO algorithm as base learners for two of the widely established boosting algorithms, AdaBoost and Gradient Boosting, results in better overall tree ensembles both in terms of accuracy and model size as compared against greedily induced suboptimal axis-aligned decision trees.

In terms of forest optimization, we made an important step further by properly defining an objective function over a decision forest of fixed size, and extended the alternating optimization approach to learn the entire forest without using any of the greedy techniques of boosting. The resulting Forest Alternating Optimization (FAO) algorithm proves highly effective at minimizing training error. However, due to the inherent flexibility and localized structure of decision forests, these models are prone to overfitting. We analyzed this issue in depth and showed that averaging several small, independently optimized forests prevents overfitting and delivers state-of-the-art test accuracy across a variety of benchmarks.

We further applied FAO to the special case of decision stump ensembles. Viewed as generalized additive models (GAMs), we proposed regularization strategies that successfully prevent overfitting for stump forests. The resulting alternating optimization algorithm for ensembles of stumps produces state-of-the-art GAMs in terms of both accuracy and model size. Given the widespread use of tree ensembles and interpretable GAMs in many practical applications, the algorithms

proposed in this dissertation can have a considerable practical impact.

Based on the results of this dissertation, the following future research directions are possible:

Efficient implementation of TAO and FAO. In machine learning and data science, the practical impact of new algorithms often depends on efficient, user-friendly implementations. For instance, although gradient boosting decision trees were introduced in the early 2000s, their widespread adoption was largely driven by the highly optimized XGBoost library. Similarly, the accessibility and popularity of neural networks have been significantly boosted by hardware acceleration and user-friendly frameworks like PyTorch and TensorFlow. In the same way, developing efficient, system-level implementations of TAO and FAO (which we are working on) can help unlock their full potential and make these advanced tree ensemble methods more accessible to practitioners.

Developing better methods to control overfitting. While simple averaging provides a basic mechanism for controlling overfitting, a promising direction for future research is the development of more principled objective functions and regularization techniques that intrinsically promote smoothness in the forest. This line of research connects to broader theoretical questions about generalization, particularly why some over-parameterized models, such as neural networks, often generalize well contrary to the traditional statistical wisdom.

Application of decision forests beyond machine learning. This dissertation has focused on the traditional machine learning setting, where the primary objective is strong generalization to unseen data. However, in other fields, such as computer graphics and data compression, models are often required to fit the training data precisely, with little concern for generalization. With their flexible modeling capacity and fast inference, decision forests are well-suited for such scenarios. These properties suggest promising opportunities for applying decision forests in domains beyond conventional machine learning. A first step in this direction is [26, 27], where a single image is modeled with oblique trees/forests. Other

successful applications include compressing neural networks by replacing its layers with forests [63], and using trees in radio selection problems in wireless sensor networks [123, 124].

Appendix A

Datasets description

Regression dataset description:

Cpuact The goal is to estimate the percentage of time a CPU operates in user mode. The features are various statistics related to memory and other activities. Source is the Delve project: `http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html`.

Wine The task is to predict the wine quality from 0 (very bad) to 10 (very excellent) based on the attributes as acidity, sugar, chlorides, etc. Obtained from the UCI ML repository [82].

Housing California Housing dataset, a standard regression benchmark. Obtained through the scikit-learn's `fetch_california_housing` function.

Diamond The task is to predict the price of a diamond based on the characteristics of clarity, carat weight, etc. We download it from the OpenML dataset repository: `https://www.openml.org/search?type=data&sort=runs&id=42225&status=active`.

Year predicting the release year of a song from audio features. Obtained from the UCI ML Repository [82].

FPS the goal is to predict the frames-per-second in video games based on characteristics of a CPU, GPU and the game itself. Sourced from the OpenML

repository:
https://www.openml.org/search?type=data&status=active&id=42737. We only use the "userbenchmark" source, and discard the "fpsbenchmark" source. We also drop the columns with missing values. We perform one-hot-encoding for the categorical variables.

