UNIVERSITY OF CALIFORNIA, MERCED

Learning Tree-Based Models with Manifold Regularization: Alternating Optimization Algorithms

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

in

Electrical Engineering & Computer Science

by

Arman Zharmagambetov

Committee in charge:

Professor Miguel Á. Carreira-Perpiñán, Chair
Professor Ming-Hsuan Yang
Professor Hyeran Jeon

2022

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

(Professor Ming-Hsuan Yang)

(Professor Hyeran Jeon)

(Professor Miguel Á. Carreira-Perpiñán, Chair)

University of California, Merced

2022

DEDICATION

To my parents and grandparents:

Serik Zharmagambetov and Galiya Shabdirova

Suiesh Zharmagambetov and Saniya Zharmagambetova

and to my family:

Kalamkas Kuanyshbekova and Tomiris Serikova

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Miguel Á. Carreira-Perpiñán for helping me enormously at every step of my PhD journey. This dissertation would not be possible without him. He is a great advisor and I will definitely miss our long and comprehensive research discussions at his office. I would like to thank my current and past thesis committee members: Marcelo Kallmann, Ming-Hsuan Yang and Hyeran Jeon for giving me critical comments and advices. Special thanks to Sungjin Im for being an amazing TA mentor and for teaching excellent grad level courses. I gratefully acknowledge UC Merced's Office of the Chancellor and School of Engineering for their support of my PhD study.

Dulat Dzhumabaev, Alexander Pak, Sergazy Narynov, and Bolatbek Rysbaiuly have greatly influenced my career by sparkling an interest in science and technology. A huge kudos to my dearest friends Shynggys, Raiymbek, Yermek and other members of Robotics lab at IITU where I had my first steps in machine learning. This list should also include friends that I met in the US: Nariman, Batyrkhan, Nursultan, Aidyn, and many others, including an amazing Kazakh community in Merced and Bay Area. I also want to mention my labmates and collaborators: Magzhan, Suryabhan, Yerlan. Thanks for creative discussions and supportive comments!

I would like to thank my parents Serik, Galiya and grandparents Suiesh, Saniya who raised me with love and encouraged me to pursue my dreams. Your son is doing great! I am forever grateful to my relatives who had a huge impact on my life: my aunt Aigul, little sister Alina and brother Almas. Finally and most importantly, I would like to express my love and appreciation to my dear wife, Kalamkas, for her immense support and inspiration; and to our daughter Tomiris. Thank you both for always being there for me, for listening and healing! This dissertation would not be possible without my girls :-)

Thank you all, sincerely!

VITA

2015	B. S. in Mathematical and Computer Modeling *cum laude*, International Information Technology University, Almaty, Kazakhstan
2017	M. S. in Mathematical and Computer Modeling, International Information Technology University, Almaty, Kazakhstan
2017	UC Merced Chancellor's Graduate Fellowship for Inclusive Excellence
2019	Outstanding Teaching Award for valuable contributions to teaching and pedagogy at University of California, Merced
2022	Ph. D. in Electrical Engineering and Computer Science, University of California, Merced

PUBLICATIONS

A. Zharmagambetov and M. Á. Carreira-Perpiñán. Semi-Supervised Learning with Decision Trees: Graph Laplacian Tree Alternating Optimization. Advances in Neural Information Processing Systems (NeurIPS), 2022.

A. Zharmagambetov and M. Á. Carreira-Perpiñán. Learning Interpretable, Tree-Based Projection Mappings for Nonlinear Embeddings. International Conf. on Artificial Intelligence and Statistics (AISTATS), 2022.

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 and M. Á. Carreira-Perpiñán. Smaller, More Accurate Regression Forests Using Tree Alternating Optimization. International Conf. on Machine Learning (ICML), 2020.

A. Zharmagambetov, Q. Tang , C.-C. Kao, Q. Zhang, M. Sun, V. Rozgic, J. Droppo, C. Wang. Improved Representation Learning for Acoustic Event Classification Using Tree-structured Ontology. IEEE International Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2022.

A. Zharmagambetov and M. Á. Carreira-Perpiñán. Learning a Tree of Neural Nets. IEEE International Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2021.

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.

A. Zharmagambetov and M. Gabidolla and M. Á. Carreira-Perpiñán. Improved Boosted Regression Forests Through Non-Greedy Tree Optimization. International Joint Conf. on Neural Networks (IJCNN), 2021.

A. Zharmagambetov and S. S. Hada and M. Gabidolla and M. Á. Carreira-Perpiñán. Non-Greedy Algorithms for Decision Tree Optimization: An Experimental Comparison. 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.

A. Zharmagambetov and M. Á. Carreira-Perpiñán. A Simple, Effective Way to Improve Neural Net Classification: Ensembling Unit Activations with a Sparse Oblique Decision Tree. IEEE International Conf. on Image Processing (ICIP), 2021.

S.S.Hada, M. Á. Carreira-Perpiñán, A. Zharmagambetov. Understanding and Manipulating Neural Net Features Using Sparse Oblique Classification Trees. IEEE International Conf. on Image Processing (ICIP), 2021.

M. Á. Carreira-Perpiñán and A. Zharmagambetov. Ensembles of bagged TAO trees consistently improve over Random Forests, AdaBoost and Gradient Boosting. ACM-IMS Foundations of Data Science Conf. (FODS), 2020.

S. Narynov and A. Zharmagambetov. On One Approach of Solving Sentiment Analysis Task for Kazakh and Russian Languages Using Deep Learning. Int. Conf. on Computational Collective Intelligence (ICCCI), Halkidiki, Greece, 2016. Springer.

A. A. Pak, S. Narynov, A. Zharmagambetov, Sh. Sagyndykova, Zh. Kenzhebayeva. The Method of Synonyms Extraction from Unannotated Corpus. In proceedings of the IEEE 3rd Int. Conf. on Digital Information, Networking, and Wireless Communications (DINWC), Moscow, Russia, 2015.

ABSTRACT OF THE DISSERTATION

Learning Tree-Based Models with Manifold Regularization: Alternating Optimization Algorithms

by

Arman Zharmagambetov
Doctor of Philosophy in Electrical Engineering & Computer Science

University of California Merced, 2022

Professor Miguel Á. Carreira-Perpiñán, Chair

Decision trees (DT) are considered to be one of the oldest machine learning models which received a lot of attention from practitioners and research community. Although their roots are in the 1950s, they became popular in the early 1980s with developing popular methods, such as CART and C4.5. They are conceptually simple yet powerful. State-of-the-art frameworks, such as XGBoost or LightGBM, rely on them as base learners, but they have been used as well as standalone predictors.

Despite the rich history of decision trees and existence of numerous methods, their applicability beyond traditional supervised learning has been explored in limited extent. For instance, various fast growing ML subfields, such as semi-supervised and self-supervised learning, nonlinear dimensionality reduction (e.g. nonlinear embeddings), etc. have been barely used with trees. What is common to most of these tasks is that the objective function takes a certain form, which involves *manifold regularization* to exploit the geometry of the underlying data distribution. For example, a common assumption is that similar instances have similar predictions. However, incorporating this type of regularization is non-trivial with decision trees. The main reason is that the learning decision trees from data (even in supervised learning setup) is still an open research problem because it involves solving non-trivial combinatorial optimization problem. In fact, finding an optimal decision tree or even constant-factor approximation is NP-hard in most formulations of the problem. Adding manifold regularization term into an overall objective makes the problem even harder to solve.

In this dissertation, we study decision trees and, more generally, tree-based models under above-mentioned settings, i.e., involving manifold regularization. We argue that these type of problems carry a great practical importance but directly solving them is intractable (for decision trees). Using semi-supervised learning and nonlinear dimensionality reduction as examples, we derive a generic algorithm to solve such optimization problems. It is based on a reformulation of the problem which requires iteratively solving two simpler problems. One problem always involves supervised learning of a tree, which we solve by the *Tree Alternating Optimization* algorithm. The second one depends on a particular problem type and can be one of the following: solving a linear system, optimizing non-linear embeddings or something else. We also show that the algorithm is scalable and highly effective on number of practical tasks.

Chapter 1

Introduction

Decision trees are one of the most popular learning models that have been widely used in many applications across different domains [109, 84]; and they have received much praise in machine learning and statistical literature [57, 84, 1]. Formally, decision tree is a hierarchical model (see fig. 1.1; bottom) which splits an input (or feature) space into local regions (e.g. polytope) and applies the same predictive function (like a constant) to all instances falling into the same region. Here, a region is identified by the sequence of recursive splits known as root-to-leaf path. In terms of architecture, it consists of decision nodes (having more than one child) and leaves (or terminal nodes that do not have any child). Each decision node defines a discrete split function (e.g. $x_1 > 38.5$ in fig. 1.1) which sends an input instance to the corresponding child. We repeatedly apply a split function (starting from the root) until we reach a leaf. The actual model output (e.g. predicted class) is produced by a leaf which applies its predictive function (e.g. linear, constant, etc.) to the given input. The described model is clearly nonlinear and has some unique advantageous (see fig. 1.1; top): fast training/prediction runtime, interpretability (since root-to-leaf path can be formulated as "IF-THEN" rules), handling categorical features naturally, etc. It is even the case that decision trees are preferred over more accurate methods because of these properties.

The problem of learning a decision tree model from data consists of: 1) learning a structure of a tree (maximum depth, split type, number of children, total number of nodes, etc.) and 2) optimizing over nodes' parameters (threshold value,

Characteristics	Neural Nets (FF)	SVM	Trees	k-NN, Kernels
Natural handling of categorical features	▽	▽	△	▽
Handling of missing values	▽	▽	△	△
Robustness to outliers in input space	▽	▽	△	△
Insensitive to monotone transformations of inputs	▽	▽	△	▽
Computational scalability (large N)	▽	▽	△	▽
Ability to deal with irrelevant inputs	▽	▽	△	▽
Interpretability	▽	▽	□	▽
Predictive power	△	△	▽	△

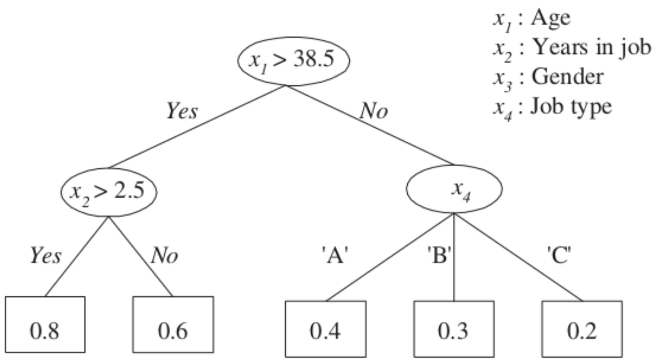


Figure 1.1: *Top*: Some characteristics of different methods taken from [57]. Keys: △ = good, ▽ = poor, □ = fair. Here, we use regular feed-forward neural nets (FF–feedforward) and conventional axis-aligned Trees. *Bottom*: Example of a (hypothetical) decision tree taken from [1]. Each root-to-leaf path can be written as a conjunctive rule constructed from decision nodes.

feature index, etc.). We will define a tree learning problem more formally in section 1.2. The rich history of decision trees (the earliest works are originated from 1950s) allowed to develop various methods and algorithms to train them. However, most of those algorithms are based on greedy recursive partition procedure (like CART [22]) which is known to be highly suboptimal. Another major problem is that most of the existing algorithms generate trees with high variance. That is, a small change in data can result to completely different trees (in terms of structure and parameters at each node). This typically leads to high error on test data (i.e., generalization error). The reason for this instability is the greedy nature of most algorithms: the error is accumulated starting from the root split down to all splits below it [57]. A recent non-greedy algorithm, *Tree Alternating Optimization (TAO)* [26, 25], may change this situation. TAO can train decision trees of arbitrary complexity (e.g. axis-aligned, oblique, neural, etc.) and guarantees monotonic decrease of an objective function over the entire decision tree, which typically leads to finding much better approximate optima than CART-type algorithms. TAO has been shown to find much better trees under a variety of losses, regularization and types of tree, as well as forests, and scales well to large datasets, see e.g. [146, 142, 148, 143, 48, 30].

A huge success of TAO in training decision trees and forests in traditional supervised learning setting motivated us to extend it to other machine learning tasks, such as semi-supervised and self-supervised learning, dimensionality reduction, clustering, etc. A particular interesting technique that appears commonly in these tasks is called *manifold regularization*. At a high level, it exploits the shape of a dataset or the geometry of the underlying data distribution to constrain the model that should be learned. For example, a common assumption is that similar instances have similar predictions. Formally, given a training dataset $\mathcal{D}$ and a target model f with parameters $\boldsymbol{\Theta}$, a learning algorithm will attempt to minimize the following regularized objective:

$$\min_{f} \; L\left(f(\mathcal{D};\boldsymbol{\Theta})\right) + \alpha\|f\|_{l} + \gamma\|f\|_{I} \tag{1.1}$$

where L is the loss function defined by a problem (e.g. cross-entropy, reconstruction error, etc.). The first regularization term $\alpha\|f\|_{l}$ is commonly referred as generalized Tikhonov regularization which penalizes the complexity of the model. As for the second term $\gamma\|f\|_{I}$, under the manifold assumption in machine learning, the data do not come from the entire input space $\mathbf{X}$, but instead from a nonlinear manifold $\mathbf{M}$. And intrinsic geometry of this manifold is used to determine the second regularization norm. In fact, there are many possible choices for $\|f\|_{I}$. Some natural choices involve a measure of smoothness of model f, which can be measured via the gradient $\nabla_{\mathbf{M}}$ on manifold $\mathbf{M}$. This gives one appropriate choice for this regularizer:

$$\|f\|_I = \int_{x\in\mathbf{M}} \|\nabla_{\mathbf{M}} f(x)\|^2 \, dp(x). \tag{1.2}$$

In practice, this integral is infeasible to compute, but can be estimated. One common choice is called a *graph prior* and it is defined as:

$$\|f\|_I = \frac{1}{N^2}\mathbf{f}^T\mathbf{L}\;\mathbf{f} \tag{1.3}$$

where $\mathbf{f}$ is a vector of values f, N is the size of dataset and $\mathbf{L}$ is the graph Laplacian, which we will define later in chapter 3. While the specific form of the manifold regularization is not relevant here, what matters is that it can be used to formulate variety of problems in machine learning. For example, it allows naturally extend supervised learning algorithms in semi-supervised learning setting as follows. We are given the dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$, where $\mathcal{D}_l = \{\mathbf{x}_n, y_n\}_{n=1}^{l} \subset \mathbb{R}^D \times \mathbb{R}$ is the labeled portion of the data, with l points, and $\mathcal{D}_u = \{\mathbf{x}_n\}_{n=l+1}^{N} \subset \mathbb{R}^D$ is the unlabeled portion, with $N - l$ points. Then our goal is to minimize the following regularized objective:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{l} L\left(f(\mathbf{x}_n;\boldsymbol{\Theta}), y_n\right) + \gamma \sum_{n,m=1}^{N} w_{nm}(f(\mathbf{x}_n;\boldsymbol{\Theta}) - f(\mathbf{x}_m;\boldsymbol{\Theta}))^2. \tag{1.4}$$

where the manifold regularization appears as weighted distances between all pair of points and can be reformulated as the graph Laplacian term shown in eq. (1.3). Its interpretation in this particular context is that the data points with different labels are not likely to be close together.

Nonlinear dimensionality reduction is another field where manifold regularization appears. In this context, they are closely related to spectral techniques and does not necessarily takes the form of graph Laplacian. For example, the elastic embedding [24] optimizes:

$$E(\mathbf{X}) = \sum_{n,m=1}^{N} \left(w_{nm}\|\mathbf{x}_n - \mathbf{x}_m\|^2 + \alpha e^{-\|\mathbf{x}_n - \mathbf{x}_m\|^2} \right) \tag{1.5}$$

where the first term encourages projecting similar patterns near each other, while the second term repels all pairs of projections. Other algorithms of this type are stochastic neighbor embedding (SNE) [58], t-SNE [127], the Sammon mapping [112], etc. Although the literature on this topic typically does not explicitly mention the name manifold regularization, but the form of the loss has similar meaning: assume smoothness of f to learn the parameters of a model. Because of this assumption, a manifold regularization can use input data to inform where the model is allowed to change quickly and where it is not.

Although TAO shows remarkable results on several supervised learning tasks, extending it to above-mentioned problems is problematic. As we will see later in chapter 2, TAO relies on separability property of the loss function, i.e., the loss should decompose over each instance. However, this is not the case with manifold regularization, where we typically have pairwise distances involved in a certain form. Therefore, TAO cannot be applied as is. Other tree learning methods have not been successful as well. Although the literature on the topic of tree-based models is immense (see [153, 128]), incorporating decision trees into manifold regularization framework has received almost no attention. One possible explanation is the difficulty of the optimization problem. As one can see, loss functions in eq. 1.4-1.5 are non-linear involving weighted pairwise distances. Minimizing an objective with such regularization is non-trivial when we have decision trees as a predictive model. It also destroys the smoothness assumption, which is now limited to closeness measure in discrete space.

1.1 Contributions

In this dissertation, we study the learning algorithms for decision trees under manifold regularization framework. We argue that this has a great practical interest and opens interesting research directions within machine learning. Furthermore, we derive a generic algorithm to solve such non-trivial optimization problems. It is based on a reformulation of the problem which requires iteratively solving two simpler problems. One problem will always involve supervised learning of a tree. The second one will depend on a particular problem type and can be one of the following: solving a linear system, optimizing non-linear embeddings or something else. Below we give a summary of every chapter in this dissertation.

- In the remaining part of this chapter we provide an overview of tree-based models and the corresponding learning problem (section 1.2). We also present an extensive review of related work on decision trees and their applicability beyond supervised learning (section 1.3).

- In chapter 2 we review the Tree Alternating Optimization (TAO), a recently introduced algorithm to train tree-based models. Using TAO is critical to the success of our approach. Therefore, we provide in-detail description of the algorithm with specific examples. To demonstrate the full performance of TAO, we provide experimental results on standard machine learning tasks (section 2.2), where TAO shows outstanding results, exceeding the performance of the state-of-the-art methods. We also show its scalability feature on extreme classification problems (section 2.3).

- Chapter 3 describes the first example where we apply manifold regularization with decision trees, namely semi-supervised learning. We first formulate an optimization problem and then propose an efficient and scalable iterative algorithm to solve it. We also discuss how to generalize this framework to other models and demonstrate our experimental findings.

- In chapter 4 we consider how to train nonlinear embeddings where out-of-sample mapping is obtained via decision trees. This can be considered as another example of applying manifold regularization with decision trees. These final two chapters suggest a generic way of training decision trees under manifold regularization, which we derive in chapter 4 as well. We also discuss experimental results with specific focus on interpretability.

- Finally, chapter 5 concludes this dissertation and discusses future research directions.

1.2 Overview of the tree learning problem

In this section, we introduce a generic tree learning optimization problem and define some useful notations which will appear throughout this dissertation. By a tree learning, we mean building a model (using a certain algorithm) based on sample data, known as training dataset, in order to make predictions or decisions. More specifically, we are given a training dataset $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \mathbb{R}^K$ of size N with D-dimensional real-valued instances and their labels (in $\mathbb{R}^K$). This is a typical *supervised learning* setup where we have an access to all labels. Please note that the exact form of labels is dictated by the nature of a problem: real-valued output for regression ($\mathbb{R}^K$), indicator of a category for classification (K), etc. When labels are not provided and we still want to obtain some insights from data, the problem is known as *unsupervised learning* in which case $\mathcal{D} = \{(\mathbf{x}_n)\}_{n=1}^{N} \subset \mathbb{R}^D$. Lastly, in *semi-supervised learning* setup, we have a mixture of both with dominant portion of unlabeled data.

Next, decision tree can be considered as a mapping applied to a single instance $\mathbf{x}$ which produces a certain output (e.g. prediction of a class). Namely, the tree's prediction $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ for an instance $\mathbf{x}$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and applying its predictor (as was described earlier). Here, we denote $\mathbf{T}$: $\mathbb{R}^D \rightarrow \mathbb{R}^K$ as the tree predictive mapping with trainable parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i \in \mathcal{N}}$ where each node (both decision and leaf) has its own trainable parameters $\boldsymbol{\theta}_i$. The nodes in a tree are indexed in set $\mathcal{N} = \{1 ... N_d\}$ where N_d is the total number of nodes. Now the question is how the routing of $\mathbf{x}$ happens? For that, we need to introduce a decision function $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathcal{C}_i$ which maps an input instance to an indicator variable denoting which child comes next. For example, if we have a binary tree, then $\mathcal{C}_i = \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$. We continue this procedure of applying $f_i(\cdot)$ recursively until we reach a leaf. An exact form of $f_i(\cdot)$ defines type of decision nodes and consequently type of a tree. Several commonly used decision node types are listed below which will be the main focus of this thesis:

- **Oblique (linear multivariate) trees** which have a hyperplane based split. That is, the corresponding decision function $f_i(\cdot)$ has a form: "go to the right child if $f_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0} \geq 0$ and to the left child, otherwise". Here, $\mathbf{w}_i, w_{i0}$ are node parameters, i.e. $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$.

- **Axis-aligned (univariate) trees**: each decision node i has a decision function "go to the right child if $f_i(\mathbf{x}) = x_j \geq b_i$ and go to the left child, otherwise", where x_j is the jth value of the input feature vector $\mathbf{x}$ and b_i (threshold value) is the parameter of node i. Axis-aligned trees are the special case of the oblique trees where $\mathbf{w}_i$ is an indicator vector for a single feature.

Finally, each leaf i (i.e., terminal node which does not have any child) has a predictor function $\mathbf{g}_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \to \mathbb{R}^K$ that produces the actual output/prediction. We will discuss the specific form of the leaves in the next sections. But a typical choice is a constant function or rarely a linear transformation followed by some non-linearity (e.g. softmax).

Using all these definitions and notations, we can state the formal optimization problem:

$$\min_{\mathbf{T},\boldsymbol{\Theta}} \; L(\mathbf{T}(\mathcal{D}; \boldsymbol{\Theta})) + \alpha \|\mathbf{T}\|_l \tag{1.6}$$

where L is the loss/objective function defined by a problem (e.g. cross-entropy, RMSE for supervised learning; reconstruction error, clustering loss for unsupervised learning, etc.) and $\alpha \|\mathbf{T}\|_l$ is a regularization term which penalizes a model (e.g. number of nodes, model parameters) to achieve a better generalization and/or to obtain more compact tree. Another common name for this term is generalized Tikhonov regularization. The user-defined hyperparameter $\alpha \geq 0$ controls the trade-off between the loss and the regularization. Note that the given minimization problem is over a *tree structure* $\mathbf{T}$ and a *tree parameters/weights* $\boldsymbol{\Theta}$. This makes it much harder to solve since the space over tree structures is a discrete subspace and the solution involves combinatorial optimization. In fact, finding optimal decision trees or even constant-factor approximations is NP-hard in most formulations of the problem [62, 54] and we will discuss this further in the next section. Therefore, it is sometimes reasonable to assume that the initial tree structure is given either by generating a random tree of depth Δ (not necessarily complete) or initializing it from some algorithms (like CART). In this case, the problem reduces to learning the parameters of a tree with the given structure by minimizing:

$$\min_{\mathbf{\Theta}} \; L(\mathbf{T}(\mathcal{D};\mathbf{\Theta})) + \alpha \sum_{i \in \mathcal{N}} \phi_i(\boldsymbol{\theta}_i). \tag{1.7}$$

Note that in this formulation, the regularization term penalizes the parameters $\boldsymbol{\theta}_i$ of each node, where ϕ_i is e.g. a norm such as ℓ_1 or ℓ_2. It worth to mention that even in this setup (fixed tree structure) the problem is still NP-hard. Furthermore, as a specific example, training a regression tree typically involves the following optimization problem:

$$\min_{\mathbf{\Theta}} \; \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{T}(\mathbf{x}_n;\mathbf{\Theta})\|_2^2 + \alpha \sum_{i \in \mathcal{N}} \phi_i(\boldsymbol{\theta}_i). \tag{1.8}$$

In the next section, we will review the related work on decision trees and discuss various attempts to approximately solve the optimization problems in eq. (1.6)-(1.8).

1.3 Related work on decision trees

Decision trees (and forests) have been extremely well studied in machine learning and statistical literature [57, 150, 76]. Although their roots are in the 1950s, they became really popular in the early 1980s. In this section, we review a literature on decision trees which includes the most popular, long-established methods and some important research frontiers. We will start with reviewing various methods and algorithms to train a single tree. Then we consider different techniques to ensemble them into a forest which are commonly used models in recent years. Unfortunately, most of these works focus on only fully supervised learning scenario. Therefore, we also list and describe methods which can train decision trees in unsupervised/semi-supervised learning setup at the end of this section.

1.3.1 Learning a single tree

Most of the papers presented in this section consider binary axis-aligned trees with constant leaves which are commonly accepted as the classical type of decision trees and they are predominantly used in many applications. That is, each decision node i has a decision function "go to the right child if $x_j \geq b_i$, go to the left child otherwise", where x_j is the jth value of the input feature vector $\mathbf{x}$ and b_i is the threshold value of the node i. A feature indicator j and thresholding value b_i are learnable parameters. As we have discussed earlier, even in such relatively simple construction, training a single decision tree on data involves searching over a huge space of tree structures (number of children, number of nodes, depth, etc.) and optimizing over parameters at each node (e.g. thresholding value) which is non-differentiable and severely non-convex problem. We also discussed that finding optimal decision trees or even constant-factor approximations is NP-hard in most formulations of the problem [62, 54]. Therefore, various (mostly approximate) methods have been proposed to tackle this issue. Based on textbooks on statistical/machine learning [16, 57, 94, 1] and specialized review papers [22, 65, 12], we group the literature on learning decision trees into the following three categories:

Greedy recursive splitting This is considered as the most popular (as of now) method for constructing trees [16, 57, 109, 110, 84]. As a convincing evidence, the leading and successful softwares for routine machine learning tasks (such as XGBoost [33], LightGBM [71], CatBoost [104], scikit-learn [100], etc.) use this type of procedure to construct decision trees. Historically, this approach is originated from CART, proposed by [22], and other related algorithms, such as ID3 [105], C4.5 and C5.0 [106], etc. Multiple public and commercial implementations of these exist, e.g. [121, 75, 100]. There are more recent developments [139, 109, 110, 84] which aim to improve over previous algorithms by adding a number of heuristics. Although many variations exist that fall into this category, many of them capitalize on the top-down approach to build a tree. That is, we start from the root and recursively split a node into two or more children based on solving a "purity" optimization problem (e.g. minimizing Gini index or entropy), which seeks a partition of the space such that the training instances reaching each child belong to one class or as few classes as possible (occasionally this is called a variance-based criterion). The procedure is recursively repeated until each leaf contains instances from the same class (or some other stopping criterion is satisfied). Note that once we split the node, it is fixed and it will not be modified afterwards (thus the name greedy). Except the pruning phase, where the resulting (usually large) tree is optionally pruned in order to reduce overfitting. This is done by removing subtrees to optimize a cost-complexity trade-off. However, the node parameters are still fixed and not allowed to change. This greedy nature of the algorithm and the fact that it does not optimize an objective function directly over the tree parameters (but rather uses indirect measures like purity) makes such trees produce severely *suboptimal trees* [57].

Approximate brute force search These approaches attempt to find an optimal decision tree (which is an NP-hard problem) using a variety of heuristic methods or using constrained optimization formulation (mainly via mixed-integer programming). Bennett [9, 10] formulates a large linear programming over a tree with the fixed structure for binary classification problem. The problem is now convex and the global optimum can be found. However, the resulting LP is so large that the problem size grow exponentially with the tree and dataset sizes (trees with only 4 internal nodes were used experimentally). Another group of methods [4, 123] focus on efficient enumeration over a space of small trees (up to depth 3). Motivated by the recent advances in branch-and-bound techniques implemented in commercial solvers (such as Gurobi [52]), Bertsimas and Dunn [12] formulate the tree optimization as a mixed-integer optimization (MIO). This is done by introducing binary variables and carefully designed constrains to encode a tree structure. Then, it is straightforward to apply state-of-the-art MIO solvers to solve the problem. In practice, the method is only applicable to small datasets and small trees (up to depth 4). More recent work from [152] attempts to make this MIO formulation scalable to larger datasets (but still using small trees) by selecting a subset of points. The next series of papers [61, 82] follow a similar approach but use a custom branch-and-bound algorithm. Moreover, they propose a number of additional techniques (e.g. using a dynamic programming) to accelerate the training procedure. However, they are restricted to binary inputs and outputs only (i.e. boolean functions), which requires discretization of the continuous features. *The main problem with all of these methods is that they still have an exponential time complexity in most cases.* Since running them to completion for larger trees and/or larger datasets is intractable, they are almost never used for practical applications. It is still possible to stop earlier with an approximate solution. However, there is no possible way of estimating how good that approximate solution w.r.t. global optima. Although some works provide a certain bounds on the estimate [82].

Non-greedy, global optimization algorithms In contrast to greedy methods and similar to brute force search methods, these type of algorithms focuses on a joint optimization of the decision tree nodes under a global objective function. However, various approximations are used instead of directly addressing an NP-hard problem. One approach is to ignore the discrete nature of the decision tree and make it *probabilistic (or soft)*, where an input is routed to each leaf with a certain probability. Then the optimization becomes amenable to gradient-based methods. Another possible approach is to learn the parameters of a tree by maximum likelihood optimization with an Expectation Maximization (EM) algorithm [40]. The method is originated from the hierarchical mixtures of experts (HME), proposed by Jordan and Jacobs [68]. It is still possible to make the resulting tree hard again by following a path with the highest probability (or choosing a child with the highest probability). Obviously, such approximation introduces suboptimality. Norouzi et al. [98, 97] directly optimize a tree with discrete decision nodes by formulating a convex-concave upper bound on the tree's empirical loss and optimize that loss using stochastic gradient descent (SGD). The initial tree structure must be provided (e.g. initialized from CART). The use of SGD enables efficient optimization for large scale datasets. However, the method does not have guarantees to decrease the desired loss function (e.g. misclassification loss) over a decision tree and may even marginally worsen an already induced tree. The *TAO algorithm*, proposed by Carreira-Perpiñán et al. [26, 25], optimizes a decision tree with predetermined structure and can be applied to minimize the desired objective function (e.g. classification, regression). Each iteration of the TAO algorithm guarantees monotonic decrease of the objective function and it generalizes well by imposing various regularizations (e.g. ℓ_1-regularization). TAO has been successfully applied to train a single decision tree of increasing complexity (axis-aligned, oblique, neural trees) [147, 143] as well as to construct various tree ensembles

[142, 30]. We provide a detailed description of the algorithm in section 2. To sum up, non-greedy algorithms show promising results and have been gaining popularity in recent years. Though they are less popular compared to greedy recursive partition algorithms, they have a huge potential to replace them in the future.

1.3.2 Combining multiple trees into a forest

The true power of decision trees comes when combining (or ensembling) them into a forest. These are among the most successful and widely used of all machine learning models (in isolation or combined with other models, such as deep nets), and they are well-studied in the statistical and machine learning literature [57, 21, 23]. They are widely used in many applications, such as data mining [129], computer vision for body and visual tracking [36, 140], face and object detection [117, 130], shape recognition [2] and many other applications [35].

The success of forests is due to their ability to achieve low bias and low variance by combining weakly correlated trees, for which different ensembling mechanisms exist. The main approaches for ensembling different trees are based on bagging [20], where individual trees are trained independently on bootstrap samples of the data; or on boosting [113, 114], where individual trees are trained sequentially on the whole data but with adaptively weighted instances. The final prediction is usually obtained by majority voting (for classification problems) or averaging (for regression problems). The are many variations of bagging and boosting. Random Forests [19, 15] combine bagging with choosing random feature subsets at each node when considering candidate splits to maximize a purity measure. Extremely randomized trees (Extra-Trees) [49] combine bagging with random splits. AdaBoost [46, 114] trains each tree sequentially by reweighting the samples, i.e. it assigns higher weights for misclassified samples and lower weights for the correctly classified ones. Variation of boosting called gradient boosting [47] which approximates functional gradient of the loss function and trains each tree to minimize a distance to that gradient. It has attracted much attention in recent years, particularly with the availability of efficient implementations that can scale to large datasets [33, 71]. Other variations include random subspace forests [59] and rotation forests

[108]. After several decades of active research on decision tree ensembles, it is fair to say that the methods that at present are recognized as the state-of-the-art are Random Forests, AdaBoost and gradient boosting. What is common to all these approaches is the way they construct the individual trees: they use a top-down induction algorithm such as CART that recursively splits nodes which are known [57] to be highly suboptimal tree optimizers. This has led to a perception that decision trees are generally low-accuracy models in isolation [57, p. 352], although combining a large number of trees does produce much more accurate models.

Other methods exist which combine the individual tree learning with the ensembling. Schulter et al. [117, 116] use a variation of the boosting algorithm which alternates between: greedily adjusting depth of a tree (or ensemble of trees) using some purity criterion and updating weights for each training sample. Zhou and Feng [151] apply cascading technique for random forests of a large size to achieve comparable performance against neural nets for some tasks. Several works [45, 135] construct ensembles using nested dichotomies for multiclass problems. The idea is to learn a multiclass problem using several binary classifiers (as an alternative to one-vs-all or ECOC). The training algorithm in all of these methods is quite more complicated and it is not clear that it translates into more accurate or compact forests, although they may be preferable in particular applications. Some methods experiment with different losses, such as robust losses [80]. Finally, some techniques exist to take an existing forest and postprocess it. Pruning a forest by removing redundant trees can be done greedily with forward selection [150]. This can often reduce the size of a forest with little degradation of its accuracy. Also, it is possible to optimize jointly the constant predictors at the leaves of all trees (keeping the decision nodes and tree structure the same) and increase the accuracy a bit [107].

1.3.3 Extreme classification

Classification problems having thousands or more classes naturally occur in some ML problems, e.g. in Natural Language Processing (NLP): language modeling, document classification, etc. Handling all classes in regular softmax or in one-vs-all fashion would be very slow at inference time, because every class score must be calculated to find the top class. We decided to provide a separated literature review on extreme classification since decision trees have been actively used in this area [7, 37]. Nevertheless, traditional axis-aligned decision trees, such as C4.5 or CART, have very low accuracy [34]. Nested dichotomies [45] rely on a tree structure to divide a set of classes into two disjoint subsets and learn a binary classifier to separate them. However, human expertise is necessary to obtain a tree structure and class assignments. Additionally, the total error of the model accumulates over the depths since there is no way to refine binary classifiers once split is performed. More recent works that are specifically designed to cope with large number of classes [13, 7] employ similar idea but take into account class distributions to generate a tree structure. Other tree-based approaches include global or partial optimization over parameters of a tree. For instance, Daume et al. [37] propose to use a fixed structured tree where each node has much smaller sized linear multi-class classifier. Sun et al. [118] extend this work by allowing a tree structure to grow. Other works capitalize on generating "perfectly" balanced trees to guarantee logarithmic inference time [67, 34]. Optimizing a tree parameters in these methods is typically done by approximating gradient information in a certain way (possibly in "online" fashion). Other tree-based methods exist with more focus on large scale extreme multi-label classification and ranking [103, 14].

In the context of NLP, most of the above-mentioned methods are applicable in the number of practical applications, such as large-scale document classification and language modeling. Moreover, there are methods that are specifically designed for language modeling tasks where vocabulary size can be very large and it demands efficient computation of the softmax outputs. Hierarchical softmax (HSM) [51] is an approximation which employs a "soft" decision tree with linear nodes to address this issue. HSM has been actively used in the problem of learning distributed representations of words [8, 91] where it can be jointly trained with neural nets of various complexity [93]. Follow up works on this topic [92, 88] propose various initializations for the tree structure (e.g. random, Huffmann tree, etc.). Although the training of HSM can be efficiently done using specific loss functions, but during prediction time, input follows all children with a certain probability which brings no speedup compared to the plain softmax. It is still possible to transform a soft tree back into a "hard" tree once training is done (by choosing a child with the highest probability at each split). For example, a recent work from Han et al. [53] apply a similar approach. Recently, certain pruning mechanisms have been proposed as an alternative approach to speed up the prediction time [17].

1.3.4 Beyond supervised learning

Although the literature on this topic is immense, the study of decision trees beyond supervised learning is somewhat limited. For instance, *semi-supervised learning (SSL)* can be considered as an extension to the regular supervised learning, which incorporates both labeled and unlabeled sets of data into a model training. This specific area has received almost no attention in context of decision trees. Probably, this is due to difficulty of the optimization problem. Most SSL methods are based on adding a graph prior (or similarity matrix) as regularization to exploit the geometry of the underlying data distribution [6, 149, 154, 155]. Minimizing an objective with such regularization is non-trivial when we have decision trees as a predictive model. That said, several attempts have been made to apply SSL for trees. Levatic et al. [78] modify a splitting criterion for recursive tree induction by taking into account unlabeled data. That is, a splitting score for each [feature, threshold] pair consists of two parts: the traditional purity score (e.g. Gini index) for labeled data and a "clustering" score for all available data. Tanha et al. [119] apply a self-training framework [138] to train classification trees with labeled and unlabeled data. Note that self-training is a generic framework that can be applied with any classifier which predicts class probabilities. It trains a classifier iteratively where the first iteration includes only labeled data. After that, it uses model predictions and adds the most confidently predicted instances to enrich the amount of supervision. Another direct approach is to apply a label smoothing technique [154] to propagate label information to all data and then fit a tree. This method has been recently reused within the graph neural networks framework to train boosted trees [32, 66]. Finally, Kemp et al. [72] proposed a

Bayesian approach where unlabeled data assist in inferring a latent tree structure from a distribution over trees. The limitations of this approach are that it covers classification tasks only, and it uses Markov Chain Monte Carlo to sample from the distribution over trees, which leads to scalability issues.

Dimensionality reduction (DR) (more specifically, non-linear DR) is another area where decision trees have been rarely used. The natural application in this scenario is to use trees as low dimensional mappings and focus on interpretability. The problem with this is the same as in SSL: difficulty of the optimization problem which involves DR objective. In this regard, some work has proposed using soft decision trees in an autoencoder framework (i.e., training the decoder-encoder to minimize the reconstruction error of the training instances) [64]. This makes the optimization simple, since soft trees are differentiable, but soft trees are hard to interpret, more so if the input instance has many features (as is to be expected in DR). We are aware of very little work, if any, regarding learning interpretable mappings for nonlinear embeddings such as t-SNE. An obvious approach would be to optimize the embedding and then fit to it a sparse linear or otherwise nonlinear but interpretable mapping (e.g. decision trees), but as we note later this is suboptimal. Since the form of our projection mapping is a tree of linear mappings, this can also be regarded as a form of local DR. Examples of this are mixtures of factor analyzers [50] or of PCAs [70, 122] and related models [111].

Chapter 2

The Tree Alternating Optimization Algorithm

This chapter provides an overview of the Tree Alternating Optimization (TAO) algorithm, proposed in [26] and later extended in [146, 142, 148, 143, 48, 30, 25], to train classification and regression trees. This is a central framework used in the remaining chapters, but for now we consider a fully supervised learning scenario and extend it later. The TAO algorithm takes as input an initial tree and produces a new tree with the same or smaller structure but new parameter values (weights), which provably lower or leave unchanged the desired objective function. This can be applied to decision trees of arbitrary complexity (axis-aligned, oblique trees, etc.) and to any decomposable loss function. We start by deriving a generic TAO framework (section 2.1) and provide experimental results showing practical importance of TAO. Specifically, sections 2.2-2.3 demonstrate superiority of tree-based models trained via TAO compared to state-of-the-art methods. We also show that the algorithm is scalable for large datasets. Most of the notations in this chapters are taken from chapter 1.2 and we refer a reader to it for a generic tree learning problem setup.

2.1 Overview of the TAO algorithm

We start by defining a rooted directed binary tree, i.e., each decision node has exactly two children. We assume that the initial tree structure is given either by generating a random tree of depth Δ (not necessarily complete) or initializing it from other algorithms (like CART). Nodes in a tree are indexed in set $\mathcal{N} = \{1...N_d\}$, where N_d is the total number of nodes. Each node (both decision and leaf) has learnable parameters $\boldsymbol{\theta}_i$ and the total set of parameters of a tree is $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i \in \mathcal{N}}$. Each decision node i has a decision function $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathcal{C}_i$, where $\mathcal{C}_i = \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$, sending an instance $\mathbf{x}$ to the corresponding child of i. Although TAO can be extended to more general types of nodes, we focus on two types of decision nodes for simplicity:

- **Oblique (linear multivariate) trees** which have a hyperplane based split. That is, the corresponding decision function has a form: "go to the right child if $\mathbf{w}_i^T\mathbf{x} + w_{i0} \geq 0$ and to the left child, otherwise". Here, $\mathbf{w}_i, w_{i0}$ are the node parameters, i.e. $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$. A particular type of oblique trees are called *sparse oblique* trees, which use small subset of features at each node. As we describe later, on can obtain such trees by enforcing ℓ_1 penalty on nodes.

- **Axis-aligned (univariate) trees**: each decision node i has a decision function "go to the right child if $x_j \geq b_i$ and go to the left child, otherwise", where x_j is the jth value of the input feature vector $\mathbf{x}$ and b_i (threshold value) is the parameter of the node i. Axis-aligned trees are the special case of the oblique trees where $\mathbf{w}_i$ is an indicator vector for a single feature.

Each leaf (terminal node which does not have any child) i has a predictor function $\mathbf{g}_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathbb{R}^K$ that produces actual output. We shall discuss the specific form of $\mathbf{g}(\cdot)$ later in this section (see the "reduced problem over a leaf"). The prediction of a tree $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ for an input $\mathbf{x}$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and applying its predictor ($\mathbf{g}_i$).

We consider the problem of learning the parameters of a classification/regression tree of given structure by minimizing:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha \sum_{i \in \mathcal{N}} \phi_i(\boldsymbol{\theta}_i), \tag{2.1}$$

given a training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \mathbb{R}^K$. We emphasize that we consider T as a parametric model with trainable weights in each node (like fixing a neural net architecture and optimizing over its parameters). The loss function $L(\mathbf{y}, \mathbf{z})$ measures the disagreement between two vectors $\mathbf{y}$ (ground-truth label) and $\mathbf{z}$ (tree prediction). For instance, the squared error $\|\mathbf{y} - \mathbf{z}\|_2^2$ is commonly used for regression problems, whereas 0/1 loss $I(\mathbf{y} \neq \mathbf{z})$ is a standard for classification (although it is possible to use other losses, such as hinge loss, absolute deviation, etc.). The regularization term penalizes the parameters $\boldsymbol{\theta}_i$ of each node, where ϕ_i is e.g. a norm such as ℓ_1 (typically leads to sparsity) or ℓ_2. The user-defined hyperparameter $\alpha \geq 0$ controls the tradeoff between the loss and the regularization. We define a *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$ of node i (decision node or leaf) as the training instances that reach i given the current tree parameters. That is, to compute the reduced set of each of node i's children we pass each instance $\mathbf{x}_n$ in i's reduced set $\mathcal{R}_i$ through i's decision function f_i, and add $\mathbf{x}_n$ to the resulting child's reduced set.

We say that $\mathcal{S} \subset \mathcal{N}$ is a set of non-descendant nodes if $\forall i, j \in \mathcal{S}$ neither i is a descendant of j nor j is a descendant of i in the tree graph. For example, all nodes at the same depth is the set of non-descendant nodes. Further, we assume that the parameters are not shared across nodes: $i, j \in \mathcal{N},\ i \neq j \Rightarrow \boldsymbol{\theta}_i \cap \boldsymbol{\theta}_j = \varnothing$. Next, we briefly summarize the following three fundamental theorems for TAO which allow us to minimize eq. (2.1) over a tree $\mathbf{T}$ parameters. Details and proofs can be found in [26] or in suppl. mat. of [142, 25] (see theorems 1.1-1.3).

Separability condition This theorem states that by picking any subset of nodes $\mathcal{S}$ that are non-descendants and fixing the parameters of the remaining nodes ($\boldsymbol{\Theta}_{\text{rest}}$), the loss function in eq. (2.1) *separates* over parameters of the nodes in set $\mathcal{S}$. That means we can train each node in the set $\mathcal{S}$ independently from each others.