CT slice the task is to predict the relative location of the CT slice on the axial axis of the human body. The features are histograms describing bone structures and air inclusions. We split into train, test, validation sets using by patient ID, so that not to mix patient slices. Obtained from the UCI Machine Learning Repository [82].

CASP a dataset of Physicochemical Properties of Protein Tertiary Structure. The task is to predict the size of the residue. Obtained from the UCI Machine Learning Repository [82].

Classification dataset description:

Letter English letter recognition task, available in the UCI ML Repository [82]. We convert it into binary classification by combining the letters A-M into one class and the rest to the other class.

Churn The binary classification task to predict which customer will churn. The features are various characteristics of the customer. Obtained from Kaggle: https://www.kaggle.com/datasets/blastchar/telco-customer-churn.

FICO Dataset from the Explainable Machine Learning Challenge. The task is to predict whether a consumer pays the credit on time or is overdue. The features come from an anonymized credit bureau data. Obtained from the official website:
https://community.fico.com/s/explainable-machine-learning-challenge.

IJCNN The dataset comes from an IJCNN 2001 competition. We obtain it from the LIBSVM binary dataset collection:
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

Covtype Classification task of pixels into 7 forest cover types based on attributes such as elevation, aspect, soil-type, slope, etc. We turn it into binary classification by as the majority class vs the rest. Sourced from the UCI ML Repository [82].

Bank the task to predict whether a marketing campaign of a banking institution is successful or not. The features are various characteristics of a customer. Obtained from the UCI ML Repository [82].

MNIST Handwritten digit recognition task [78]. The features are grayscale pixel values in [0,1] of each 28×28 digit image. We use the training/test partition in [78].

Char74k Image classification task [31] where each image contains a character in one 62 classes (0–9, A–Z, a–z). The characters were obtained from natural images, hand drawn using a tablet PC or synthesized from computer fonts. We used the modified version of [116], as described in [134]: this uses images resized into a grayscale image of 8×8 pixels, and the features are the 64 pixel grayscale values. We split the dataset randomly into 7 400 images as test and the rest as training.

R8 Top 8 (by frequency) classes of the Reuters-21578 text categorization dataset (a collection of documents that appeared on Reuters newswire in 1987). We use the preprocessed and stemmed version:
`https://www.cs.umb.edu/~smimarog/textmining/datasets`. We train the Doc2Vec model available from the `gensim` package [130] on all the document collections and obtain 400-dimensional word-embedding features.

RCV1 Text categorization dataset [79]. We obtained it from the LIBSVM multiclass data

SUSY a problem in physics to classify a process into a signal which produces super symmetric particles or into a background which does not [6].

CIFAR100 a standard image classification benchmark in computer vision. We use as features the output of the last convolutional layer of a pretrained VGG16 network [76].

News20 a standard document classification benchmark. The features are normalized word counts. Obtained from a LIBSVM multiclass data collection `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`.

Bibliography

[1] R. Agarwal, L. Melnick, N. Frosst, X. Zhang, B. Lengerich, R. Caruana, and G. E. Hinton. Neural additive models: Interpretable machine learning with neural nets. In M. Ranzato, A. Beygelzimer, P. Liang, J. W. Vaughan, and Y. Dauphin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34. MIT Press, Cambridge, MA, 2021.

[2] G. Aglin, S. Nijssen, and P. Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proc. of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 3146–3153, New York, NY, Feb. 7–12 2020.

[3] Z. G. Alex Davies. The random forest kernel and other kernels for big data from random partitions. arXiv:1402.4293, 2014.

[4] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, Oct. 1997.

[5] S. Arlot and R. Genuer. Analysis of purely random forests bias. arXiv:1407.3939, 2014.

[6] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5:4308, 2014.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *J. Machine Learning Research*, 3(1137–1155), Feb. 2003.

[8] K. P. Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 26:156–160, 1994.

[9] K. P. Bennett and J. A. Blue. Optimal decision trees. Technical Report 214, Dept. of Mathematical Sciences, Rensselaer Polytechnic Institute, 1996.

[10] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. In *Int. J. Conf. Neural Networks (IJCNN'98)*, pages 2396–2401, Anchorage, Alaska, May 4–9 1998.