Reduced problem over a decision node Separability condition above allows us to train each node in $\mathcal{S}$ independently. A single decision node optimization can be intuitively explained using fig. 2.1. Suppose we optimize over the node $i = 2$, which means we fix all parameters of a tree except $\boldsymbol{\theta}_2$ and we need to only consider the reduced set $\mathcal{R}_2$. That node has two children $\mathcal{C}_2 = \{4, 5\}$ with the corresponding subtrees: $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ and $\mathbf{T}_5(\mathbf{x}; \boldsymbol{\Theta}_5)$ for the left and right child of node i, respectively. If node $i = 2$ sends an instance $\mathbf{x}$ to the left child, then we compute $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ and return its prediction. Note that $\boldsymbol{\Theta}_4$ is fixed which means $\mathbf{x}$ follows a unique path starting from node 4 down to some leaf. Then we apply a predictor at that leaf to compute the output. The function $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ does not depend on $\boldsymbol{\theta}_2$. Similar arguments hold for the right child. Therefore, the loss term $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta}))$ can take only one of two possible values for all $\mathbf{x}_n \in \mathcal{R}_2$: $l_{in,4} = L(\mathbf{y}_n, \mathbf{T}_4(\mathbf{x}_n; \boldsymbol{\Theta}_4)) \in \mathbb{R}$ if $f_2(\mathbf{x}_n; \boldsymbol{\theta}_2) = 4$ (the left child is chosen) and $l_{in,5} = L_n(\mathbf{T}_5(\mathbf{x}_n; \boldsymbol{\Theta}_5)) \in \mathbb{R}$ if $f_2(\mathbf{x}_n; \boldsymbol{\theta}_2) = 5$ (the right child is chosen), where the decision function for node 2 is f_2: $\mathcal{X} \rightarrow \mathcal{C}_2$. Then, define a function l_{in}: $\mathcal{C}_i \rightarrow \mathbb{R}$ as $l_{in}(z) = L(\mathbf{y}_n, \mathbf{T}_z(\mathbf{x}_n; \boldsymbol{\Theta}_z))$, where $z \in \mathcal{C}_i$ is any child of i and $\mathbf{T}_z(\cdot; \boldsymbol{\Theta}_z)$ is the predictive function for the subtree rooted at z. The function l_{in} gives the loss value incurred by each of the two children of node i. Since we have only two children, our problem reduces to encouraging a decision function f_2 to send $\mathbf{x}$ to the "best" child (either left or right). Suppose our best child is denoted as $\overline{y}_{in}$ and we want to send $\mathbf{x}$ to that child (ideally). In order to achieve this, define a function $\overline{L}_{in}$ that satisfies $\overline{L}_{in}(\overline{y}_{in}, y) = 0$ if $y = \overline{y}_{in}$, and $\overline{L}_{in}(\overline{y}_{in}, y) > 0$ otherwise, hence it is a weighted 0/1 loss function with "ground-truth" label $\overline{y}_{in}$ (the best child). For example, if $\mathcal{C}_i = \{\texttt{left}, \texttt{right}\}$ and $l_{in}(\texttt{left}) > l_{in}(\texttt{right})$, then $\overline{y}_{in} = \texttt{right}$, $\overline{L}_{in}(\texttt{right}, \texttt{right}) = 0$ and $\overline{L}_{in}(\texttt{right}, \texttt{left}) > 0$. This leads us to the following weighted binary classification problem over a single decision node i:

$$\min_{\boldsymbol{\theta}_i} \overline{E}_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} \overline{L}_{in}(\overline{y}_{in}, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i). \tag{2.2}$$

Optimizing the weighted 0/1 loss above over a hyperplane is an NP-hard problem even for the unweighted case [102, 60] (except for axis-aligned hyperplanes, which can be solved exactly and quickly by enumeration over the features). How-

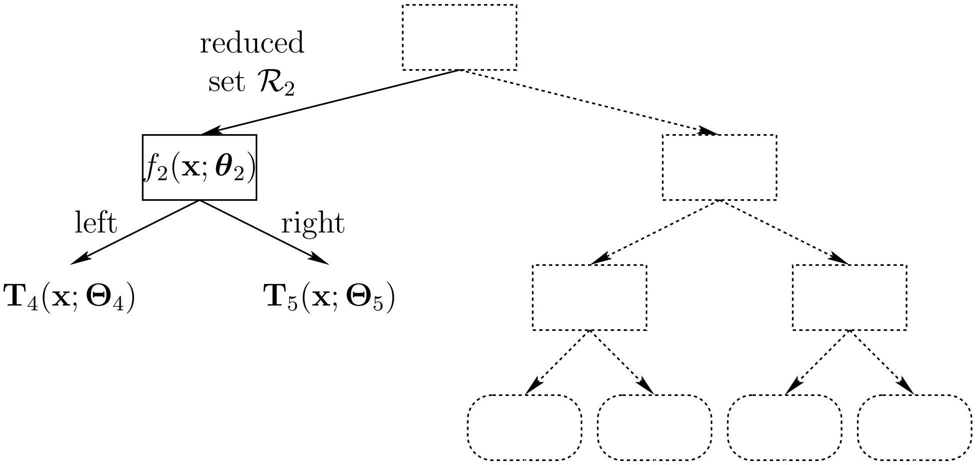

reduced
set $\mathcal{R}_2$
$f_2(\mathbf{x};\boldsymbol{\theta}_2)$
left
right
$\mathbf{T}_4(\mathbf{x};\boldsymbol{\Theta}_4)$
$\mathbf{T}_5(\mathbf{x};\boldsymbol{\Theta}_5)$

Figure 2.1: Schematic representation of the optimization over node 2 (a decision node, with parameters $\boldsymbol{\theta}_2$) in an example tree. The left and right subtrees of node 2 behave like two fixed predictor functions $\mathbf{T}_4(\mathbf{x};\boldsymbol{\Theta}_4)$ and $\mathbf{T}_5(\mathbf{x};\boldsymbol{\Theta}_5)$ which produce an output for an input $\mathbf{x}$ when going left or right in node 2, respectively. Only the training instances that reach node 2 under the current tree (the reduced set $\mathcal{R}_2$ of node 2) participate in the optimization. This figure is obtained from [25].

ever, good approximate solutions can be obtained efficiently by using a convex surrogate loss. In most of our experiments, we use the logistic loss with an ℓ_1 regularizer (implemented in LIBLINEAR [43]), which generate *sparse oblique trees* if case when we have linear nodes.

Reduced problem over a leaf This can be derived in a similar fashion as we did for decision nodes. However, the problem is now much easier since leaves do not have any children. Clearly, leaves are non-descendant w.r.t each others and we can apply *separability condition.* By fixing the remaining parameters of a tree, we can train each leaf independently. Note that $\mathbf{x}$ follows root-to-leaf path and the actual prediction is given by a leaf (a predictor function $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathbb{R}^K$). Therefore, tree output from eq. (2.1) can be replaced by the output of a leaf. But that leaf operates only on a subset of points reaching that particular leaf (its reduced set $\mathcal{R}$). Therefore, the solution for any leaf i is the minimization of (2.1) over its parameters $\boldsymbol{\theta}_i$ on a reduced set $\mathcal{R}_i$: it corresponds either to fitting a K-

class classifier for classification problems or fitting a regressor for regression tasks. As for the specific form of $\mathbf{g}(\cdot)$, the following two examples are commonly used:

- **Constant leaves**. The predictor takes form $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i) = \mathbf{w}_i$, i.e., it returns the same value independently from the input $\mathbf{x}$. During training, the solution for the constant leaf optimization can be found exactly (usually) and its form depends on type of problems and form of an objective. If the loss function is the mean squared error (e.g. in regression problems), then the solution would be taking the average of the ground truth values of the reduced set $\mathcal{R}_i$ in leaf i. Similarly, for classification problems where 0/1 loss is usually used, the solution is given by majority voting (i.e., taking the most frequently observed $\mathbf{y}$) on a reduced set $\mathcal{R}_i$.
- **Linear leaves** The predictor takes form $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i) = \mathbf{W}_i\mathbf{x}+\mathbf{w}_i$. Additionally, for classification problems, we need to apply a softmax function to normalize the resulting vector and obtain a probability distribution over K classes. During prediction, we need to take $\arg\max$ on softmax outputs to obtain a predicted class. During training, we solve an ℓ_1-penalized linear regression (Lasso [55]). For classification problems, we solve a logistic regression ($K = 2$) or linear softmax (for $K > 2$ classes) classifier (both with ℓ_1-regularization).

Algorithm Although the previous derivation looks complicated, the resulting algorithm is simple to understand (see Algorithm 1): we repeatedly train one subset of nodes and fix all the rest, until convergence. We can parallelize training of nodes that are not descendants of each others (from the separability condition). In this paper, we do this in breadth-first search (BFS) order, i.e., we update all nodes at the same depth; one pass over the entire tree defines one TAO iteration. Updating a node requires solving its reduced problem via a binary classifier (decision node) or regressor/K-class classifier (leaf). After each iteration, the objective (2.1) decreases or stays unchanged [26, 25]. We stop iterating when eq. (2.1) changes little or we reach a set number of iterations. Finally, we can remove dead subtrees from the tree, i.e., nodes receiving no training instances. This is particularly likely

Algorithm 1: TAO algorithm to train a single decision tree.

```
Result: trained tree $\mathbf{T}(\cdot;\boldsymbol{\Theta})$
input
training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$;
  initial tree $\mathbf{T}(\cdot;\boldsymbol{\Theta})$ of depth $\Delta$ (either random or pretrained);
repeat
    for depth $d = 0$ to $\Delta$ do
        for $i \in$ nodes at depth $d$ do
            if $i$ is a leaf then
                $\boldsymbol{\theta}_i \leftarrow$ fit a regressor/classifier (constant, linear, neural net,
                  etc.) on a reduced set $(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{R}_i$;
            else
                generate a pseudolabel $\overline{y}_n$ and weight $|l_{left}(\mathbf{x}_n) - l_{right}(\mathbf{x}_n)|$
                  for each instance $\mathbf{x}_n \in \mathcal{R}_i$;
                $\boldsymbol{\theta}_i \leftarrow$ fit a weighted binary classifier to minimize eq. (2.2);
            end
        end
    end
    update $\mathcal{R}_i$ for each node;
until convergence occurs or max number of iterations;
postprocessing: remove dead or pure subtrees;
```

to happen with an ℓ_1 regularizer on the decision nodes or leaves, which encourages weights to become zero (a decision node with $\boldsymbol{\theta}_i = \mathbf{0}$ creates a dead subtree). *We emphasize that, unlike in traditional tree induction algorithms such as CART or C4.5, with TAO we do not grow a tree greedily.* Instead, we assume a parametric tree with a given structure, just as when we train a neural net we choose an architecture and then optimize its parameters. That said, the final tree structure can be smaller than the initial one (due to regularization penalty), similarly to pruning a deep net.

2.2 TAO trees in action: a case study on supervised learning

To show the full performance of TAO algorithm on standard classification and regression problems, we conduct a set of experiments on real world data. We denote a single decision tree trained with TAO as *TAO tree* and consequently *TAO forest* to mean a forest of TAO trees. Out of many possible ways to ensemble decision trees described in section 1.3.2 (e.g. bagging, boosting, etc.), we choose a simple one: we train each TAO tree independently on a random subset of M samples of the available training data (N instances). A particular, simple case of this is *bagging*, where the subset is a bootstrap sample ($M = N$ but sampled with replacement; [20]). Each TAO tree was trained using Algorithm 1 and as an initialization, we use a complete binary tree of depth Δ and random node parameters (each node's weight vector has Gaussian (0,1) entries, and then we normalize the vector to unit length). Since we train each TAO tree independently, we can train them in parallel, just as with random forests. Although our TAO algorithm works with axis-aligned trees, in this section we report the results on *oblique trees*. These are more powerful, since they can better model correlations between features. Regarding the leaf predictors, we use constant (denoted as *TAO-c*) and linear leaves (denoted as *TAO-l*). We refer to the previous section for details of each type. Empirically we found that TAO forests with linear leaves (TAO-l) are the clear winners in both accuracy and forest size. The forest prediction (once training is done) is the average of its trees' predictions in case of regression. For classification, it is obtained by majority voting (TAO-c) or by averaging class probabilities (TAO-l).

2.2.1 Experimental setup

We implemented TAO in Python 3.x with process level parallel processing. For training a single TAO tree, we take as initial tree a complete binary tree of given depth (Δ in the tables) with random parameters at each node. In a forest setup, we train each TAO tree on a 90% random sample of the training data using 20-40

Classification					Regression				
Dataset	N_{train}	N_{test}	D	K	Dataset	N_{train}	N_{test}	D	K
Letter	16 000	4 000	16	26	cpu_act	4 915	3 277	21	1
MNIST	60 000	10 000	784	10	ailerons	7 154	6 596	40	1
Char74k	66 707	7 400	64	62	CT slice	42 800	10 700	384	1
RCV1	15 564	518 571	47 236	53	YearPred	463 715	51 630	90	1

Table 2.1: Datasets used in our experiments (left for classification and right for the regression): number of points for training and test (N_{train}, N_{test}), number of features D, number of classes K (or output dimensionality for regression problems). Companion papers [142, 30] contain more datasets and extended results.

iterations (depending on problem size). We tune its sparsity hyperparameter λ but usually it is a small value (e.g. $\lambda = 0.01$). We use scikit-learn [100] to perform individual node optimization described in Algorithm 1. Specifically, we use its internally implemented version of the coordinate descent algorithm for Lasso [55], LIBLINEAR [43] for logistic regression, SAGA [39] for the linear softmax classifier at leaves.

We perform extensive evaluations of TAO forests across standard classification and regression benchmarks to show effectiveness of our proposed method. Table 2.1 summarizes the characteristics of the datasets (see details in Appendix B). We compare TAO forests with the state-of-the-art tree ensembling algorithms: Random Forests (RF) [19], Extra-Trees (ET) [49], AdaBoost [46] (all using the Python scikit-learn implementation [100]); and gradient boosting [47] (using the highly optimized XGBoost implementation [33]). We explored as best as we could their hyperparameters, often improving over reported results in the literature (for the same dataset and method, e.g. for RF in several datasets in [116]). In particular, we tried different choices of the number of trees T and maximum depth Δ. We do not restrict the `max_depth` hyperparameter for RF and ET, and allow each tree to grow fully, as is recommended for random forests [19]. But we do tune this hyperparameter for XGBoost and AdaBoost. We also compare with published results of some recent forest algorithms: Alternating Regression/Decision Forests (ARF,ADF) [116, 117], Adaptive Neural Trees (ANT) [120], Shallow Neural Decision Forests (sNDF) [74], Oblique Random Forests [87], Globally Induced

Forest (GIF) [5], Consistent Random Forest (cRF) [41], and Refined Random Forest (rRF) [107]. Finally, we also give the result of training a single CART tree [22] for reference. Reported errors for regression are root mean squared error (RMSE) $E = \sqrt{\frac{1}{NK} \sum_{n=1}^{N} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2}$ and 0/1 loss $E = \frac{1}{N} \sum_{n=1}^{N} I(\mathbf{y}_n \neq \hat{\mathbf{y}}_n)$ for classification, where N is sample size, K is output dimension, and $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the ground truth and predicted values, respectively. We report the mean error (training and test) and standard deviation over 5 independent runs. Additionally, we report the total number of parameters (#pars.) and (estimated) FLOPS for inference, see Appendix A for details on how we estimate them for each model.

2.2.2 Results: TAO versus state-of-the-art

Tables 2.2-2.3 show the results (sorted by decreasing test error). Our results for ET, RF, AdaBoost and XGBoost are in overall agreement with previous works [116, 107] or sometimes better. Which of them has the highest accuracy depends on the dataset, although (with well-set hyperparameters and especially with sufficiently many trees) they generally are close to each other. ETs and RFs are simplest to use in terms of hyperparameter choice and have extremely high training speed. XGBoost and particularly AdaBoost take much longer to train. They also generate forests with many more trees if the output is high-dimensional (K times more trees if there are K outputs).

Next, consider the case of a single TAO tree ($T = 1$ in the tables). In many cases, a single TAO tree *already shows a decent performance compared with state-of-the-art forest methods*, exceeding the accuracy of CART and some forests (e.g. on RCV1 data). This is particularly true if the TAO tree has linear leaves (TAO-l). The accuracy margin becomes extreme in high dimensional classification problems (e.g. RCV1) and/or some of the regression problems (e.g. SARCOS, YearPredictionMSD). All these results show that TAO trains high quality decision trees which shows the power of optimization in the tree learning problem.

Let us now consider the case of TAO forests ($T > 1$). TAO forests with constant leaves (TAO-c) already achieve higher accuracy than most forest methods. However, we will focus on TAO trees with linear leaves (TAO-l), which we find always beat TAO-c and use smaller trees. We see that *TAO-l has the lowest test error in all datasets, often by a considerable margin over the other forest methods.* Also, *TAO-l forests have few, shallow trees.* TAO-l forests use few trees (up to 30 in all datasets), much less than the other forest methods, which need 100s or 1000s of trees to achieve their best accuracy and yet cannot match the accuracy of TAO-l. TAO-l forests also use shallow trees (up to 7 in most datasets), far shallower than the other forest methods, whose depth can exceed 100. This is because such methods use axis-aligned trees which are typically unbalanced and grown very deep. The TAO-l trees are mostly complete but pruning of nodes does occur during TAO training.

Finally, Tables 2.2-2.3 also report the number of parameters and inference time. We provide a detailed explanation on how we estimate them in Appendix A. Most forest methods use axis-aligned trees, where each decision node uses a single input feature, while TAO forests use oblique trees in these experiments. Although oblique trees are shallower and their forests require fewer trees, each oblique node uses $D+1$ parameters (weight vector and bias at each node) while an axis-aligned node uses just 2 (feature index and bias). However, our TAO oblique trees use fewer than $D+1$ parameters because we run TAO with ℓ_1 penalty which imposes sparsity. Moreover, this leads to pruning of some nodes without hurting much the model performance. As shown in the tables, *when we compare the most accurate TAO forests with the most accurate AdaBoost, XGBoost or Random forests, the TAO forests usually have lower (or comparable) number of parameters and FLOPS.*

Conclusion Random Forests (closely followed by AdaBoost and gradient boosting) have long been considered as the best off-the-shelf classifiers and regressors, due to their ease of use and high accuracy. However, these methods rely on heuristic tree learning algorithms such as CART. We show that, by using the recently proposed TAO algorithm to learn each tree instead of CART, and by ensembling trees trained on bootstrapped or random data samples, we obtain smaller forests with consistent, significant improvements in accuracy. This is a remarkable result in that 1) it has immediate practical application in machine learning, computer vision and other areas, and 2) it opens new research directions in ensemble learning

	Forest	E_{test} (%)	#pars.	FLOPS	T	Δ
MNIST	CART	12.11±0.04	5 957	(50)	1	50
	TAO-c	5.25±0.20	24k	1 163	1	8
	TAO-l	5.07±0.13	16k	1 194	1	5
	AdaBoost	2.96±0.05	5.9M	(29k)	1k	30
	RF	2.84±0.06	10M	(35k)	1k	48
	sNDF [74]	2.80±0.12	22M	22M	80	10
	ADF [117]	2.71±0.10	(3.6M)	(2 500)	100	25
	TAO-c	2.31±0.08	1.2M	52k	40	8
	XGBoost	2.17±0.00	540k	(57k)	10k	30
	rRF[107]	2.05±0.02	(160k)	(2 500)	100	25
	TAO-l	2.02±0.06	475k	38k	30	6
Letter	CART	13.06±0.15	2 985	(27)	1	27
	TAO-c	9.59±0.31	9 904	111	1	11
	TAO-l	6.60±0.33	6 449	192	1	6
	XGBoost	4.00±0.00	551k	(124k)	26k	30
	ADF [117]	3.52±0.12	(960k)	(2 500)	100	25
	RF	3.44±0.09	4.2M	(28k)	1k	36
	rRF[107]	2.98±0.15	(180k)	(2 500)	100	25
	sNDF [74]	2.92±0.17	2.4M	2.4M	70	10
	TAO-c	2.88±0.09	310k	3 210	30	11
	AdaBoost	2.69±0.04	2.7M	(20k)	1k	20
	TAO-l	2.09±0.10	276k	6 310	30	7
Char74k	CART	32.14±0.15	20 157	(55)	1	55
	TAO-c	23.94±0.37	46k	432	1	12
	TAO-l	20.82±0.31	42k	2 839	1	4
	XGBoost	17.04±0.00	3.3M	(923k)	62k	50
	AdaBoost	16.93±0.18	25M	(60k)	1k	60
	ADF [117]	16.67±0.21	(4M)	(2 500)	100	25
	RF	16.61±0.14	26M	(51k)	1k	65
	sNDF [74]	16.04±0.20	59M	59M	200	12
	rRF[107]	15.40±0.10	(1.1M)	(2 500)	100	25
	TAO-c	15.19±0.18	1.5M	13k	30	12
	TAO-l	15.00±0.17	1.4M	92k	30	4
RCV1	CART	29.33±0.13	3 460	(150)	1	150
	RF	18.78± 0.37	10M	(0.2M)	1k	233
	TAO-c	17.96± 0.03	3.2M	37k	1	12
	AdaBoost	15.95± 0.39	4M	(99k)	1k	100
	TAO-l	15.73± 0.21	14k	3 357	1	4
	XGBoost	13.84± 0.00	522k	(151k)	53k	30
	TAO-c	13.77± 0.02	84M	1.1M	30	12
	TAO-l	13.29± 0.03	411k	98k	30	4

Table 2.2: Comparison on classification datasets of different forest models. We report the test error (%, avg±stdev over 5 repeats), number of parameters and FLOPS (numbers in parentheses are estimates), number of trees T and maximum depth of the forest Δ.

	Forest	E_{test}	#pars.	FLOPS	T	Δ
ailerons ($E \times 10^{-4}$)	CART	2.88±0.00	103	9	1	9
	ET	1.84±0.00	1.4M	(4 068)	100	49
	ARF	1.78±0.01	(36k)	(750)	50	15
	TAO-c	1.76±0.02	681	87	1	6
	rRF	1.75±0.02	(71k)	(1 000)	100	10
	RF	1.75±0.00	9M	(35k)	1k	47
	AdaBoost	1.75±0.00	200k	(12k)	1k	15
	TAO-l	1.74±0.01	447	93	1	5
	XGBoost	1.72±0.00	4k	(1 264)	1k	7
	TAO-c	1.67±0.04	21k	2 513	30	6
	TAO-l	1.66±0.04	27k	2 611	30	5
cpu_act	CART	3.63±0.32	9 691	25	1	25
	TAO-c	2.71±0.04	498	51	1	6
	ARF	2.62±0.01	(98k)	750	50	15
	RF	2.60±0.01	6M	(28k)	1k	37
	TAO-l	2.58±0.02	246	41	1	5
	XGBoost	2.57±0.00	294k	(8 780)	1k	10
	AdaBoost	2.56±0.11	0.7M	(10k)	1k	10
	ET	2.49±0.03	10M	(38k)	1k	50
	TAO-c	2.39±0.05	24k	1 590	30	7
	TAO-l	2.35±0.01	8k	1 179	30	5
CT slice	CART	2.71±0.06	85k	51	1	51
	TAO-c	1.54±0.05	7k	1 123	1	7
	AdaBoost	1.31±0.01	1M	(10k)	1k	10
	XGBoost	1.18±0.00	465k	(10k)	1k	10
	TAO-l	1.16±0.02	5k	768	1	5
	ET	1.06±0.01	85M	(62k)	100	82
	cRF	1.00	(17M)	–	1k	–
	RF	0.97±0.01	54M	(57k)	1k	78
	TAO-c	0.89±0.02	214k	31k	30	7
	TAO-l	0.58±0.03	242k	25k	30	6
YearPred	CART	13.41±0.11	621k	49	1	49
	ET	9.31±0.00	77M	(6 091)	100	73
	RF	9.23±0.00	401M	(52k)	1k	73
	AdaBoost	9.21±0.03	24M	(15k)	1k	15
	TAO-c	9.11±0.05	7k	448	1	8
	TAO-l	9.08±0.03	2k	388	1	6
	XGBoost	9.01±0.00	1.1M	(10k)	1k	10
	cRF	8.90	(184M)	–	1000	–
	TAO-c	8.85±0.01	246k	14k	30	9
	TAO-l	8.83±0.01	148k	12k	30	7

Table 2.3: Similar to Table 2.2 but for regression datasets.

based on better tree optimization.

2.3 Learning softmax trees with TAO

Extraordinary results from previous section on standard ML benchmarks motivated us to try TAO on large scale data. As an example, we consider extreme classification–classification problems with thousands or more classes. Such problems naturally occur in NLP and other areas. One example are language models. There are about 171k words in the current edition of the Oxford English Dictionary, and many more if we include all forms of a word, names, technical acronyms, etc. Another example is document classification. The Open Directory Project (ODP) contains over 1M website categories organized in a hierarchical ontology scheme. In this many-class setting, it is considerably difficult to learn a model that is accurate and fast at inference time. The simplest and most widespread model is a linear (e.g. softmax) classifier, possibly as the output layer of a neural net. One important problem with a softmax classifier is that one must compute the score or probability of (nearly) *all* classes, conditional on the input instance, in order to determine the (top-n) predicted class. This has a cost $\mathcal{O}(DK)$ where D is the input dimension of the softmax and K the number of classes, which is slow when K and D are large. This problem also occurs with other classifiers, such as soft decision trees. Indeed, computational constraints on the vocabulary size are a major challenge for neural machine translation [73], for example.

We argue that having the classifier output a positive probability (however small) for each class is slow and unnecessary when K is large, because, for any given instance, the majority of classes should indeed have a negligible probability. A much faster classifier is a traditional decision tree, which assigns zero probability to all classes except the predicted one obtained from root-leaf path (in $\log K$ time if the tree is balanced). However, such trees are known to be insufficiently accurate even if grown very deep.

We propose a *softmax tree (ST)*, a binary tree having sparse hyperplanes at the decision nodes (which make hard, not soft, decisions) and a small softmax classifier outputting $k < K$ classes at each leaf (a class may appear in more than one leaf). A ST is still very fast at inference: it sends the input instance to a single leaf via a path whose length is logarithmic on the number of leaves (for a complete tree), and it assigns (without computing them) probability zero to most classes (namely, all classes not in the leaf). Trading off the depth Δ of the tree and the number of classes k per leaf can potentially result in fast, highly accurate classifiers. However, STs are still hard to train because they define a nonconvex, nondifferentiable problem. We solve this by modifying *Tree Alternating Optimization (TAO)*, so that it can handle softmax leaves, and by using a good initialization. We cover related literature on this topic in section 1.3.3.

2.3.1 Softmax Trees: Definition and Training

Unlike soft decision trees, which can be readily optimized via gradient-based methods, hard decision trees pose a far more difficult optimization problem, not just nonconvex but nondifferentiable (and NP-hard). This is also true in the context of softmax trees. Traditional tree learning algorithms, such as CART [22], are based on greedily and recursively partitioning the input space, and pruning the resulting tree to reduce overfitting. However, as we have mentioned earlier, they are known to produce suboptimal trees [57] and this suboptimality hits the performance even more drastically in the extreme classification setup.

Similar to usual classification setup, we consider a K-class problem with training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \{1, \dots, K\}$ of D-dimensional instances and labels. Let $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ be a binary decision tree which produces a prediction for each input $\mathbf{x}$ by routing $\mathbf{x}$ from the root to exactly one leaf and applying a predictor function at that leaf. Each node (both decision and a leaf) has learnable parameters $\boldsymbol{\theta}_i$ and the total set of parameters of a tree is $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i \in \mathcal{N}}$, where $\mathcal{N}$ is the set of nodes. Each decision node i has a decision function $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$, sending instance $\mathbf{x}$ to the corresponding child of node i, and each leaf has a predictor function $\mathbf{g}_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \{1, \dots, K\}$ that produces the actual output. In a *softmax tree (ST)*:

- Each decision function uses a *(sparse) hyperplane* (*oblique tree*): “go to the right child if $\mathbf{w}_i^T\mathbf{x} + w_{i0} \geq 0$, else go to the left child", with parameters $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$.

- Each leaf predictor is a k-class *linear softmax*: $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i) = \sigma(\mathbf{W}_i\mathbf{x} + \mathbf{w}_i)$, where $\sigma(\cdot)$ is the softmax function and $\mathbf{W}_i \in \mathbb{R}^{k\times D}$, $\mathbf{w}_i \in \mathbb{R}^k$, where $k \leq K$ and usually $k \ll K$. This is unlike [26], which used a constant-label predictor.

We optimize the following objective function:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n;\boldsymbol{\Theta})) + \alpha \sum_{i\in\mathcal{N}} \|\boldsymbol{\theta}_i\|_1 \tag{2.3}$$

where $L(\cdot,\cdot)$ is the cross-entropy, and the ℓ_1 penalty over the weight vectors (of both decision nodes and leaves) promotes sparsity, via a hyperparameter $\alpha \geq 0$. The remaining part closely follows the original TAO algorithm described in the beginning of this chapter, i.e., separability condition applies, decision node optimization will be the same. However, optimizing over leaves now changes due to handling the softmax classifier with $k \ll K$ classes. In our STs, $\mathbf{g}_i$ for leaves is a k-class softmax classifier with an ℓ_1 sparsity penalty. We first estimate the k classes (out of K possible classes) as the k most populous classes in $\mathcal{R}_i$. Then we train the softmax, which is a convex problem. We solve it using SAG [115]. Algorithm 2 shows the modified pseudocode for training softmax tree using TAO.

Dealing with zero probabilities In our STs, each leaf operates on k classes. If $k = K$, each possible class receives a positive probability, but if $k \ll K$ then many $(K - k)$ classes receive exactly zero probability. This is necessary to achieve the fast prediction we seek, but it results in an infinite cross-entropy value whenever an instance with ground-truth class y is routed to a leaf that does not contain y. This causes no issue in the reduced problem over a leaf (the softmax uses only the top-k classes in that leaf), but it does cause an issue in the reduced problem over a decision node. Here, we have to solve a weighted 0/1 loss binary classification problem where the weights are obtained by evaluating the prediction's loss from the left and right subtrees for each instance in the node, and some of those weights can be infinity. To mitigate these issues and make sure learning succeeds, we tried the following approaches:

Algorithm 2: Softmax tree (ST) training.

```
Result: trained tree $\mathbf{T}(\cdot;\mathbf{\Theta})$
input
training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$;
  initial tree $\mathbf{T}(\cdot;\mathbf{\Theta})$ of depth $\Delta$;
repeat
    for depth $d = \Delta$ downto 0 do
        for $i \in$ nodes at depth $d$ do
            if $i$ is a leaf then
                $\overline{\mathcal{R}}_i \leftarrow$ instances of the most populous $k$ classes in $\mathcal{R}_i$;
                $\boldsymbol{\theta}_i \leftarrow$ fit a linear classifier on $\overline{\mathcal{R}}_i$;
            else
                generate pseudolabels $\overline{y}_n$ for each point $n \in \mathcal{R}_i$;
                $\boldsymbol{\theta}_i \leftarrow$ fit a weighted binary classifier to minimize eq. (2.2);
            end
        end
    end
    Update the reduced set $\mathcal{R}_i$ for each node;
until max number of iterations;
postprocessing: remove dead or pure subtrees;
```

1. Remove from the reduced problem any instance with loss=∞ (in either the left or right subtree). This performs very badly.

2. Replace loss=∞ by loss=β, where β is typically a large value (e.g. 100, 10^7). This is the option that works best in a number of datasets we have tried (see [146]), but it requires an extra hyperparameter β. This is essentially the same as using a leaf model which predicts class probabilities with a softmax for its k classes and a constant, small value $\exp(-\beta)$ for all other $K - k$ classes.

3. Use the 0/1 loss instead of the cross-entropy in the overall objective function of eq. (2.3). This avoids the infinity issue altogether, since the pseudolabels'

weight is either 0 or 1 (as in [26]). However, the reduced problem over a leaf must now optimize the 0/1 loss (which is NP-hard) rather than the cross-entropy; we approximate this by using the cross-entropy as surrogate loss, so we still learn a softmax as usual. This requires no additional hyperparameter and does quite well. It is our default option.

2.3.2 Obtaining an initial tree

While TAO monotonically decreases the objective function, it still converges to a local optimum. For the constant-label leaf oblique trees of [26], which were applied to problems with few classes, using as initial tree a complete tree of depth Δ with random parameters worked well (we call this "random initialization"). However, with many classes we have observed that the following *greedy hierarchical clustering* initialization works quite better. Assume a complete tree of depth Δ having $L = 2^{\Delta}$ leaves (although the idea carries over to any binary tree structure). The following simple algorithm is guaranteed to assign classes to leaves in a way that respects the ST structure and keeps similar classes near each other in the tree.

First, we cluster the training instances into L clusters using k-means. The L clusters will be assigned one-to-one to the L leaves by a greedy hierarchical clustering, as follows. We greedily merge pairs of clusters to achieve $\frac{L}{2}$ "superclusters". That is, we first merge the two closest clusters into one supercluster (which becomes their parent node). Then, we merge the two closest clusters of the remaining clusters, etc. Note that, unlike in regular hierarchical agglomerative clustering, the resulting supercluster is not considered for merging immediately, but rather each level is considered separately, so that we obtain a tree with a desired structure (balanced). We define the distance between two (super)clusters as the Euclidean distance between their means. We repeat the greedy merging into $\frac{L}{4}$, $\frac{L}{8}$, etc. superclusters until we reach a single supercluster containing all training instances (the root of the tree). This gives the assignment of clusters to leaves of our tree. (A faster version of this is obtained by first replacing all the training instances within each class with a "class prototype", weighted by the number of instances, and then proceeding as above to find a greedy hierarchical clustering of these K prototypes.)

Now that each instance is assigned to one leaf, the first TAO iteration can start, in reverse BFS order.

The idea is that the tree leaves induce a hierarchical partition of the input space into polytopes, hence 1) the training instances within one leaf's polytope should generally be closer to each other than to instances in other polytopes, and 2) this remains true as clusters are merged according to the tree (i.e., the polytopes of two sibling leaves will be near each other, etc.).

2.3.3 Experiments

We demonstrate the performance of our method on two popular NLP tasks: (a) large scale text classification, and (b) language modeling. Experiments suggest that *our resulting softmax trees outperform simple and advanced baselines either in accuracy (and yet very fast) or in prediction time (and yet showing competitive accuracy); or quite often in both of these indicators.* Moreover, the resulting models are compact in terms of memory requirements.

We initialize our softmax trees (ST) using a "clustering-based" method described in section 2.3.2 (unless otherwise specified). The sparsity penalty (α) set according to the cross-validation (10% of the training data). Increasing the number of TAO iterations results to a better performance but at a cost of having slower training time. Maximum number of classes (k) at each leaf is another tunable hyperparameter and we report it for each performed experiment (e.g. ST(k=50)).

As for the baselines, we use scikit-learn's [100] implementation of the one-versus-all and softmax linear classifiers. Additionally, we compare our results with more recent baselines which show state-of-the-art performance on various extreme classification problems: LOMTree [34], RecallTree [37], (π, κ)-DS [69] and MACH [86]. Where applicable, we use the available implementations of the mentioned methods. Finally, we have implemented hierarchical softmax as a tree-based baseline for language modeling tasks.

We report the top-1 and top-5 errors, maximum depth (Δ), mean inference time per test sample (in ms) and uncompressed model sizes (in GB). We average the errors over 3 independent runs for softmax trees, whereas the best performance

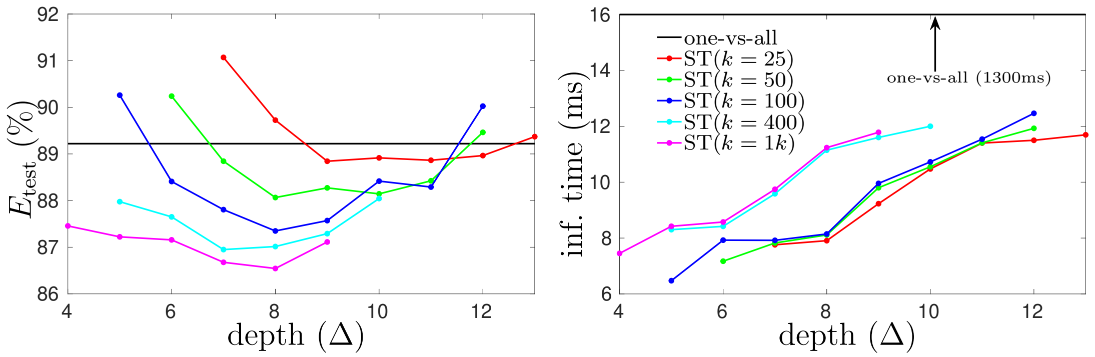


Figure 2.2: Top-1 errors and avg. inference time tradeoff of the ST for various settings of Δ and k on the ODP dataset.

is reported for other baselines. The inference time is calculated in a single CPU without parallel processing using the following methodology: we sequentially pass each test sample to the trained model and measure its prediction time. Then we average the results over all test set. Also, we report the storage requirement for each model (uncompressed and stored in sparse format if applicable).

2.3.4 Results: text classification

We perform the first set of experiments on two document categorization benchmarks with large number of classes: ODP–website categorization problem which has over 105k classes and WIKI–Small (with > 36k classes). Input feature vector for each document is normalized bag-of-words representation containing around 400k dimensions. See [146] for details and additional benchmarks.

Table 2.4 shows that the STs consistently outperform other baselines and by a considerable margin, showing outstanding performance on these benchmarks. Moreover, they achieve faster inference time compared to most of the baselines (e.g. one-vs-all, MACH) and shows a similar speed as of RecallTree and LOMTree (i.e., other tree-based methods).

Additionally, fig. 2.2 shows a tradeoff between error-vs-depth and inference_time-vs-depth. It also examines different values for k. In general, increasing k results to better models in terms of error. On the other hand, it increases the inference time (right figure), although the difference is typically negligible. Finally, the results

	Method	top-1	Δ	inf.(ms)	size(GB)
WIKI–Small	RecallTree	92.64	15	0.97	0.8
	one-vs-all	85.71	0	10.70	53.5
	MACH	84.80	–	252.64	1.3
	(π, κ)-DS	78.02	–	10.33	0.01
	ST$(k = 100)$	77.26	7	0.33	0.03
	ST$(k = 300)$	76.86	7	0.49	0.04
	ST$(k = 150)$	76.33	8	0.57	0.05
	ST$^+(k = 150)$	75.65	8	0.52	0.05
ODP	RecallTree	94.64	6	8.42	3.4
	LOMTree	(93.46)	(17)	(0.26)	–
	one-vs-all	89.22	0	1317.58	155.7
	(π, κ)-DS	86.31	–	36.41	1.0
	MACH	84.55	–	684.04	1.2
	ST$(k = 300)$	83.78	9	9.59	0.1
	ST$^+(k = 300)$	81.84	9	9.87	0.1

Table 2.4: Results on text classification datasets. We report the top-1 test error, maximum depth (Δ), avg. inference time per test sample (in ms) and uncompressed model sizes (in GB). ST$(k = x)$ indicates our method which uses at most k classes at each leaf. The results in brackets are taken from the corresponding papers. "+" shows the results of using cross-entropy loss with $\beta = 100$.

suggest that the Depth (Δ) should be sufficiently large but overfitting may occur passing a certain point (e.g. middle plot).

Model sizes Table 2.4 reports another critical aspect–compactness of our models. Just as our STs are very fast, they also generate extremely compact models compared to baselines (at least 10x gain). This is due to the ℓ_1 penalty applied at each node, which leads to sparse weights. Moreover, we observe that the best performance for STs is typically achieved with shallow trees (see Δ) which also helps to reduce the model size.

2.3.5 Results: language modeling

We conduct experiments on PTB dataset which has been extensively used to study language modeling problems. Dataset description as well as our preprocess-

Method	top-1/top-5	PPL(% covered)	Δ	inf.(ms)
HSM-approx	92.2 / 86.5	575 (100%)	18	0.184
HSM	91.1 / 81.1	575 (100%)	18	0.421
one-vs-all	87.5 / 80.2	220 (100%)	0	0.402
softmax	86.9 / 79.6	217 (100%)	0	0.467
ST(k=50)	86.5 / 72.5	17 (44%)	8	0.058
ST(k=100)	86.5 / 71.5	27 (51%)	7	0.058
ST(k=200)	86.4 / 70.6	45 (58%)	6	0.053
ST(k=400)	86.4 / 69.7	71 (67%)	5	0.064
ST(k=800)	86.4 / 68.4	117 (77%)	4	0.066
ST*(k=800)	86.4 / 68.4	427 (100%)	4	0.066

Table 2.5: Like Table 2.4 but on PTB–language modeling task. We also report the test Perplexity (with percentage of the covered points) and top-5 error. "*" indicates that smoothing was applied to replace 0 probabilities with some small epsilon and re-normalize the output.

ing steps can found in [146] and in Appendix B. As for the baselines, we use the same one-vs-all classifier described earlier and Hierarchical Softmax (HSM) model (we closely follow the setup from [88]). Also, we have implemented "HSM-approx" which chooses a child with the highest probability at each split (i.e., it achieves a faster prediction time). We use the random initialization for ST. As for the HSM, we use our own implementation in Pytorch (see details in [146]).

We report the train/test Perplexities (PPL), which is commonly done for such tasks: PPL $= \exp(-\frac{1}{N}\sum_{i=1}^{N}\log Pr(y_i|\mathbf{x}_i))$, where N is the sample size (train or test), y_i and $\mathbf{x}_i$ are ground truth label and input feature vector of the instance i, respectively. Most of the baselines described in the previous section (especially tree-based methods) do not produce class probabilities and they can not be directly applied to solve the language modeling problem, so we omit their comparison. For ST, we calculate $Pr(y_i|\mathbf{x}_i)$ by routing an instance x_i to the corresponding leaf of a tree and taking softmax on the output produced by that leaf. If y_i (correct class) is not presented in that leaf (it may happen since a leaf stores $k < K$ classes) then we do not include it to the calculation. Therefore, we provide the total number of points with non-zero probability predictions. Note that the fact that our STs output exactly zero probability for many classes is by design and results in its

Method	top-1/top-5	PPL(% covered)	Δ	inf.(ms)
HSM-appox	78.3 / 64.1	184 (100%)	18	0.097
HSM	77.7 / 63.1	184 (100%)	18	0.372
softmax	74.3 / 54.8	96 (100%)	0	0.346
ST(k=50)	75.2 / 57.3	9 (59%)	8	0.046
ST(k=100)	75.0 / 56.8	13 (64%)	7	0.045
ST(k=200)	74.9 / 56.2	18 (70%)	6	0.067
ST(k=400)	74.7 / 55.9	24 (76%)	5	0.066
ST(k=800)	74.5 / 55.5	33 (81%)	4	0.069
ST*(k=800)	74.5 / 55.5	145 (100%)	4	0.069

Table 2.6: Like Table 2.5, but models were trained on the output of the recurrent neural net (LSTM).

inference speed. Also note that a softmax classifier will happily assign a positive probability to a class whose region is actually empty (i.e., no input $\mathbf{x} \in \mathbb{R}^D$ ever results in that class winning). That said, we also provide the results of applying a smoothing technique [42, section 6.2] to ensure positive probabilities for all classes, without any increase in inference time (denoted by "*" in tables). Specifically, we assign some small ϵ to all instances with zero probability and renormalize the output probabilities. This requires additional hyperparameter ϵ which we tune using cross-validation.

Table 2.5 summarizes our results. First of all, one can notice that both versions of HSM perform worse compared to one-vs-all (both error and PPL) which coincides with previous findings [91]. As for the ST, it shows a decent test error (both top-1/top-5) and the fastest inference time than the other baselines. Regarding the perplexity score, our method produces exactly zero probability for some instances which makes overall PPL unbounded (i.e., infinity). However, if we discard such cases and focus on a subset of data for which probability estimate is non-zero (see "% covered" in the table), then it achieves a significantly low PPL. Moreover, it is clear from the Table 2.5 that ST covers majority of the points and such coverage increases as we increase k. As for the results using smoothing (denoted by "*"), the PPL score is still much lower compared to HSM but higher than one-vs-all. This logically makes sense since instances with zero probability increase PPL score

Method	WIKI–Small	ODP
one-vs-all	>7d	>7d
LOMTree	–	(36m)
RecallTree	53m	113m
MACH	1445m	2301m
ST	1033m	2880m

Table 2.7: Training times in minutes (m) or days (d) for the datasets in Table 2.4. For ST, we report the training times for the best performing architecture (in terms of test error). For LOMTree, we report the results from [37] when applicable.

substantially.