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

[12] G. Biau and E. Scornet. A random forest guided tour. *TEST*, 25(2):197–227 (with comments, pp. 228–268), June 2016.

[13] L. Breiman. Fitting additive models to regression data: Diagnostics and alternative views. *Computational Statistics and Data Analysis*, 15(1):13–46, Jan. 1993.

[14] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493—-1517, Oct. 1999.

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

[16] L. Breiman. Consistency for a simple model of random forests. Tech. rep. 670, Dept. of Statistics, University of California, Sept. 9 2004.

[17] L. Breiman and J. H. Friedman. Estimating optimal transformations for multiple regression and correlation. *J. Amer. Stat. Assoc.*, 80(391):580–598 (with comments, pp. 598–619), Sept. 1983.

[18] L. Breiman and C. J. Stone. Parsimonious binary classification trees. Technical report, Santa Monica, Calif.: Technology Service Corporation, 1978.

[19] L. J. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug. 1996.

[20] L. J. Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801–824 (with comments, pp. 824–849), June 1998.

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

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

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

[24] 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.

[25] M. Á. Carreira-Perpiñán, M. Gabidolla, and A. Zharmagambetov. Towards better decision forests: Forest Alternating Optimization. In *Proc. of the 2023 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'23)*, pages 7589–7598, Vancouver, Canada, June 18–22 2023.

[26] E. Chan, M. Gabidolla, and M. Á. Carreira-Perpiñán. Cubist-style image effects with oblique decision trees. In *Bay Area Machine Learning Symposium (BayLearn 2024)*, Cupertino, CA, Oct. 10 2024.

[27] E. Chan, M. Gabidolla, and M. Á. Carreira-Perpiñán. Oblique decision trees as an image model for cubist image restyling. In *IEEE International Conference on Image Processing (ICIP 2025)*, Anchorage, AK, Sept. 14–17 2025.

[28] T. Chen and C. 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.

[29] H. Chipman, E. George, and R. Mcculloch. Bayesian ensemble learning. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.

[30] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *Annals of Applied Statistics*, 4(1):266–298, Mar. 2010.

[31] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proc. Int. Conf. Computer Vision Theory and Applications (VISAPP 2009)*, pages 273–280, Lisbon, Portugal, Feb. 5–8 2009.

[32] S. Diamond and S. Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *J. Machine Learning Research*, 17(83):1–5, 2016.

[33] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, Aug. 2000.

[34] P. H. C. Eilers and B. D. Marx. Flexible smoothing with b-splines and penalties. *Statistical Science*, 11(2):89–121, May 1996.

[35] P. H. C. Eilers and B. D. Marx. Splines, knots, and penalties. *WIREs Computational Statistics*, 2(6):637–653, Sept. 2010.

[36] 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.

[37] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences*, 55(1):119–139, 1997.

[38] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–407, Apr. 2000.

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

[40] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, Jan. 2002.

[41] J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954, Sept. 2008.

[42] N. Frosst and G. Hinton. Distilling a neural network into a soft decision tree. arXiv:1711.09784, Nov. 27 2017.

[43] M. Gabidolla and M. Á. Carreira-Perpiñán. Pushing the envelope of gradient boosting forests via globally-optimized oblique trees. In *Proc. of the 2022 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'22)*, pages 285–294, New Orleans, LA, June 19–24 2022.

[44] M. Gabidolla and M. Á. Carreira-Perpiñán. Optimal interpretable clustering using oblique decision trees. In *Proc. of the 28th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2022)*, pages 400–410, Washington, DC, Aug. 14–18 2022.

[45] M. Gabidolla, A. Zharmagambetov, and M. Á. Carreira-Perpiñán. Improved multiclass AdaBoost using sparse oblique decision trees. In *Int. J. Conf. Neural Networks (IJCNN'22)*, Padua, Italy, July 18–22 2022.

[46] M. Gabidolla, A. Zharmagambetov, and M. Á. Carreira-Perpiñán. Beyond the ROC curve: Classification trees using Cost-Optimal Curves, with application to imbalanced datasets. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proc. of the 41th Int. Conf. Machine Learning (ICML 2024)*, pages 14343–14364, Vienna, Austria, July 21–27 2024.

[47] K. Gazizov, A. Zharmagambetov, and M. Á. Carreira-Perpiñán. A critical comparison of soft vs hard oblique classification trees. 2025.

[48] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr. 2006.

[49] J. Good, T. Kovach, K. Miller, and A. Dubrawski. Feature learning for interpretable, performant decision trees. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 66571–66582. MIT Press, Cambridge, MA, 2023.

[50] J. Goodman. Classes for fast maximum entropy training. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'01)*, pages 561–564, Salt Lake City, Utah, USA, May 7–11 2001.

[51] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 507–520. MIT Press, Cambridge, MA, 2022.

[52] T. J. Hastie and R. J. Tibshirani. Generalized additive models. *Statistical Science*, 1(3):297–318 (with comments), Aug. 1986.

[53] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1990.

[54] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

[55] H. Hazimeh, N. Ponomareva, P. Mol, Z. Tan, and R. Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In H. Daumé III and A. Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 4138–4148, Online, July 13–18 2020.

[56] E. Henrichon and K.-S. Fu. A nonparametric partitioning procedure for pattern classification. *IEEE Trans. Computers*, C–18(7):614–624, 1969.

[57] J. Hill, A. Linero, and J. Murray. Bayesian additive regression trees: A review and look forward. *Ann. Rev. Statistics and Its Application*, 7:251–278, Mar. 2020.

[58] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(8):832–844, Aug. 1998.

[59] X. Hu, C. Rudin, and M. Seltzer. Optimal sparse decision trees. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 7267–7275. MIT Press, Cambridge, MA, 2019.

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

[61] S. Ibrahim, H. Hazimeh, and R. Mazumder. Flexible modeling and multitask learning using differentiable tree ensembles. In *Proc. of the 28th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2022)*, pages 666–675, Washington, DC, Aug. 14–18 2022.

[62] S. Ibrahim, G. Afriat, K. Behdin, and R. Mazumder. Grand-slamin' interpretable additive modeling with structural constraints. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36. MIT Press, Cambridge, MA, 2023.

[63] Y. Idelbayev, A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán. Faster neural net inference via forests of sparse oblique decision trees. arXiv, 2022.

[64] R. Johnson and T. Zhang. Learning nonlinear functions using regularized

greedy forest. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 36 (5):942–954, May 2013.

[65] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, Mar. 1994.

[66] Kaggle. State of data science and machine learning 2021, Oct. 14 2021.

[67] R. Kairgeldin, M. Gabidolla, and M. Á. Carreira-Perpiñán. Adaptive softmax trees for many-class classification. In N. Kiyavash and J. M. Mooij, editors, *Proc. of the 40th Conf. Uncertainty in Artificial Intelligence (UAI 2024)*, pages 1825–1841, Barcelona, Spain, July 15–19 2024.

[68] A. Karthikeyan, N. Jain, N. Natarajan, and P. Jain. Learning accurate decision trees with bandit feedback via quantized gradient descent. *Trans. Machine Learning Research*, Sept. 2023.

[69] R. Katuwal, P. N. Suganthan, and L. Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078, Mar. 2020.

[70] V. Kazemi and J. 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.

[71] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. 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.

[72] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, Jan. 1994.

[73] H. Kim and W.-Y. Loh. Classification trees with unbiased multiway splits. *J. Amer. Stat. Assoc.*, 96:589–604, Feb. 2001.

[74] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky. ℓ_1 trend filtering. *SIAM Review*, 51(2):339–360, 2009.

[75] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Buló. Deep neural decision forests. In *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*, pages 1467–1475, Santiago, Chile, Dec. 11–18 2015.

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

[77] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, second edition, 2014.

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

[79] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. Machine Learning Research*, 5: 361–397, Apr. 2004.

[80] P. Li. Robust logitboost and adaptive base class (abc) logitboost. In *Proc. of the 26th Conf. Uncertainty in Artificial Intelligence (UAI 2010)*, pages 302–311, Catalina Island, CA, July 8–11 2010.