Neural language modeling

Modern neural nets are well known to achieve the state-of-the-art performance in language modeling problems. As a comparison, simple RNNs can easily reach PPL = 101 on the same problem [89] from the previous section. Therefore, we combine our softmax trees with the output of LSTM and show that it achieves a comparable performance with faster inference time. Specifically, we use our Pytorch implementation (see details in [146]) of the RNN model for the word-level language modeling on the same PTB dataset with all 10k unique words as the vocabulary. Table 2.6 summarizes our findings. The neural net model achieves 96.33 perplexity score on a test set using softmax classifier as the last layer. Once training is done, we extract the last output of the LSTM layer and use it as input to the ST (i.e., input is a vector $\in \mathbb{R}^{150}$). In other words, ST is not trained in end-to-end fashion but sequentially. Despite this, our method shows a similar performance compared to the plain softmax in terms of train/test errors and consistently faster during inference time (about 5.7 times). Regarding the perplexity score, as in the above case, we cover majority of the data points for which the PPL is significantly low compared to the baseline.

2.3.6 Training time

Table 2.7 gives representative runtimes for several datasets. We train all methods using at most 16 parallel threads. In general, all tree-based and hashing-based methods are faster to train compared to one-vs-all. For smaller datasets, training softmax trees is as expensive as RecallTree, but faster than MACH. For larger datasets, ST requires more time to find a good solution. Even in that case, it shows a comparable runtime against MACH. Overall, the runtime of ST is reasonable and more than justified by the fast inference time and low test error it achieves.

Chapter 3

Semi-Supervised Learning with Decision Trees

Tree Alternating Optimization (TAO) algorithm described in the previous chapter shows remarkable performance on supervised learning tasks, such as classification and regression. However, extending TAO trees beyond these problems meet certain challenges, which we study in this chapter. Actually, we partially discussed some of the issues in section 1.3.4 for generic tree-based models and TAO trees naturally inherit all those issues. In this chapter, we discuss the first example of the application where *manifold regularization* appears: semi supervised learning.

Semi-supervised learning (SSL) is an important subfield of machine learning which has received a lot of attention in recent years given today's growing amount of data and widespread deployment of machine learning systems. One of the major reasons is that SSL is applicable when labels are scarce. This is in contrast to the traditional fully supervised learning, which requires access to a large amount of high-quality labeled data. However, obtaining such samples is often costly, time-consuming and sometimes even impractical. Therefore, SSL methods have received much praise in the machine learning literature [153] and they are widely used in many applications. A common strategy in SSL is to assume that similar instances have similar predictions, which is commonly incorporated into an objective as a graph prior or manifold regularization (e.g. graph Laplacian).

As we have mentioned earlier, there are several challenges with training tree-

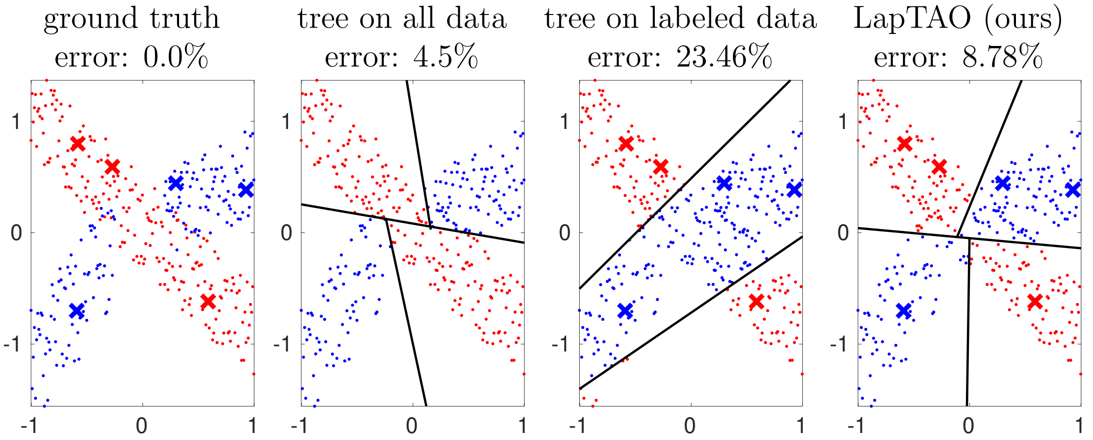


Figure 3.1: Binary classification on 2D. Plot 1 shows the original data and corresponding class labels. Cross markers (×, six in total) indicate the labeled points that we provide to any given SSL algorithm. "Tree on all data" shows the decision boundary obtained by using all available labeled data, whereas plot 3 uses only six labeled points to train a tree. Plot 4 shows the result of our SSL framework. All trees are oblique of depth $\Delta = 2$.

based models in semi-supervised learning setup. One problem is due to difficulty of the optimization problem, which involves non-differentiable and non-convex objective. Another challenge is, like many non-linear methods, decision trees are well known to overfit for small-sized (labeled) data, which is the case in SSL. As an illustration, consider fig. 3.1, that shows a synthetic binary classification problem in 2D. An oblique tree achieves a certain good performance when it is provided the entire population of labeled data. But the error significantly increases if a tree is trained on six labeled instances only (plot 3). Whereas the benefit is evident when we provide all data (six labeled and the rest are unlabeled) and properly optimize a tree within SSL framework.

In this chapter, we propose a novel SSL framework that is specifically designed for training discrete structured models (e.g. decision trees). In our proposed approach, we first state the objective, which consists of a supervised loss (for the labeled data only) and a graph Laplacian regularization (also known as manifold regularization [6]). The resulting optimization problem is long considered to be hard to solve since trees define a non-differentiable, non-convex mapping. By reformulating the problem as a constrained optimization, we derive an efficient and scalable iterative algorithm (section 3.1) which requires solving two simpler problems at each step: a sparse linear system and a supervised tree learning problem. For the latter, we use the TAO algorithm, which is crucial for the success of our approach. Moreover, for a special case where the tree structure as well as the parameters in each decision node (not leaves) are fixed, we derive the exact solution given by another linear system (section 3.1.2). Experimental results (section 3.2) show the algorithm is able to learn accurate and interpretable decision trees even with very few labeled instances.

3.1 LapTAO: semi-supervised learning framework for decision trees

We are given the dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$, where $\mathcal{D}_l = \{\mathbf{x}_n, y_n\}_{n=1}^{l} \subset \mathbb{R}^D \times \mathbb{R}$ is the labeled portion of the data, with l points, and $\mathcal{D}_u = \{\mathbf{x}_n\}_{n=l+1}^{N} \subset \mathbb{R}^D$ is the unlabeled portion, with $N - l$ points. Then our goal is to minimize the following regularized objective:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{l} (T(\mathbf{x}_n; \boldsymbol{\Theta}) - y_n)^2 + \alpha\, \phi(\boldsymbol{\Theta}) + \gamma \sum_{n,m=1}^{N} w_{nm}(T(\mathbf{x}_n; \boldsymbol{\Theta}) - T(\mathbf{x}_m; \boldsymbol{\Theta}))^2. \tag{3.1}$$

For simplicity, we consider a single real-valued output (e.g. as in regression) and thus the ground truth label y_n is scalar. However, we extend this assumption in the next section. Here, w_{nm} are the weights in the affinity (similarity) matrix based on a graph on all the data points $\mathcal{D}$, usually a nearest-neighbor graph; T: $\mathbb{R}^D \rightarrow \mathbb{R}$ is the tree predictive mapping, with parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{\text{nodes}}$; $\phi(\cdot)$ is a regularization penalty, such as $\|\cdot\|_1$; and γ,α are regularization hyperparameters. Rather than using a greedy recursive partitioning procedure (such as CART [22] or C5.0 [106]), which does not optimize any loss function, we consider T as a parametric model with trainable weights in each node (like fixing a neural net architecture and optimizing over its parameters). If T was differentiable, one could optimize (3.1) via gradient-based methods, as can be done for neural nets [134]. Similarly, the solution is relatively straightforward to obtain if problem (3.1) is convex [6]. However, solving problem (3.1) is non-trivial with a tree which defines a non-differentiable and non-convex mapping. Instead, we apply the *method of auxiliary coordinates* [28, 29], a generic method for optimizing nested systems.

We proceed as follows by reformulating problem (3.1) in an equivalent form. We introduce a new auxiliary variable $z_n \in \mathbb{R}$ for each training instance n and consider the constrained problem:

$$\min_{z_1,\dots,z_N,\boldsymbol{\Theta}} \sum_{n=1}^{l} (z_n - y_n)^2 + \alpha\ \phi(\boldsymbol{\Theta}) + \gamma \sum_{n,m=1}^{N} w_{nm}(z_n - z_m)^2 \tag{3.2}$$

$$\text{s.t.} \quad z_n = T(\mathbf{x}_n; \boldsymbol{\Theta}) \quad n = 1, \dots, N. \tag{3.3}$$

Obviously, by putting the constraints (3.3) back into eq. (3.2), we end up with the same objective as in (3.1), so these two problems are equivalent. Let us denote $\mathbf{y} = [y_1, y_2, \dots, y_l, 0, 0, \dots]^T \in \mathbb{R}^N$ the augmented ground truth vector, i.e., we put zeros in the unlabeled portion of the data. Similarly, introduce a diagonal matrix $\mathbf{J} = \operatorname{diag}(1, \dots, 1, 0, \dots, 0) \in \mathbb{R}^{N\times N}$ with the first l diagonal entries equal to 1 and the rest to 0. Also, let the graph Laplacian be $\mathbf{L} = \mathbf{D} - \mathbf{W}$ with a diagonal matrix $\mathbf{D} \in \mathbb{R}^{N\times N}$ (the degree matrix) having entries $d_{nn} = \sum_{m=1}^{N} w_{nm}$, and let $\mathbf{W} = (w_{nm}) \in \mathbb{R}^{N\times N}$ be the affinity matrix. Finally, call $\mathbf{z} = [z_1, \dots, z_N]^T$ and $\mathbf{t}(\mathbf{X}; \boldsymbol{\Theta}) = [T(\mathbf{x}_1; \boldsymbol{\Theta}), \dots, T(\mathbf{x}_N; \boldsymbol{\Theta})]^T$, where $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$. Then we can rewrite eq. (3.2)-(3.3) as follows:

$$\min_{\mathbf{z},\boldsymbol{\Theta}} (\mathbf{z} - \mathbf{y})^T \mathbf{J}\ (\mathbf{z} - \mathbf{y}) + \alpha\ \phi(\boldsymbol{\Theta}) + \gamma\ \mathbf{z}^T \mathbf{L}\ \mathbf{z} \quad \text{s.t.} \quad \mathbf{z} = \mathbf{t}(\mathbf{X}; \boldsymbol{\Theta}). \tag{3.4}$$

Now we solve this using the augmented Lagrangian method [96]. This defines a new, unconstrained optimization problem:

$$\min_{\mathbf{z},\boldsymbol{\Theta}} (\mathbf{z}-\mathbf{y})^T \mathbf{J}\ (\mathbf{z}-\mathbf{y}) + \alpha\ \phi(\boldsymbol{\Theta}) + \gamma\ \mathbf{z}^T \mathbf{L}\ \mathbf{z} - \boldsymbol{\lambda}^T (\mathbf{z} - \mathbf{t}(\mathbf{X}; \boldsymbol{\Theta})) + \mu \|\mathbf{z} - \mathbf{t}(\mathbf{X}; \boldsymbol{\Theta})\|^2 \tag{3.5}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^N$ are the estimates of the Lagrange multipliers. Optimizing this for each $\mu > 0$ produces a sequence of $(\mathbf{z}_\mu, \mathbf{t}_\mu(\mathbf{X}; \boldsymbol{\Theta}))$ and, as $\mu \to \infty$, we gradually force the minimizer to be in the feasible set of the constrained problem. Finally, in order to minimize (3.5) over $\mathbf{z}$ and $\mathbf{t}(\mathbf{X}; \boldsymbol{\Theta})$ for fixed μ, we apply alternating optimization over $\mathbf{z}$ and $\boldsymbol{\Theta}$:

- **Label-step** (optimizing over $\mathbf{z}$ given fixed $\mathbf{t}(\mathbf{X};\boldsymbol{\Theta})$). The objective in eq. (3.5) is a quadratic function and a minimizer is obtained by solving the linear system:

$$\begin{aligned}&\min_{\mathbf{z}}\ (\mathbf{z}-\mathbf{y})^T\mathbf{J}\ (\mathbf{z}-\mathbf{y})+\gamma\ \mathbf{z}^T\mathbf{L}\ \mathbf{z}-\boldsymbol{\lambda}^T(\mathbf{z}-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta}))+\mu\|\mathbf{z}-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta})\|^2\Rightarrow\\ &\frac{\partial\mathcal{L}}{\partial\mathbf{z}}=\mathbf{J}(\mathbf{z}-\mathbf{y})+\gamma\mathbf{L}\mathbf{z}-\frac{1}{2}\boldsymbol{\lambda}+\mu(\mathbf{z}-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta}))\quad\text{and}\\ &\frac{\partial^2\mathcal{L}}{\partial\mathbf{z}\partial\mathbf{z}^T}=\mathbf{A}=\mathbf{J}+\mu\mathbf{I}+\gamma\mathbf{L}\Rightarrow\\ &\mathbf{A}\mathbf{z}=\mathbf{J}\mathbf{y}+\mu\mathbf{t}(\mathbf{X};\boldsymbol{\Theta})+\frac{1}{2}\boldsymbol{\lambda}\end{aligned}\tag{3.6}$$

  where $\mathbf{A}$ is a positive definite matrix, because: $\mu,\gamma>0$, $\mathbf{L}$ is positive semidefinite [6], $\mathbf{I}$ is identity and $\mathbf{J}$ is the diagonal matrix with first l entries equal 1 and the rest are 0. Therefore, $\mathbf{x}^T\mathbf{A}\mathbf{x}=\mathbf{x}^T\mathbf{J}\mathbf{x}+\mu\mathbf{x}^T\mathbf{I}\ \mathbf{x}+\gamma\mathbf{x}^T\mathbf{L}\mathbf{x}>0$ for all $\mathbf{x}\in\mathbb{R}^N$. This means that our problem is strictly convex with a unique solution given by the linear system shown above. Moreover, $\mathbf{A}$ is a sparse matrix if the graph Laplacian $\mathbf{L}$ is sparse, which is the case in practice if we construct $\mathbf{W}$ by using a nearest neighbors graph. This allows us to solve a large scale linear system in an efficient way (e.g. by caching a matrix factorization as described in section 3.1.1 or using the conjugate gradient method). Intuitively, the label-step can be interpreted as approximating the labels (for $\mathcal{D}_u$) using the graph Laplacian and predictions obtained from the current tree (i.e., label smoothing).

- **Tree-step** (optimizing over $\boldsymbol{\Theta}$ given fixed $\mathbf{z}$). Problem (3.5) reduces to a regression fit of a tree:

$$\begin{aligned}&\min_{\boldsymbol{\Theta}}\ \mu\|\mathbf{z}-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta})\|^2+\alpha\ \phi(\boldsymbol{\Theta})-\boldsymbol{\lambda}^T(\mathbf{z}-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta}))\Leftrightarrow\\ &\min_{\boldsymbol{\Theta}}\left\|\left(\mathbf{z}-\frac{1}{2\mu}\boldsymbol{\lambda}\right)-\mathbf{t}(\mathbf{X};\boldsymbol{\Theta})\right\|^2+\frac{\alpha}{\mu}\phi(\boldsymbol{\Theta}).\end{aligned}\tag{3.7}$$

  Note that here we use $(\mathbf{z}-\frac{1}{2\mu}\boldsymbol{\lambda})$ as “ground-truth” labels (not y_n, which is not defined for $\mathcal{D}_u$ anyway). We solve this problem using the TAO algorithm, which we discussed in the previous chapter. Intuitively, this step can be understood as fitting a tree with the current estimates of the labels.

  As we later discuss in experiments section, we pick a *sparse oblique tree with constant leaves* as our main model. This is purely for demonstration purposes since this method is quite generic. The motivation behind this choice is twofold. First, traditional axis-aligned trees are very restrictive since each split uses a single feature, which neglects interactions or correlations between features. Indeed, empirical results show oblique trees achieve far better performance [148]. Besides, thanks to the sparsity, only few features are active at each split, and nodes in the initial tree can become redundant and be pruned if their weight $\mathbf{w}_i$ becomes zero [26]. This, together with the fact that oblique trees typically have small depth, makes the final model more interpretable.

Finally, the step over Lagrange multipliers is done by the update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \mu(\mathbf{z} - \mathbf{t}(\mathbf{X}; \boldsymbol{\Theta}))$. In summary, our algorithm alternates between solving a linear system and training a tree. After each (label,tree)-step, we increase the penalty parameter μ, we update $\boldsymbol{\lambda}$ and we keep iterating until approximate convergence or other stopping criterion (e.g. maximum number of iterations reached). We call our algorithm *LapTAO* and provide the pseudocode below in Algorithm 3.2.

3.1.1 Practicalities of LapTAO

Algorithm 3.2 provides a general picture on how our proposed semi-supervised learning framework works. However, there are several implementation details that are worth mentioning, which we discuss below.

Initialization for LapTAO To start our iterative algorithm, we need to obtain initial solutions ($\mathbf{z}_0$, $\mathbf{t}_0$) for eq. (3.5) when $\mu \rightarrow 0^+$. This is straightforward to achieve for $\mathbf{z}_0$, as it involves solving the same linear system as in eq. (3.6) but with $\mu = 0$ and $\boldsymbol{\lambda} = \mathbf{0}$: $(\mathbf{J} + \gamma\mathbf{L})\mathbf{z} = \mathbf{J}\mathbf{y}$. This can be considered as a non-parametric smoothing of the labels obtained by propagating (diffusing) the ground-truth labels over all points (labeled and unlabeled) through the graph Laplacian. Although these smoothed labels are not optimal in problem (3.1), which requires optimizing them jointly with the tree, they do provide a good initialization, and it is convenient to solve the linear system exactly for $\mu = 0$. After that, we fit a tree using $\mathbf{z}_0$ as ground-truth labels (tree-step). Recall that TAO requires an initial

```
input labeled set $\mathcal{D}_l = \{\mathbf{x}_n, y_n\}_{n=1}^{l}$ and unlabeled set $\mathcal{D}_u = \{\mathbf{x}_n\}_{n=l+1}^{N}$;
    penalty parameters: $\alpha$, $\gamma$; $\mu$ schedule: $\mu_0, \dots, \mu_{\max}$;
    graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$;
$\boldsymbol{\lambda} \leftarrow \mathbf{0}$ (initialize Lagrange multipliers);
$\mathbf{z}_0 \leftarrow$ solve the linear system in eq. (3.6) with $\mu = 0$;
$\mathbf{t}(\cdot; \boldsymbol{\Theta}) \leftarrow$ fit a tree to $(\{\mathbf{x}_n\}_{n=1}^{N}, \mathbf{z}_0)$ (algorithm 1);
for $\mu = \mu_0 < \mu_1 < \mu_2 < \dots < \mu_{\max}$;
  repeat
    label-step: $\mathbf{z} \leftarrow$ solve the linear system in eq. (3.6) given fixed $\mathbf{t}(\cdot; \boldsymbol{\Theta})$;
    tree-step: $\mathbf{t}(\cdot; \boldsymbol{\Theta}) \leftarrow$ fit a tree to $(\{\mathbf{x}_n\}_{n=1}^{N}, \mathbf{z} - \frac{1}{2\mu}\boldsymbol{\lambda})$ given fixed $\mathbf{z}$
      using algorithm 1 (see eq. (3.7));
    Lagrange multipliers step: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \mu(\mathbf{z} - \mathbf{t}(\cdot; \boldsymbol{\Theta}))$;
  until stop
end for
Post-processing (see section 3.1.2)
return $\mathbf{t}(\cdot; \boldsymbol{\Theta})$
```

Figure 3.2: Pseudocode for LapTAO. “Stop” for inner loop occurs when ($\mathbf{z}$,$\mathbf{t}(\cdot; \boldsymbol{\Theta})$) converge (ideally). However, in practice, we use a fixed number of iterations (e.g. 1 in most experiments).

tree, which, for instance, can be obtained by generating a complete tree of depth Δ (a hyperparameter) with a Gaussian random weight vector at each decision node.

Hyperparameters of LapTAO The hyperparameters are γ for the graph prior, μ schedule for the augmented Lagrangian, α (sparsity) and Δ (depth) for the tree. They can be selected by cross-validation.

Extension to multioutput regression and classification We can extend LapTAO for multiple outputs in a straightforward way: the label-step solves the linear system (3.6) for each dimension separately and the tree-step applies TAO as usual, as it can handle a vector-valued output. For classification, we use a one-hot encoding of the labels and consider the problem as a regression task, as is commonly done in the decision tree literature [57] (especially for boosted trees). It is possible to extend our framework for other losses (e.g. hinge, logistic) that may work better for classification. This requires certain changes in the label-step and we will explore it in our future works.

Extension to models other than trees Although our focus in this paper are decision trees, the semi-supervised learning optimization algorithm we propose is perfectly general. The model ($T(\mathbf{x};\boldsymbol{\Theta})$) appears in the algorithm in the tree-step, with the form of a regression problem having the smoothed labels as ground-truth. Obviously, we can use other regression models, such as random forests, gradient boosted trees, neural networks, etc. The motivation to use our approach was the fact that trees are not differentiable, so one cannot optimize problem (3.1) by gradient-based methods. But our approach has another, computational advantage: by separating terms through the auxiliary variables $\mathbf{z}$, the quadratic-cost term is confined to the label-step. For large datasets, this is a sparse linear system, for which efficient algorithms exist. Hence, the complexity associated with the model (tree) is linear on the dataset size (in the tree-step). This is much faster than having to deal directly with the quadratic term *and* the model, as in eq. (3.1), whether for trees or other models.