[81] P. Li, Z. Qin, X. Wang, and D. 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.

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

[83] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer. Generalized and scalable optimal sparse decision trees. In H. Daumé III and A. Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 6150–6160, Online, July 13–18 2020.

[84] B. Liu and R. Mazumder. Forestprune: Compact depth-pruned tree ensembles. In F. Ruiz, J. Dy, and J.-W. van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 9417–9428. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/liu23h.html.

[85] B. Liu and R. Mazumder. Fast: An optimization framework for fast additive segmentation in transparent ml. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 1863–1874, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671996. URL https://doi.org/10.1145/3637528.3671996.

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

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

[88] Y. Lou, R. Caruana, and J. Gehrke. Intelligible models for classification and regression. In *Proc. of the 18th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2012)*, pages 150–158, Aug. 21–24 2012.

[89] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. Accurate intelligible models with pairwise interactions. In *Proc. of the 19th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2013)*, pages 623–631, Chicago, IL, Aug. 11–14 2013.

[90] D. McElfresh, S. Khandagale, J. Valverde, V. P. C, G. Ramakrishnan, M. Goldblum, and C. White. When do neural nets outperform boosted trees on tabular data? In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing*

Systems (NeurIPS), volume 36, pages 76336–76369. MIT Press, Cambridge, MA, 2023.

[91] D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *J. Machine Learning Research*, 9:131–156 (with discussion, pp. 157–201), 2008.

[92] L. Mentch and S. Zhou. Randomization as regularization: A degrees of freedom explanation for random forest success. *Journal of Machine Learning Research*, 21(171):1–36, 2020. URL `http://jmlr.org/papers/v21/19-905.html`.

[93] R. Messenger and L. Mandell. A modal search technique for predictive nominal scale multivariate analysis. *J. Amer. Stat. Assoc.*, 67(340):768–772, Dec. 1972.

[94] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR Workshop*, 2013.

[95] J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *J. Amer. Stat. Assoc.*, 58(302):415–434, June 1963.

[96] K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning Series. MIT Press, 2022.

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

[98] 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.

[99] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138(2):221–254, Sept. 1 2003.

[100] 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`.

[101] A. Petersen, D. Witten, and N. Simon. Fused lasso additive model. *Journal of Computational and Graphical Statistics*, 25(4):1005–1025, 2016.

[102] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: Unbiased boosting with categorical features. 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 6638–6648. MIT Press, Cambridge, MA, 2018.

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

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

[105] F. Radenovic, A. Dubey, and D. Mahajan. Neural basis models for interpretability. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 8414–8426. MIT Press, Cambridge, MA, 2022.

[106] G. Ratsch, T. Onoda, and K.-R. Muller. Soft margins for adaboost. *Machine Learning*, 42:287—-320, Mar. 2001.

[107] S. Ren, X. Cao, Y. Wei, and J. 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.

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

[109] D. Ruppert. Selecting the number of knots for penalized splines. *Journal of Computational and Graphical Statistics*, 11(4):735–757, Jan. 2002.

[110] M. Saberian and N. Vasconcelos. Multiclass boosting: Margins, codewords, losses, and algorithms. *J. Machine Learning Research*, 20(137):1–68, 2019.

[111] V. Sadhanala and R. J. Tibshirani. Additive models with trend filtering. *Annals of Statistics*, 47(6):3032–3068, 2019.

[112] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2): 197–227, June 1990.

[113] R. E. Schapire and Y. Freund. *Boosting. Foundations and Algorithms*. Adaptive Computation and Machine Learning Series. MIT Press, 2012.

[114] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297–336, Dec. 1999.

[115] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[116] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. 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.

[117] D. Servén and C. Brummitt. pygam: Generalized additive models in python, 2018.

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

[119] R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, May 2022.

[120] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C.-J. Hsieh. Gradient boosted decision trees for high dimensional sparse output. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 3182–3190, Sydney, Australia, Aug. 6–11 2017.

[121] D. Sorokina, R. Caruana, and M. Riedewald. Additive groves of regression trees. In J. N. Kok, J. Koronacki, R. López de Mantaras, S. Matwin, D. Mladenic, and A. Skowron, editors, *Proc. of the 18th European Conf. Machine Learning (ECML–07)*, pages 323–334, Warsaw, Poland, Sept. 17–21 2007.

[122] P. Sun, M. D. Reid, and J. Zhou. Aoso-logitboost: Adaptive one-vs-one logitboost for multi-class problem. In J. Langford and J. Pineau, editors, *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, Edinburgh, Scotland, June 26 – July 1 2012.

[123] J. P. S. Sundaram, A. Zharmagambetov, M. Gabidolla, M. Á. Carreira-Perpiñán, and A. Cerpa. Mars: Multi-radio architecture with radio selection using decision trees for emerging mesoscale cps/iot applications, 2024. URL `https://arxiv.org/abs/2409.18043`.

[124] J. P. S. Sundaram, M. Gabidolla, M. Á. Carreira-Perpiñán, and A. E. Cerpa. Comnets: Cost-sensitive decision trees approach to throughput optimization for multi-radio iot networks, 2025. URL `https://arxiv.org/abs/2502.03677`.

[125] Y. S. Tan, C. Singh, K. Nasseri, A. Agarwal, J. Duncan, O. Ronen, M. Epland, A. Kornblith, and B. Yu. Fast interpretable greedy-tree sums. *Proc. Natl. Acad. Sci. USA*, 122(7):e2310151122:1–10, Feb. 18 2025.

[126] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori. Adaptive neural trees. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc.*

of the 36th Int. Conf. Machine Learning (ICML 2019), pages 6166–6175, Long Beach, CA, June 9–15 2019.

[127] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *J. Statistical Society B*, 67(1):91–108, Feb. 2005.

[128] T. M. Tomita, J. Browne, C. Shen, J. Chung, J. L. Patsolic, B. Falk, C. E. Priebe, J. Yim, R. Burns, M. Maggioni, and J. T. Vogelstein. Sparse projection oblique randomer forests. *J. Machine Learning Research*, 21:1–39, 2020.

[129] S. Verwer and Y. Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 1625–1632, Honolulu, HI, Jan. 27 – Feb. 1 2019.

[130] R. Řehůřek and P. Sojka. Software framework for topic modelling with large corpora. In *Proc. LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 22 2010.

[131] S. N. Wood. Thin plate regression splines. *J. Statistical Society B*, 65(1): 95–114, Feb. 2003.

[132] S. N. Wood. *Generalized Additive Models: An Introduction with R*. Texts in Statistical Science. Chapman & Hall/CRC, second edition, 2017.

[133] A. J. Wyner, M. Olson, J. Bleich, and D. Mease. Explaining the success of AdaBoost and random forests as interpolating classifiers. *J. Machine Learning Research*, 18(48):1–33, 2017.

[134] Z. Zhang, P. Sturgess, S. Sengupta, N. Crook, and P. H. S. Torr. Efficient discriminative learning of parametric nearest neighbor classifiers. In *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'12)*, pages 2232–2239, Providence, RI, June 16–21 2012.

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

[136] A. Zharmagambetov, S. S. Hada, M. Á. Carreira-Perpiñán, and M. Gabidolla. An experimental comparison of old and new decision tree algorithms. arXiv:1911.03054, Mar. 20 2020.

[137] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán. Improved boosted regression forests through non-greedy tree optimization. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[138] A. Zharmagambetov, M. Gabidolla, and M. 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.

[139] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán. Softmax tree: An accurate, fast classifier when the number of classes is large. In M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, editors, *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP 2021)*, pages 10730–10745, Online, 2021.

[140] A. Zharmagambetov, S. S. Hada, M. Gabidolla, and M. Á. Carreira-Perpiñán. Non-greedy algorithms for decision tree optimization: An experimental comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[141] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC Publishers, 2012.

[142] J. Zhu, H. Zou, S. Rosset, and T. Hastie. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360, 2009.