Practical reasons of using TAO in tree-step Potentially, one could apply any tree fitting algorithm to solve the tree-step in LapTAO, such as CART [22], C5.0 [106], OC1 [95], etc. But there are several important considerations. Firstly, from an optimization point of view, it is known that alternating optimization is most effective when the step over each block is (ideally) exact. This is computationally achievable for the label-step, which involves a linear system. However, training a tree optimally even in the simplest case (axes-aligned with binary inputs and output) is NP-hard [62]. Therefore, we need an approximate but good solution. Most traditional tree learning algorithms, based on greedy recursive partitioning (such as CART), are highly suboptimal [57] and do not even consider any specific loss function over trees. In contrast, the TAO algorithm fits decision trees by monotonically decreasing a well-defined and very general loss function and regularization over a well-defined parametric space of trees, given an initial tree structure and parameter values. This makes it also possible to use warm-start in the tree-step, i.e., to continue improving the tree from the previous iteration—which greedy recursive partitioning cannot do, as it constructs a new tree from scratch every time. Warm-start is essential to speed up the optimization and to achieve stability in the results (CART-type algorithms are notoriously unstable in that small changes in the training set can result in drastically different tree structures and parameters [57]). A further, important advantage of TAO is that it can learn trees of quite general types, such as oblique trees, which are far more powerful that the traditional axes-aligned trees.

Accelerating the label-step in LapTAO: caching the SVD Although Conjugate Gradients (CG) method is a reasonable choice to solve the linear system for large scale problems, there is a way to accelerate the label-step for small-medium sized problems. The crucial observation is that the coefficient matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is changed by adding $\mu\mathbf{I}$ at each iteration of the Algorithm 3.2 and the remaining part is static ($\mathbf{J} + \gamma\mathbf{L}$). This naturally leads to a question: can we improve the computation of $\mathbf{A}^{-1}$ from $O(N^3)$ to $O(N^2)$ to solve the linear system (3.6)? Denote the static part of the matrix as $\mathbf{B} = \mathbf{J} + \gamma\mathbf{L}$. Moreover, $\mathbf{B}$ is a symmetric matrix since $\mathbf{L}$ is symmetric and $\mathbf{J}$ is diagonal. Therefore, we can calculate its eigendecomposition $\mathbf{B} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$, where $\mathbf{Q}$ is an orthogonal matrix. One can derive the inverse via Sherman-Morrison-Woodbury formula. However, a more direct and easier derivation is:

$$\mathbf{A}^{-1} = (\mu\mathbf{I}+\mathbf{B})^{-1} = (\mu\mathbf{I}+\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T)^{-1} = (\mathbf{Q}(\mu\mathbf{I}+\boldsymbol{\Lambda})\mathbf{Q}^T)^{-1} = \mathbf{Q}(\mu\mathbf{I}+\boldsymbol{\Lambda})^{-1}\mathbf{Q}^T \quad (3.8)$$

where $\mu\mathbf{I} + \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T = \mathbf{Q}(\mu\mathbf{I} + \boldsymbol{\Lambda})\mathbf{Q}^T$ comes from the orthogonality of $\mathbf{Q}$: $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. Notice that $\mu\mathbf{I} + \boldsymbol{\Lambda}$ is a diagonal matrix and computing its inverse takes $O(N)$. Therefore, calculating eq. (3.8) costs $O(N^2)$. The only costly part is computing the eigendecomposition (for $\mathbf{B}$) which still requires $O(N^3)$ time (and destroys the sparsity) but we do it only once before starting our algorithm. In practice, we found this method to be useful only when N is a few thousands at most.

3.1.2 Special case: exact solution when the tree structure is fixed

Problem (3.1) is hard to solve over the entire space of decision trees, but we can obtain an exact solution if the structure and decision node parameters are fixed, since then the problem reduces to solving a linear system. In this case, the only parameters to optimize are in the leaves. Assuming each leaf outputs a constant value, we can reformulate the tree prediction as a sum of basis functions $T(\mathbf{x}) = \sum_{i=1}^{m} c_i \; b_i(\mathbf{x})$, where m is the number of leaves, $c_i \in \mathbb{R}$ is leaf i's label and $b_i(\cdot) \in \{0,1\}$ is 1 only if $\mathbf{x}$ ends up in leaf i. Now we can rewrite (3.1) as the following minimization problem over the parameters of all the leaves $\mathbf{c} = (c_1, \dots, c_m)^T$:

$$E(\mathbf{c}) = \sum_{n=1}^{l} \left(\sum_{i=1}^{m} c_i \; b_i(\mathbf{x}_n) - y_n \right)^2 + \gamma \sum_{n,m=1}^{N} w_{nm} \left(\sum_{i=1}^{m} c_i \; (b_i(\mathbf{x}_n) - b_i(\mathbf{x}_m)) \right)^2 \tag{3.9}$$

$$= (\mathbf{Bc} - \mathbf{y})^T \mathbf{J} \; (\mathbf{Bc} - \mathbf{y}) + \gamma \; \mathbf{c}^T \mathbf{B}^T \mathbf{L} \mathbf{B} \mathbf{c} \tag{3.10}$$

where $\mathbf{B} = (b_i(\mathbf{x}_n)) \in \mathcal{R}^{N \times m}$ can be precomputed since we fix the tree structure and parameters in all decision nodes. Minimizing this over $\mathbf{c}$ yields the following linear system:

$$\mathbf{Ac} = \mathbf{B}^T \mathbf{J} \mathbf{y} \tag{3.11}$$

where $\mathbf{A} = \mathbf{B}^T \mathbf{J} \mathbf{B} + \gamma \; \mathbf{B}^T \mathbf{L} \mathbf{B}$ is a matrix of $m \times m$. This is very fast to solve since oblique trees are quite shallow, so the number of leaves m is not large (at most 1 000 in our experiments), and this computation does not depend on dataset size. Once LapTAO is finished, we apply the above procedure as a post-processing to the final tree.

3.1.3 Computational complexity of LapTAO

At the top level, LapTAO runs a fixed number of iterations (depending on the μ schedule, typically less than 20). Each iteration has to solve (approximately) two subproblems:

- **Label-step**: this is a large, sparse linear system of $N \times N$ (where N is the sample size). We solve it approximately with conjugate gradients (CG), initialized by the previous iterate (warm-start). Each CG iteration is $O(Nk)$ where k is the average number of neighbors in the graph, and we run just a few CG iterations. The total runtime of the label step is less than 30 seconds in the largest experiment we conducted (1M points). Convergence can be further improved via preconditioning (e.g. Jacobi). We can also solve the linear system exactly in $O(N^2)$ by caching its SVD, but this is only convenient if N is a few thousands at most.

- **Tree-step**: fitting an oblique tree with TAO to the N training points. Although the TAO algorithm is applicable to a large spectrum of decision tree types, here we pick a *sparse oblique tree with constant leaves* as our main model. Each iteration of TAO updates each decision node and leaf node. For each leaf, we compute the average of the labels of its reduced set (training points reaching it), so this is $O(N)$ over all the leaves. For each decision node, we train a logistic regression on its reduced set. Assuming logistic regression is linear on the sample size and dimensionality, this is $O(ND)$ total for all the decision nodes at the same depth, although with a larger constant factor in the big-O notation than for the leaves. Hence, processing all the decision nodes in the tree is $O(\Delta ND)$, or equivalently, running Δ logistic regressions on the whole training set. See more details in [26, 25]. A critical computational advantage of TAO is due to the fact that each node (decision or leaf node) only handles the points in its reduced set. Therefore, TAO itself can be parallelized depthwise. In summary, the overall runtime of TAO is $O(\Delta ND)$ per TAO iteration. We run 10 TAO iterations in our experiments.

Since the tree-step dominates the label-step, in terms of runtime our algorithm is almost like sequentially training decision trees (as in boosting). Additionally, each tree-step can be parallelized. Further acceleration can be done using GPUs. This is possible with GPU-friendly implementations of logistic regression, and also because oblique trees involve scalar products (unlike axis-aligned trees).

Finally, this computational cost should also include computing the nearest-neighbor graph and its affinity matrix $\mathbf{W}$. This is indeed a large cost, and it affects all semi-supervised learning methods based on the graph Laplacian. A naive implementation requires $O(DN^2)$ to calculate the distance vector for each point and determine the nearest neighbors. For large datasets, one usually uses approximate nearest neighbors (e.g. via Locality Sensitive Hashing or other techniques [3]).

3.2 Experiments

This section shows our experimental findings. We demonstrate that *the proposed method dominates over other semi-supervised learning frameworks in accuracy and approaches fully supervised baseline with far less amount of labeled data.* This is true with practically no exceptions against baseline tree-based models where accuracy margin is often quite large. As for the other methods (non tree-based), we either outperform them or achieve similar error, which makes LapTAO a strong competitor. To show that, we consider several regression and classification benchmarks of varying sizes and across different domains. We were able to run our algorithm on a dataset with up to 1 million instances on a regular PC, which shows its scalability. Moreover, using fashion-mnist as an example, we demonstrate that the final model, a shallow oblique tree with sparse parameter vector in each node, provides insights into how it achieves a prediction allowing model interpretability.

3.2.1 Experimental setup

We compare our proposed approach (LapTAO) with the following baselines: 1) *oblique–all* fits an oblique tree with full supervision, this shows the theoretical maximum performance we can achieve; 2) *oblique–lbl* is the oblique trees trained on labeled portion of data $\mathcal{D}_l$ (this completely discards large portion of unlabeled data); 3) Self-training (*axis–self*, *oblique–self*) is an iterative procedure that uses the model predictions to enlarge the portion of labeled data, we closely follow implementation from [138]; here, “axis” means traditional axis-aligned trees; 4) Laplacian SVM is a seminal work by Belkin et al. [6] which has similar problem formulation as in eq. (3.1) but for SVM. Regarding hyperparameters, given the fixed cross-validation set (1% of train data), we explored as best as we could all important hyperparameters for all methods (see details in [145]). These include: controlling a tree depth (Δ), confidence threshold for self-training, σ and C values for LapSVM, etc. It worth to mention that the hyperparameter settings suggested by authors or their default values work best in most cases.

We use TAO to train oblique trees and CART [22] to train axis-aligned trees. For all methods that use TAO, we set the total number of TAO iterations to 15. The depth Δ as well as the regularization parameter α are tuned via cross-validation. As for the settings that are specific to LapTAO, we proceed as follows. To construct the graph Laplacian, we use the Gaussian affinities with k-nearest neighbors and perplexity parameter $\mathcal{K}$ tuned for each dataset. The linear system in the label-step is solved either using direct methods (less than 20k dimensions, see caching SVD in section 3.1.1) or Conjugate Gradient method for large scale problems. We use $\gamma = 0.1$ in all experiments. As for the main loop of the augmented Lagrangian, we iterate 20 times starting from small value for $\mu_0 = 0.001$ multiplied by 1.5 after each iteration. The remaining details, additional results and datasets descriptions can be found in [145] and in Appendix B.

3.2.2 Main results

Fig. 3.3 summarizes the main results which are the trade-off plots of test error versus the percentage of labeled data on two regression and classification tasks. Intuitively, the error should go down monotonically as we increase the amount of supervision which is clearly the case in all figures. According to our findings, KNN and "axis–self" show the worst results in almost all benchmarks. The only case when KNN performed reasonably good was on MNIST, which is known to work well with "template classifiers" (e.g. RBF network, kernel SVM, KNN, etc.). Even in that case it has a large error gap with respect to LapTAO. The poor performance of the "axis–self" can be explained by suboptimality of greedily grown trees [57] and suboptimality of the self-training approach, which is mostly based on heuristics. Next, oblique trees trained on supervised data only ("oblique–lbl") leads to the

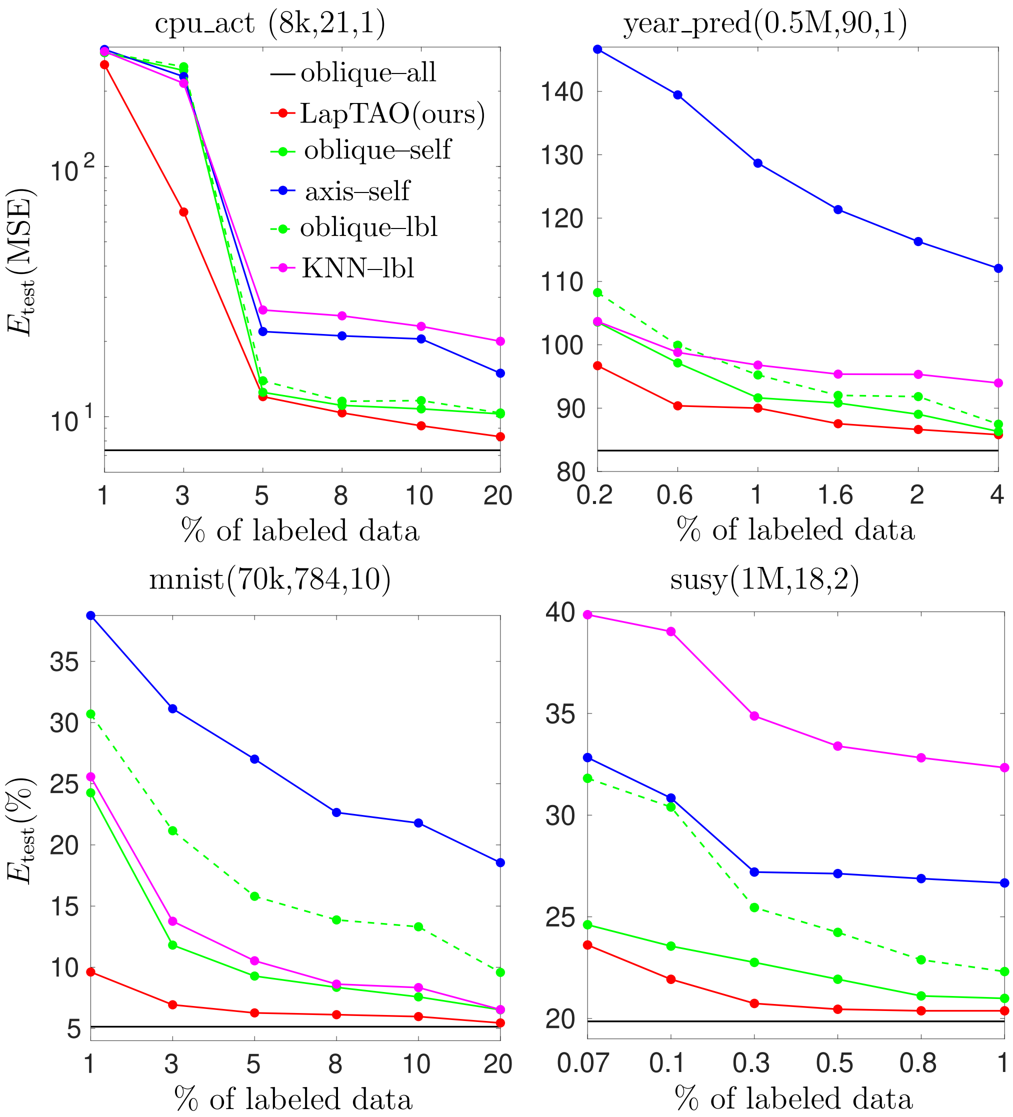


Figure 3.3: Results on regression (*top:* year_pred, cpu_act) and classification (*bottom:* mnist, susy) tasks. Numbers in brackets report the training size, number of features and output dimension (or number of classes). x–axis shows the percentage of labeled data provided to the algorithm and y–axis shows the test error. Baselines: *oblique–all* fully supervised baseline (i.e., trains an oblique tree on 100% of labeled data); **–lbl* uses $\mathcal{D}_l$ only to train the corresponding model; **–self* is an iterative self-training approach.

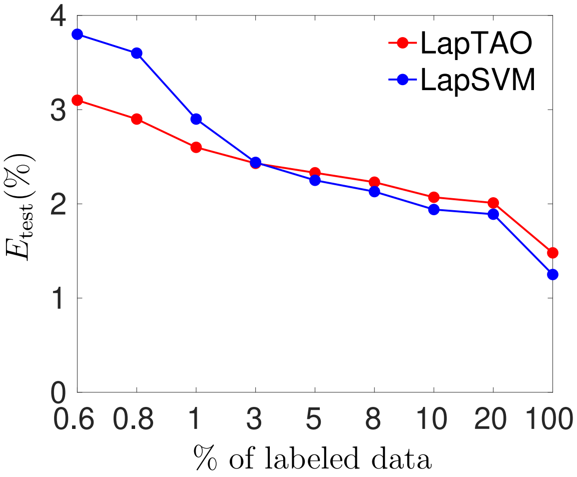


Figure 3.4: Comparison against LapSVM on Fashion-MNIST (3 classes: "shirt", "bag" and "ankle boot").

significant drop in accuracy (magenta vs black lines). This shows that relying only on labeled data is not enough to achieve a decent performance. Incorporating an oblique tree into self-training framework brings certain benefits ("oblique–self"), notably for classification tasks (green dashed vs solid lines). Finally, *LapTAO consistently improves over all other SSL methods*, often by a considerable margin. For instance, in case of 3% in cpu_act and 1% in MNIST, the difference in the error with the second best SSL approach is several orders of magnitude. It shows acceptable results even in extreme label scarcity scenarios, e.g. when we provide $< 0.5\%$ of labeled data on year_pred and susy. Moreover, LapTAO approaches the fully supervised baseline more quickly: for MNIST, we can achieve the same $\sim 5\%$ test error as "oblique–all" using only 20% of labeled training points.

Comparison with Laplacian SVM LapSVM is a natural baseline to compare with since it uses the same problem formulation as in eq. (3.1) but for the support vector machines. However, we were not able to include it to the previous comparison (in fig. 3.3) due to implementation issues: 1) it is impractical to apply it for problems beyond 30k instances as it requires computing the inverse of the modified dense Gram matrix on the entire dataset;[1] 2) it can handle a classification task only. Therefore, we pick the subset of Fashion-MNIST (3 classes: “shirt”, “bag” and “ankle boot”) resulting in 18k training points. To make the comparison as fair as possible, both of the algorithms use the same graph Laplacian matrix and we enforce the same penalty on it ($\gamma = 0.1$). For LapSVM, we use the rbf kernel with $\sigma = 5$ and the hyperparameter for LIBSVM [31] is set to $C = 100$.

The results are illustrated in fig. 3.4. It is worth to mention that similar to the original MNIST, “template classifiers”, such as kernel SVM, show quite good performance on this task which makes LapSVM a strong baseline [136]. On top of that, the problem formulation in eq. (3.1) is still convex for SVM and can be efficiently solved, whereas we are dealing with much harder problem for oblique trees. Therefore, it is nice to see that LapTAO performs similarly (but slightly worse) up to some point. However, it is surprising to see that the error gap between our approach and LapSVM narrows as we introduce more label scarcity and eventually we start to outperform when % of labeled data = 3%. From that point on, the table turns to the side of LapTAO and the difference becomes more and more noticeable (especially for 0.6%). One possible explanation for this behavior is the overfitting issue on small datasets, which is a known problem for kernel SVM, whereas sparse oblique trees are shown to be relatively robust to that [148].

Training time Table 3.1 reports the training time for different baseline methods. Overall, LapTAO algorithm for the largest experiment we performed (on susy) took less than 7 hours and around 3 hours for the moderate dataset size (mnist). This is comparable to the self-training baseline (i.e., oblique–self) but LapTAO produces far better trees in terms of accuracy. Please note that we ran our code on a regular PC (Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 32GB RAM), with little parallel processing and using unoptimized Python implementation. Therefore, the training runtime for LapTAO can be significantly improved. We did not use any GPUs.

[1] One could approximate the inverse by Nyström (or other) method but this is beyond the scope of this paper.

Dataset \ Method	LapTAO	oblique–self	axis–self	SSCT
cput_act	1072s	934s	23s	936s
mnist	11027s	9572s	514s	15932s
susy	24578s	17873s	816s	>1d

Table 3.1: Training runtime for different semi-supervised learning algorithms (in seconds).

3.2.3 Model interpretability

Model explainability and interpretability is a topic of renewed interest due to the widespread usage of machine learning and the risks associated with privacy, algorithmic bias, etc. In order to trust and rely on such automated systems, it is crucial to understand how they achieve a certain prediction. In contrast to "black box" models, decision trees are long considered as interpretable models due to the hierarchical structure. This allows to transform the model prediction as "if-then" rules extracted from root-to-leaf path. Specifically for oblique trees, each logical clause takes the following form: go to the left child if $\mathbf{w}^T\mathbf{x} < w_0$, else go to the right. This makes the interpretation a little harder since we need to look at linear combination of features at each split. However, in our case, we add ℓ_1 penalty which encourages parameter vector at each node to be sparse, i.e., only few features participate in decision making.

In this section, we argue that the oblique trees trained using LapTAO strike a good trade-off between accuracy and interpretability which is controlled via hyperparameter α. To illustrate this, we use the same subset of Fashion-MNIST as in the previous section and train a sparse oblique tree using LapTAO (10% of training data are labeled). By decreasing the value of α we enforce more sparsity resulting into shallower and more interpretable trees (since complexity decreases). However, we sacrifice the performance since the error goes up.

Fig. 3.5 shows the results for $\alpha = 1$ and $\alpha = 10$. For simplicity, let us focus on the bottom tree ($\alpha = 10$). Clearly, each leaf contains instances of nearly the same class since the average image looks like a representative "template" from the corresponding class. As for the decision node, consider the root (node #1). All "boot" images are sent to the right child of the root. Also, it is easy to

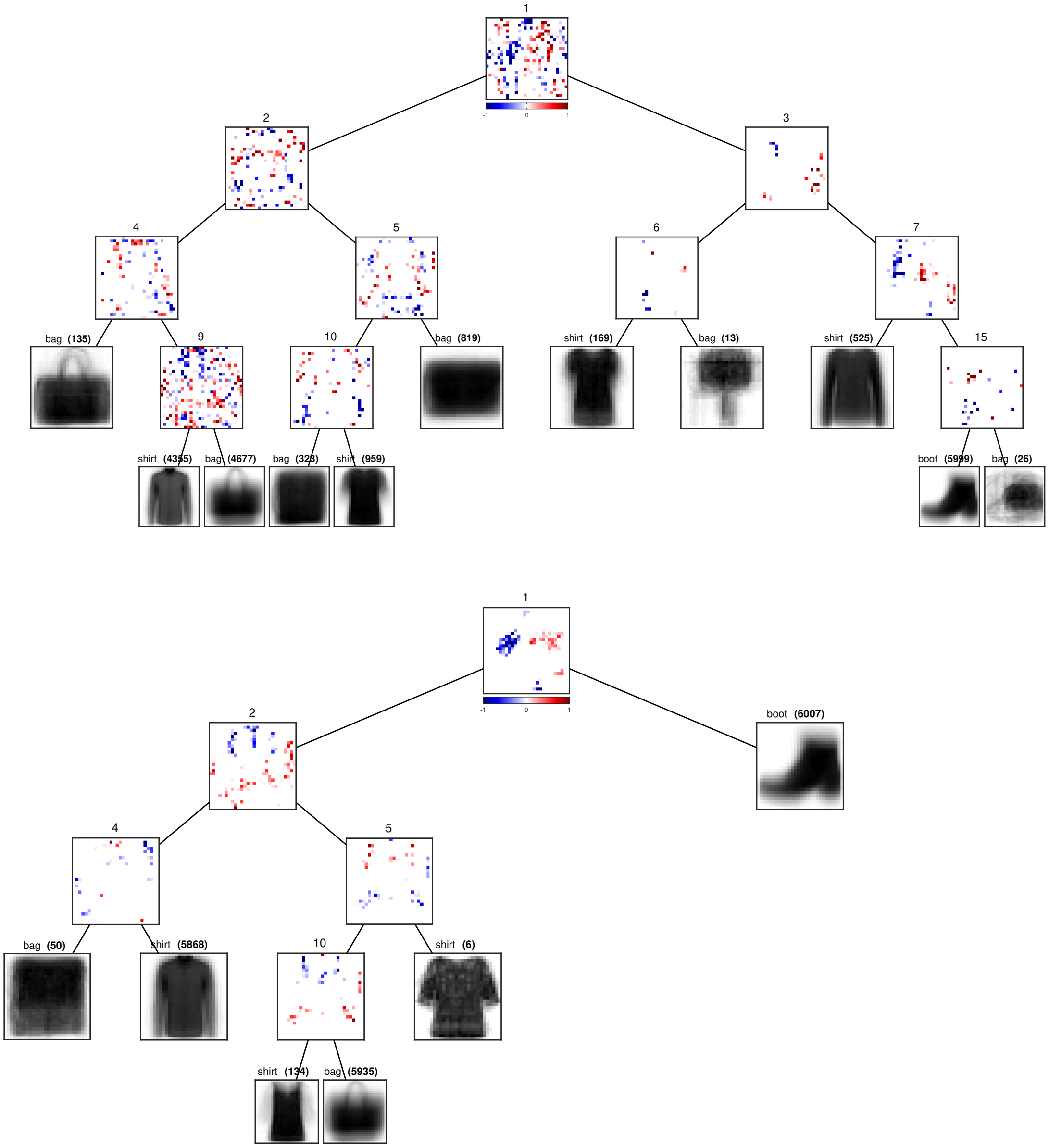


Figure 3.5: Some of the oblique trees obtained from LapTAO on Fashion-MNIST. Both figures use 10% of labeled data, but they differ in regularization penalty (α) on the tree: (*top*) $\alpha = 1$ with $E_{\text{test}} = 2.1\%$ and (*bottom*) $\alpha = 10$ with $E_{\text{test}} = 3.9\%$. At each decision node, we illustrate the weight vector of dimension 784 reshaped into 28×28 square where each value is colored according to their sign and magnitude (positive, negative and zero values are blue, red, and white, respectively). At each leaf, we show the class label, the total number of training points in that leaf (in brackets), and the average of input images in that leaf (as a greyscale image).

notice that such images do not contain any pixels in the top-left quadrant of the image. Therefore, the weight vector at the root has negative (blue) values in the corresponding elements and we know that negative values are responsible for sending an instance to the left. In other words, all images that have something in the top-left quadrant are sent to the left. Therefore, boots will end up in the right child. Similarly, node #2 sends most of the images that have M-shaped stroke in the top-center part (e.g. large bags, shirts with collar) to the left child. Following the same logic, we can obtain meaningful insights for each decision node. Also note that all nodes have majority of values equal to zero (thanks to the sparsity) which makes the interpretation easier.

Chapter 4

Decision Trees for Nonlinear Embeddings

In the previous chapter, we considered training decision trees in semi-supervised learning setup–one particular application where manifold regularization appears. Indeed, if we want to leverage unlabeled data, it is natural to take into account a geometrical shape of data and assume that similar instances have similar predictions, which is incorporated into an objective as a graph prior (e.g. graph Laplacian). Another common application of manifold regularization is *nonlinear dimensionality reduction*. Specifically, most of the methods for training *nonlinear embeddings* formulate the learning problem where manifold regularization is involved (e.g. pairwise and geodesic distances).

The reader may wonder why we may need to use decision trees in dimensionality reduction (DR) problems? The answer is directly related to the popular property of decision trees–interpretability. Model interpretability has seen much research in supervised settings (classification, regression) but much less in unsupervised ones, specifically in dimensionality reduction (DR) and nonlinear embeddings (NLE), which we focus on here. Two outputs of a DR procedure are relevant for interpretability. One is the low-dimensional projections of the data points (the *embedding*, which can be visualized (typically in 2D scatterplots) to find patterns in the data. This has been widely exploited and is indeed a main application of DR. The other output is the *projection mapping* from high- to low-dimensional points.

This has received some attention for linear models such as PCA but very little otherwise, in particular for NLE methods such as t-SNE, which do not naturally define an out-of-sample mapping, rather they directly learn a low-dimensional projection for each training point, by optimizing an objective function of the latter. A further consideration is that interpretability is not a black-and-white concept. In learning an explainable DR mapping, it is useful to be able to control the complexity of the explanation so as to span a range from a very accurate, detailed but complex explanation, to a less accurate but simpler explanation that may capture important overall patterns.

Let us call $\mathbf{F}$: $\mathbf{x} \in \mathbb{R}^D \rightarrow \mathbf{z} \in \mathbb{R}^L$ the *projection mapping*, which maps a high-dimensional point $\mathbf{x}$ in D dimensions to a low-dimensional point $\mathbf{z}$ in $L \ll D$ dimensions. This mapping is in general nonlinear and we seek a type of mapping that can be interpreted. Black-box mappings such as a neural net or a random forest, while highly accurate, are very hard to interpret in a robust way, in spite of many efforts in this direction, such as feature importance or saliency maps [83]. We focus here on models that are easier to interpret by construction. One candidate are linear mappings, but they are too restrictive and would distort the NLE considerably. Another candidate are generalized additive models [56], which define $\mathbf{F}$ as a sum of functions each operating on a single feature (or perhaps a pair of them). These functions can then be plotted to inspect them. This model is also very restrictive, in that features are generally expected to interact, and it scales poorly with the dimension D. A third candidate are traditional, axis-aligned decision trees, where each decision node thresholds a single input feature and has two children. This is also very restrictive for several reasons. First, the axis-aligned structure of the splits is wholly inadequate if features interact or have correlations, and it gives rise to a large number of nodes, which makes the tree hard to interpret. Besides, the maximum number of features that a binary tree with K leaves can use is $K-1$ (one per decision node), and each root-leaf path would use even fewer. When D is large, as is expected in DR, this would force the tree to have many nodes, or else it would apply a drastic feature selection on the embedding.

Then, our proposed mapping is a *sparse oblique tree*, which we discussed in detail in the previous chapters. This type of tree is ideal for our goal for several reasons. It can model nonlinear mappings using very few nodes compared to an axis-aligned tree. It is especially convenient when clusters exist in the data, which can be captured by the tree leaves. It can make full use of any and all features of an instance, and exactly which features will be used depends on the path the instance follows. And, crucially, we can control the tree complexity in number of nodes, number of features used in each decision node and number of nonzero weights in each leaf mapping via penalty hyperparameter. This offers a convenient way to achieve a range of explanation levels, from detailed and accurate to simple and less accurate. If penalty is large enough, the tree will collapse to a single leaf node, i.e., a sparse linear mapping. Indeed, and as seen in our experiments, the sparse oblique tree is much more accurate than an axis-aligned tree while using many fewer nodes, which makes the tree quite interpretable.

In this chapter, we consider the problem of optimally learning interpretable out-of-sample mappings for nonlinear embedding methods such as t-SNE. We first discuss (section 4.1) how to train an optimal mapping (a sparse oblique tree) jointly learned during the embedding process, not after the fact. However, this implies a difficult optimization problem, because the tree is not differentiable. We give an algorithm that solves this by alternatingly updating an embedding and a tree, and which works with any type of NLE, such as t-SNE [127], the elastic embedding [24], multidimensional scaling [18], the Sammon mapping [112] or others. Then, in section 4.2 we show that the resulting tree can indeed provide useful insight into the data beyond what the 2D visualization of the embedding itself provides.

4.1 Jointly learning an optimal tree and embedding

A nonlinear embedding (NLE) method defines an objective function $E(\mathbf{Z})$ over the low-dimensional coordinates $\mathbf{Z}_{L\times N} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$ of the high-dimensional training points $\mathbf{X}_{D\times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$. For example, for t-SNE [127] this is:

$$E(\mathbf{Z}) = \sum_{n=1}^{N} \mathrm{D}\left(P_n \| Q_n(\mathbf{Z})\right) = \sum_{n,m=1}^{N} p_{nm} \log \frac{p_{nm}}{q_{nm}(\mathbf{Z})} \tag{4.1}$$

while for the elastic embedding [24] it is:

$$E(\mathbf{Z}) = \sum_{n,m=1}^{N} \left(w_{nm} \|\mathbf{z}_n - \mathbf{z}_m\|^2 + \alpha e^{-\|\mathbf{z}_n - \mathbf{z}_m\|^2} \right) \tag{4.2}$$

where the specific terms are not relevant here (but they are instances of manifold regularization), what matters is that E is a function of the low-dimensional projections and that is what must be optimized over. Call the result the *free embedding*, in that it is not constrained to obey any particular mapping $\mathbf{F}$. However, if we want an out-of-sample mapping $\mathbf{F}$ so we can project new points, then $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n)$ for $n = 1, \dots, N$ by definition and so we have a *parametric embedding* objective function:

$$\min_{\mathbf{F}} P(\mathbf{F}) = E(\mathbf{F}(\mathbf{X})) + \lambda\, \phi(\mathbf{F}) \tag{4.3}$$

where $\phi(\mathbf{F})$ is a regularization term on the mapping. For example, for the elastic embedding $E(\mathbf{F}(\mathbf{X}))$ would be:

$$\sum_{n,m=1}^{N} \left(w_{nm} \|\mathbf{F}(\mathbf{x}_n) - \mathbf{F}(\mathbf{x}_m)\|^2 + \alpha e^{-\|\mathbf{F}(\mathbf{x}_n) - \mathbf{F}(\mathbf{x}_m)\|^2} \right) \tag{4.4}$$

Note that the process (which we call *direct fit*) of training a mapping directly to predict the free embedding, i.e., $\min_{\mathbf{F}} \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2$, will work poorly unless $\mathbf{F}$ is very flexible, and will result in a new embedding "$\mathbf{F}(\mathbf{X})$" that can considerably distort the optimal embedding. This is particularly true if we limit the complexity of $\mathbf{F}$ to make it more interpretable.

If $\mathbf{F}$ was differentiable, we could easily optimize (4.3) via gradient-based methods, as done for the Sammon mapping or MDS with a RBF network [85, 133], or for t-SNE with a neural net [124]. However, this is not possible with a tree, which defines a non-differentiable mapping. Instead, we apply the method of auxiliary coordinates (MAC) [28, 29] designed for problems involving nested functions as follows. We reformulate the problem in a way that allows us to derive an iterative algorithm that capitalizes on the fact that we can use TAO to train a regression tree and the original NLE algorithm to train the embedding.

First, consider the following constrained problem, where the nesting is broken:

$$\min_{\mathbf{Z},\mathbf{F}} E(\mathbf{Z}) + \lambda\,\phi(\mathbf{F}) \quad \text{s.t.} \quad \mathbf{Z} = \mathbf{F}(\mathbf{X}) \tag{4.5}$$

where E is the original, free embedding objective function. This is equivalent to the parametric embedding problem (4.3). A similar formulation was presented in [27], but here $\mathbf{F}$ is a nondifferentiable decision tree. We solve this using a penalty method, such as the quadratic-penalty method (in the experiments we use the augmented Lagrangian). This defines a new, unconstrained objective function of $\mathbf{Z}$ and $\mathbf{F}$:

$$\min_{\mathbf{Z},\mathbf{F}} E(\mathbf{Z}) + \lambda\,\phi(\mathbf{F}) + \mu\|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2. \tag{4.6}$$

Optimizing this for fixed $\mu > 0$ produces $(\mathbf{Z}_\mu, \mathbf{F}_\mu)$ which, as $\mu \to \infty$, progressively force the constraints to be satisfied while making the objective as large as possible. Note that, for $\mu \to 0^+$ (and $\lambda = 0$), this produces as $\mathbf{Z}_0$ the free embedding and as $\mathbf{F}_0$ a tree fitted to that embedding; this is what we called direct fit earlier. As μ increases, $\mathbf{Z}$ and $\mathbf{F}$ cooperatively reorganize to find a better solution of (4.5). Finally, we optimize (4.6) itself by alternating optimization over $\mathbf{Z}$ and $\mathbf{F}$. Over $\mathbf{Z}$, eq. (4.6) is the original embedding objective E but with a quadratic regularization term on $\mathbf{Z}$:

$$\min_{\mathbf{Z}} E(\mathbf{Z}) + \mu\|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2 \tag{4.7}$$

This can be easily solved by reusing an algorithm to optimize the original embedding (t-SNE, the elastic embedding or whatever), with a minor modification to handle the additional quadratic term. This is very convenient because we can capitalize on the literature of NLE optimization, specifically on algorithms that scale to large datasets [137, 126]. Over $\mathbf{F}$, the problem reduces to a regression fit which we solve using TAO:

$$\min_{\mathbf{F}} \|\mathbf{Z} - \mathbf{F}(\mathbf{X})\|^2 + \frac{\lambda}{\mu}\phi(\mathbf{F}) \tag{4.8}$$

The ability of TAO to take an initial tree and improve over it is essential here to make sure the step over $\mathbf{F}$ improves over the previous iteration, and to be able to use warm-start to speed up the computation.

```
input training set $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$;
$\mathbf{Z} \leftarrow$ fit the free embedding (eq. (4.1),(4.2), etc.);
$\mathbf{F} \leftarrow$ fit a tree to $(\mathbf{X}, \mathbf{Z}_0)$ via TAO algorithms;
$\boldsymbol{\beta} \leftarrow 0$ (initialize Lagrange multipliers);
for $\mu = \mu_0 < \mu_1 < \mu_2 < \cdots < \mu_{\max}$;
  repeat
    $\mathbf{F} \leftarrow \mathbf{F} + \frac{1}{\mu}\boldsymbol{\beta}$;
    optimize over $\mathbf{Z}$ given $\mathbf{F}$ (eq. (4.7));
    $\mathbf{Z} \leftarrow \mathbf{Z} - \frac{1}{\mu}\boldsymbol{\beta}$;
    optimize over $\mathbf{F}$ given $\mathbf{Z}$ (eq. (4.8)) via TAO algorithm;
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{Z} - \mathbf{F})$ (multipliers step);
  until stop
end for
return $\mathbf{F}$
```

Figure 4.1: Joint optimization framework for learning a tree and embedding (augmented Lagrangian version).

In summary, our algorithm alternates between training an embedding with a regularization term that pushes it towards the current tree predictions, and training a tree to fit the current embedding. As the penalty term μ increases, the embedding and the tree coadapt until they agree on an optimal result. Fig. 4.1 gives the pseudocode for the overall algorithm.

Relation to LapTAO and generalizing the framework One may notice a clear similarity between this algorithm and LapTAO described in chapter 3. Indeed, both of them are iterative algorithms involving two simple steps. The "tree-step" in LapTAO is similar to the $\mathbf{F}$ step above, i.e., both requires fitting a decision tree. The step over coordinates ($\mathbf{Z}$ step or "label-step") differs since for this problem we need to train embeddings whereas LapTAO simply solves a linear system. However, their essential meaning is the same: $\mathbf{Z}$ is non-parametric estimator of the underlying model which we try to "imitate" via decision tree. With this in mind, we can apply this framework more generally to any objective involving some form of manifold regularization. The steps are quite simple: 1) reformulate the problem by introducing variables and constraints for each instance; 2) move the constraints to an objective via quadratic penalty (or augmented Lagrangian); 3) apply alternating optimization where step over tree will involve fitting it in usual supervised fashion, whereas step over **Z** will generally require nonlinear optimization methods to solve.

4.2 Experiments

Our experimental findings demonstrate that 1) tree embedding as an out-of-sample mapping is quite accurate and helps to interpret the embeddings; 2) our optimization algorithm generally finds better optima compared to the naive direct fit. Moreover, we illustrate that tree embeddings can provide helpful insights (which are not covered by the original embeddings) about data. Please, refer to the companion paper for details of performed experiments [144].

4.2.1 Setup

Although our method generalizes to any type of nonlinear embeddings, we perform experiments on t-SNE and elastic embedding (EE). For both of them, we use our in-house implementation in Matlab so that we can easily handle **Z**-step of our algorithm. The reduced dimension is set to 2D. The details of the optimization are as follows: we use the spectral direction method [131] where the gradients of the embedding objective were approximated by the Barnes-Hut method [125] for t-SNE and the fast multipole method [132] for EE. We apply spectral directions until the relative error of the two recent iterates is less than 10^{-4}. For computing pairwise distances between input instances, we use entropic affinities with varying perplexity (K) depending on dataset [58] ($K = 15$ in most experiments). The α (see eq. (4.2)) parameter for the elastic embedding is set to $l = 100$ in all experiments.

We use TAO to train decision trees in **F**-step (eq. (4.8)) which is implemented in Python. All reported trees in this section are sparse oblique trees with linear leaves. An initial tree is a complete binary tree of given depth (Δ) with random parameters at each node (each node's weight vector has Gaussian (0,1) entries, and then we normalize the vector to unit length). We use 15 TAO iterations to train trees at each μ (also in direct fit). To optimize individual nodes, we use Lasso linear regression [55] in leaves and ℓ_1-regularized logistic regression in decision nodes (both implemented in scikit-learn [100]); and they use the same sparsity penalty (λ). Throughout this section, *direct fit* means a minimizer of $\min_{\mathbf{F}} \left\| \mathbf{Z}_0 - \mathbf{F}(\mathbf{X}) \right\|^2 + \lambda\, \phi(\mathbf{F})$, i.e., fitting a tree (with TAO) on the free embedding using the same λ penalty as our method. To train the direct fit, we initialize TAO from a complete tree of depth Δ with random parameters at each node. Maximum number of TAO iterations is set to 15. We denote our proposed method as the *tree embedding* which closely follows the implementation described in fig. 4.1. Initial value for $\boldsymbol{\beta}$ (estimate of the Lagrange multipliers) is set to zero. The initialization for $\mathbf{Z}$ is the *free embedding* (e.g. by running t-SNE on data). We use the direct fit as the beginning of the path (line 3 in fig 4.1). Empirically we found out that rescaling the penalty parameter in eq. (4.8) such that it always equals to λ (instead of $\frac{\lambda}{\mu}$) shows better performance, so we apply it in our experiments. After each AL step, we use warm-start in both steps by starting from the previous iteration's values. Our main running script is in Python which uses Matlab Engine API to run $\mathbf{Z}$-step.

4.2.2 Results

We first evaluate our method on the EE objective. We use a subset of 6 classes from 20 newsgroups document classification benchmark. Tf-idf statistics on unigrams and bigrams were used to represent each document (1000 features in total). Before running EE, we project data into 20 dimensions using PCA which helps us to eliminate noises. Pairwise distances were calculated using entropic affinities with perplexity $K = 15$. Then we run EE with homotopy parameter $l = 100$ to obtain the free embedding. For our method (tree embedding), we set the number of iterations to 15 and start from small value for $\mu_0 = 1e - 6$ which is multiplied by 1.4 after each iteration.

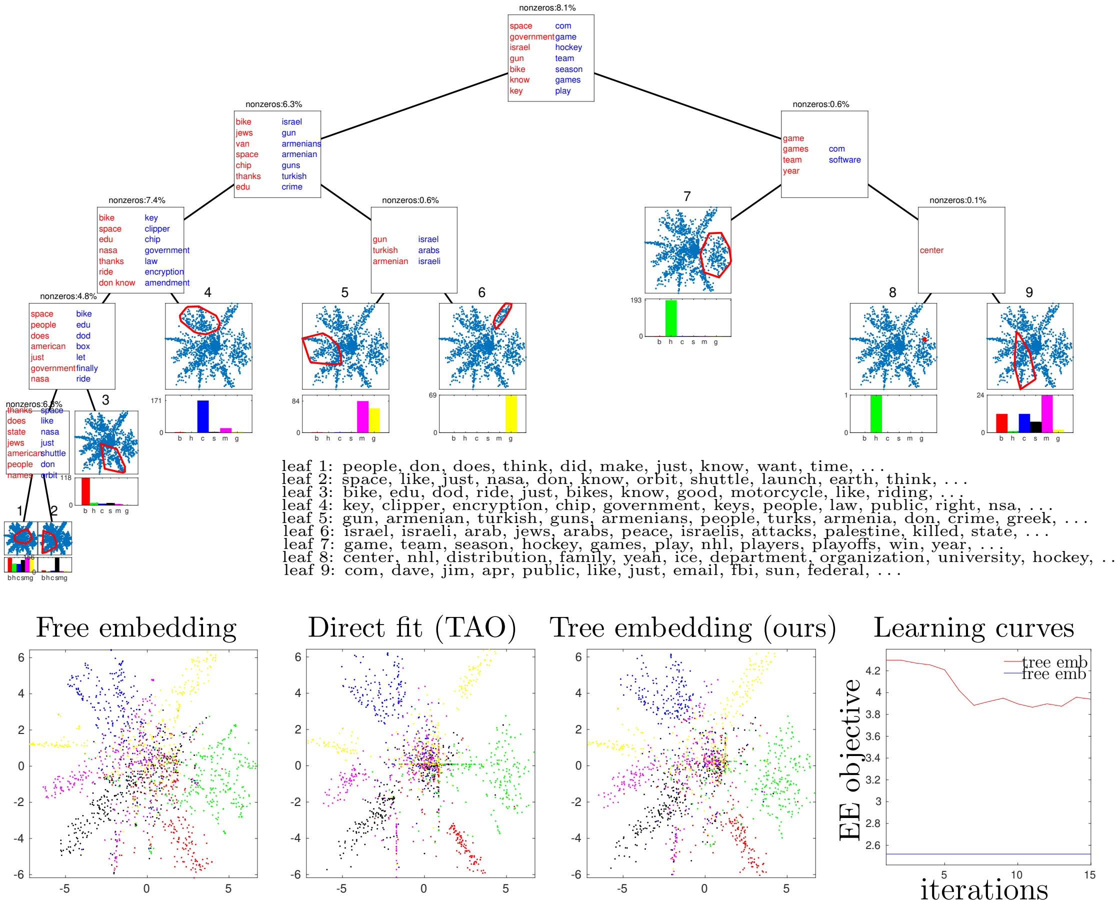
Fig. 4.2 (bottom) shows 2D embeddings obtained by several methods. Directly applying EE on preprocessed data (free embedding) yields the best results since it does not have any constraints. Next, we train a TAO tree to learn out-of-sample mapping either by the direct fit or by using our method (tree embedding). For both methods, we set the sparsity penalty for TAO $\lambda = 0.067$. However, the spectrum of various values for λ was explored in companion paper [144] (see animations). Tree embedding maps the data into 2D with a slight degradation compared to the free embedding, but noticeably better than the direct fit. The learning curves (bottom-right) align with that: the 1st iteration of the tree embedding is the direct fit and there is a clear improvement over iterations.

Fig. 4.2 (top) shows the visualization of the tree embedding (our method). Since each decision node is oblique (having hyperplane splits) and input features use tf-idf where each entry is a word, we show up to top-7 words which corresponds to the largest non-zero values in the weight vector. For each leaf, we show the region of its responsibility by using instances falling into that leaf (obtained via convex hull of the 2D mappings) and provide histogram counts of classes where encodings in x-axis mean: "b"–motorcycles, "h"–hockey, "c"–crypt, "s"–space, "m"–mideast and "g"–guns. According to the leaf regions, there is a clear clustering structure in the tree hierarchy because majority of leaves focus on few classes and this was obtained without explicit ground truth information. Moreover, the hierarchy respects class ontology by merging instances of similar classes under one subtree. For example, gun and mideast (leaf #5 and #6) are located under the same parent, whereas their locations in 2D are not next to each others. Explanation for such separation can be obtained from decision nodes: features (words) with the highest positive/negative values were responsible for sending an instance to a certain child. In some cases, only several words were enough to identify the next node (e.g. 4 words for leaf #7 and only 1 word for leaf #8). Occasionally, the tree provides some *surprising insights*: leaf #8 has only one point which lies within the region of leaf #7 (hockey). However, the tree decided to separate that point from the remaining group and assign it closer to leaf #9 (mixed classes). By closely inspecting that document, we have found that it discusses several topics (hockey team, donation form some person, private company name and a url) which seems to be an outlier. These types of insights are *not possible to infer using embeddings only.* Finally, for each leaf, we collect all documents reaching that leaf and show the most frequent words (text at the bottom of the tree). Although such information can be extracted from embeddings as well, we argue that this alone is not sufficient for interpretability. For example, top words in leaf #1 are quite generic words (mostly verbs) and it is understandable since documents in that group are from all 6 classes. But they are not insightful to determine why all these documents ended up in that region. On the other hand, the hierarchy allows us to trace the root-to-leaf path and identify region-specific words by observing weights at each decision node.

Similar conclusion can be made for fig. 4.3 which is for the "breast cancer" dataset. Our approach leads to the parametric embedding with a high quality (bottom panel) and tree allows us to interpret the mapping. Highlighted region at each leaf makes sense and in agreement with data. Surprisingly, leaf #1 covers the part of leaf #3. However, careful inspection shows that the border of leaf #3 contains a lot of patients from benign class and thus the algorithm decided to assign that region to the first leaf. Again, such observations are not trivial to detect using only embeddings as we see clear separation into two clusters. In this experiment, we run t-SNE to obtain the free embedding. We use the original input features (no PCA) to compute pairwise distances and apply entropic affinities with perplexity $K = 15$. To train the tree embedding, we set $\mu_0 = 1e - 5$ and multiply by 1.4 after each step (20 iterations in total).

4.2.3 Direct fit using CART

The naive way to get the tree embedding is to first train the free embedding (e.g. via EE, t-SNE, etc.) and then fit the traditional decision tree (e.g. using CART, C4.5) to learn the mapping. However, apart from being suboptimal, we show that it can be impractical. Potentially, we can fully grow a tree which will perfectly match the free embedding. But it is well known that such model hugely overfits and the final tree could be very deep making it hard to interpret. Instead, it


leaf 1: people, don, does, think, did, make, just, know, want, time, . . .
leaf 2: space, like, just, nasa, don, know, orbit, shuttle, launch, earth, think, . . .
leaf 3: bike, edu, dod, ride, just, bikes, know, good, motorcycle, like, riding, . . .
leaf 4: key, clipper, encryption, chip, government, keys, people, law, public, right, nsa, . . .
leaf 5: gun, armenian, turkish, guns, armenians, people, turks, armenia, don, crime, greek, . . .
leaf 6: israel, israeli, arab, jews, arabs, peace, israelis, attacks, palestine, killed, state, . . .
leaf 7: game, team, season, hockey, games, play, nhl, players, playoffs, win, year, . . .
leaf 8: center, nhl, distribution, family, yeah, ice, department, organization, university, hockey, . . .
leaf 9: com, dave, jim, apr, public, like, just, email, fbi, sun, federal, . . .

nonzeros:8.1%
space government israel gun bike know key
com game hockey team season games play
nonzeros:6.3%
bike jews van space chip thanks edu
israel gun armenians armenian guns turkish crime
nonzeros:0.6%
game games team year
com software
nonzeros:7.4%
bike space edu nasa thanks ride don know
key clipper chip government law encryption amendment
nonzeros:0.6%
gun turkish armenian
israel arabs israeli
nonzeros:0.1%
center
nonzeros:4.8%
space people does american just government nasa
bike edu dod box let finally ride
nonzeros:6.3%
thanks does state jews american people names
space like nasa just shuttle don orbit
1
2
3
4
5
6
7
8
9
156
118
171
84
69
193
1
24
0
b h c s m g
Free embedding
Direct fit (TAO)
Tree embedding (ours)
Learning curves
6
4
2
0
-2
-4
-6
-5
0
5
EE objective
iterations
tree emb
free emb
4.2
4
3.8
3.6
3.4
3.2
3
2.8
2.6
10
15

Figure 4.2: 20 newsgroups using EE objective. *Bottom*: 2D projections using the free embedding, direct fit and tree embedding (ours). Right figure shows the objective value over iterations. *Top*: Final tree embedding. Decision nodes show top-7 words with the largest positive (blue) or negative (red) non-zero values. Each leaf shows the region of its responsibility (top) and the histogram of class distributions (bottom, see section 4.2.2 for the encoding of classes in x-axis). Text at the bottom of the tree shows the most frequent 10 words in documents falling into the corresponding leaf (leaf 1, leaf 2, etc.). Note: plots at leaves are the same as the 2nd right 2D embeddings (but with uniform color).

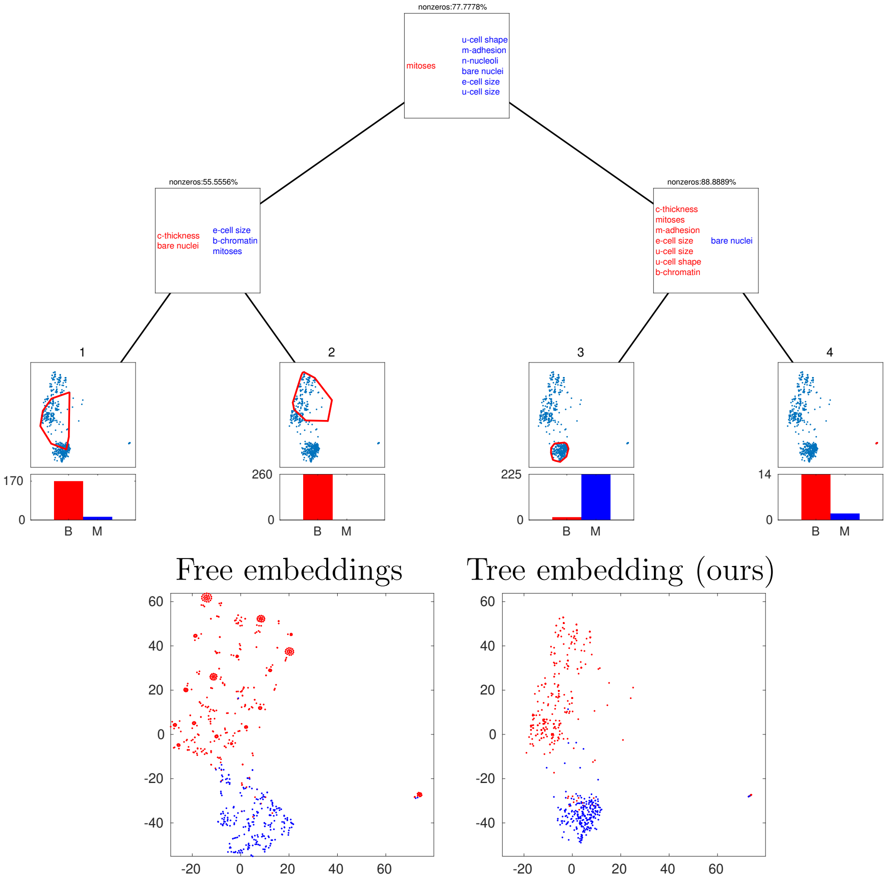


Figure 4.3: Similar to fig. 4.2 but for “breast cancer” using t-SNE.

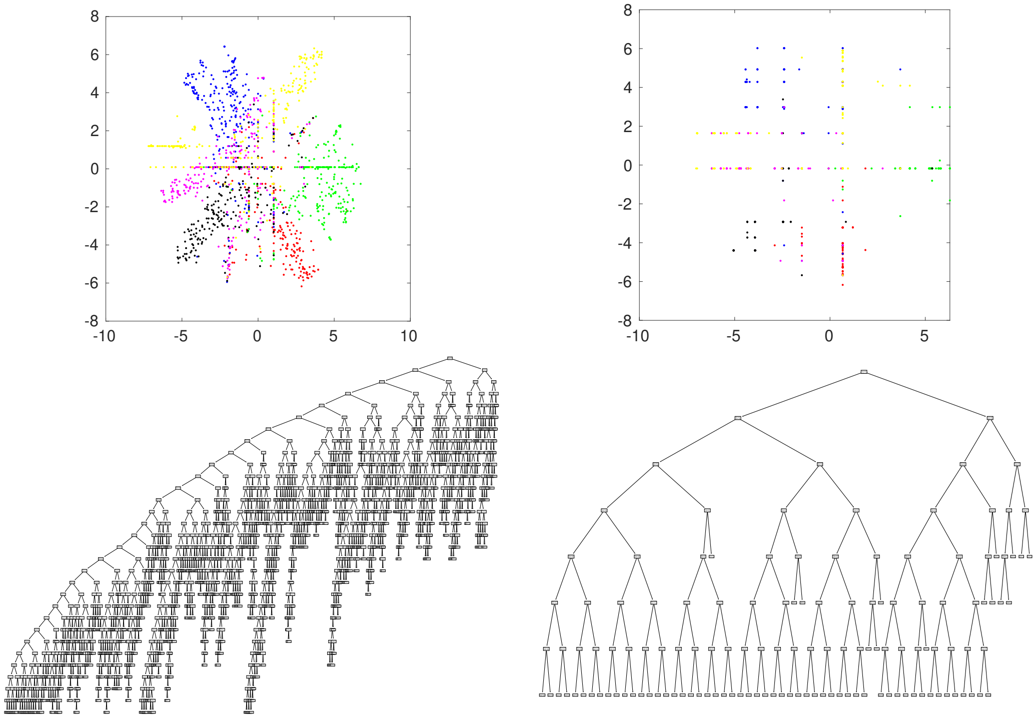


Figure 4.4: *Top*: 2D embeddings for 20 newsgroups provided by CART trees of depth 30 (left) and 7 (right). Each tree was trained using the free embedding (the same one as in fig. 4.2) as a ground truth (i.e., direct fit). *Bottom*: corresponding decision trees. It is clear that smaller (although more interpretable) CART trees lead to significant distortion of the embeddings.

makes sense to apply some pruning. We use scikit-learn's implementation of CART regression trees and apply pre-pruning strategy using 30% of data as validation set. Top plot in fig. 4.4 shows 2D embeddings obtained by CART for the same problem as in fig. 4.2. For the top left figure, although the general structure is preserved, we can clearly see artifacts in certain regions due to discrete nature of CART. Substantially reducing the depth causes a significant increase in loss (right plots). More importantly, bottom plot shows the visualization of the trees. Even for relatively simple problem as 20 newsgroups, the final pruned tree in the bottom left is very deep and contains $>$ 2400 nodes. Moreover, for regression problems, scikit-learn fits a separate tree for each output dimension and here we show one tree. It is clear that interpretability becomes non-trivial in such cases which practically limits this approach only to toy problems.

Chapter 5

Conclusion and Future Work

In this dissertation, we considered learning decision trees under loss functions involving manifold regularization. This type of regularization appears in a range of machine learning problems. Here, we covered two particular examples: semi-supervised learning and nonlinear dimensionality reduction. Using these two examples, we have shown how to reformulate the problem in a way that is amenable to iterative optimization. This gives us a generic algorithm to handle such type of problems. By introducing auxiliary variables, we isolate the difficult part (the tree optimization), and all the algorithm needs to do is to fit a regression tree in alternation with coordinates step (e.g. solving sparse linear system or fitting nonlinear embeddings). The tree fitting can be done reliably and efficiently using the Tree Alternating Optimizing (TAO) algorithm, which guarantees stability and allows us to use more powerful trees, such as sparse oblique trees.

In semi-supervised learning setting, our experimental results demonstrate that the algorithm can train accurate and interpretable decision trees even in extreme label scarcity situations. As for the nonlinear dimensionality reduction, we have argued for the use of sparse oblique trees as a convenient choice of explanatory low-dimensional mapping. By controlling the tree complexity (number of nodes and nonzero parameters) via an ℓ_1 penalty we achieve a range of solutions that span a tradeoff of accuracy and interpretability. Inspecting the tree we obtain insights about the data and about how high-dimensional instances are projected to the embedding, which go beyond the insights obtained by simply visualizing the embedding in 2D. In the remaining part, we discuss several ideas for future research projects.

This dissertation leads to several extensions and concerns for future research, which we describe below:

Semi-supervised learning for forests Undoubtedly, most successful applications of decision trees were explored by combining (or ensembling) them into a forest. These are among the most successful and widely recognized of all machine learning models: Random Forests, XGBoost, LightGBM, etc. Our proposed semi-supervised framework can be generalized to other machine learning models. Indeed, the target model ($T(\mathbf{x}; \boldsymbol{\Theta})$) appears in the algorithm in the tree-step, with the form of a regression problem having the smoothed labels as ground-truth. Obviously, we can use other regression models, such as random forests, gradient boosted trees, neural networks, etc. This has immediate practical importance as it can further expand practical usage of tree-based models.

Self-supervised learning with decision trees Representation learning with decision trees have got some attention in recent years [44] due to huge success in deep learning. The idea is to extract hierarchical representations of an input by stacking several layers of tree ensembles (forests). We can extend our proposed framework for this problem since the loss functions that are commonly used here still involve manifold regularization (variations of it): triplet loss, contrastive loss, etc. An extension might be non-trivial as the tree-step requires fitting multiple stacked forests. However, the idea of the method of auxiliary coordinates (MAC) [28, 29] can handle nested functions of arbitrary complexity, which can be applied here.

Theoretical properties of manifold regularization with decision trees In general, learning an optimal decision tree in most of its formulation is NP-hard problem. As of now, we do not have any approximation guarantees for TAO, let alone our proposed framework. Convergence of the algorithm also seems to be vague. Quadratic penalty based optimization methods have certain convergence theorems which apply with mild conditions, such as smoothness of a function and/or Lagrangian. However, applying such theorems is not possible here since decision trees operate on a discrete space. Furthermore, manifold regularization also assumes some degree of smoothness, which (generally) does not hold for tree-based models. One can potentially apply a certain mathematical tools from the theory of Reproducing kernel Hilbert space (RKHS) to study this issue.

Appendix A

Estimating Model Sizes and Inference FLOPS for Forests

For each forest (ensemble of trees) in chapter 2, we report its model size by counting the total number of parameters. Additionally, we report our estimate of FLOPS (floating point operations) for inference to measure a prediction time per input instance. Additional details on how to compute runtimes can be found in [63].

Total number of parameters We count the parameters for each node of each tree (decision nodes and leaves) and sum them up. In a decision node, we count the number of nonzero weights (and the bias), which is 2 for an axis-aligned tree and at most $D + 1$ for an oblique tree (where D is the number of features and additional scalar to store the bias term). In a leaf, we count K for constant-label leaves (where K is the output dimensionality, e.g. number of classes) and $(D+1)K$ if a leaf carries a linear model (e.g. a linear regressor or classifier). The exception is a binary classification problem ($K = 2$), where a single D dimensional vector (and the bias term) is sufficient to perform prediction in a leaf. Obviously, if the resulting ensembles are constructed using sparse decision trees (as in the case of TAO oblique trees), then the number of effective parameters in a node is reduced to the number of non-zeros entries (an their indexes). The are some baselines in our experiments where we do not have an access to the actual trees (due to their implementation). In such cases, we report an upper bound (marked with parentheses) which is computed as follows: for each tree of maximum depth Δ (but not necessarily complete), the maximum number of decision nodes and leaves is $\max(N, 2^{\Delta}) - 1$ and $\max(N, 2^{\Delta})$, respectively, where N is the number of training points. We then count the number of parameters as above and multiply it times the number of trees.

Inference FLOPS The inference time (FLOPS) for one instance along a tree is equal to the number of nonzero parameters it encounters in the root-to-leaf path it follows. For the entire forest, we simply sum an inference time of each individual tree. We repeat this process for all training instances and then average over all of them. In some situations where we do not have an access to the actual trees, we assume each root-to-leaf path to have depth Δ.

Appendix B

Datasets

This section provides the description of datasets used in our experiments. All data have dense features unless otherwise stated.

Letter English letter recognition task from UCI dataset [81]. Each letter image was generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts. The features are image features such as edge counts and statistical moments. We use the last 4 000 samples as test as in [117].

MNIST Handwritten digits recognition task [77]. The features are the pixel grayscale values in [0,1] of each 28×28 digit image. We use the training/test partition in [77].

fashion_mnist [136] is another benchmark dataset used for object recognition. It has similar characteristics as mnist (70k grayscale images of 28×28, 10 classes of different clothing items). We use this dataset primarily to compare LapTAO with LapSVM and to visualize the trained trees. Since the LapSVM has scalability issues for large number of points, we pick the subset of fashion_mnist (3 classes: “shirt”, “bag” and “ankle boot”) resulting in 18k training points.

Char74k Image classification task [38] where each image contains a character in one 64 classes (0–9, A–Z, a–z). We used the modified version of [117], as described in [141]: 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.

RCV1 Text categorization dataset [79]. We obtained it from the LIBSVM multiclass data collection[1]. The features are sparse normalized log TFIDF vectors.

SUSY Classification of particle detector collision events, available in the UCI dataset [81]. We randomly select 90% of the instances for training and the rest for test.

cpu_act Predict the portion of time that CPUs run in user mode given different system measures. We obtained it from the DELVE data collection[2].

ailerons Aircraft control action prediction[3]. The attributes describe the status of the aircraft and the target is the command given to its ailerons.

CT slice The attributes are histogram features (in polar space) of the Computer Tomography (CT) slice. The task is to predict the relative location of the image on the axial axis (in the range [0 180]). Available in the UCI Machine Learning Repository [81].

YearPredictionMSD A subset of the Million Song Dataset [11]. The task is to predict the age of a song from several song statistics given as metadata (timbre average, timbre covariance). Obtained from the UCI Machine Learning Repository [81].

ALOI (Amsterdam Library of Object Images) is a color image collection of one-thousand small objects, recorded for scientific purposes. Images for each object category are created by systematically changing viewing angle, illumination angle, and illumination color (see details here). We obtained the preprocessed form of the dataset from the LIBSVM multiclass data collection, where the extended color histogram with 128 dimensions is used to

[1] http://csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html

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

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

extract image features. We follow the same random partition of the data (90% train and 10% test) as in [34]. As a preprocessing step, we subtract the mean.

ODP (Open Directory Project) is the comprehensive human-edited directory of the website categories. As of April 2013 there were over 1M categories organized in a hierarchical ontology scheme. We use the preprocessed version of it[4] which uses 105k categories, as in [37, 86]. For each document, input feature vector is bag-of-words (normalized) and the class label is the category associated with the document.

WIKI–Small is another text categorization dataset obtained from [69]. It is a subset of Large Scale Hierarchical Text Classification challenge (LSHTC) [99]. For each document, a feature vector is bag-of-words and the class label is the category associated with the document obtained from DMOZ and DBpedia hierarchical ontology of the WEB.

PTB (Penn Treebank) is a standard dataset used to evaluate performances of language models. We use the preprocessed version from [90] which is publicly available online. The dataset consists of the plain text sentences in English with approximately 1M tokens and 10k unique words (i.e. vocabulary size). For the neural language modeling experiments, we proceed with this dataset as is without further modification (i.e. section 2.3.5). But for the section 2.3.5, we construct the dataset as follows. We filter out words that appeared less than 10 times which leaves us with 5 970 unique words (=number of classes). We construct a multiclass classification task as predicting the next word given previous 3 words. As for the input features, we use a pretrained version of GloVe [101][5] to obtain a word representation in vector space. We downloaded pretrained word vectors ($\in \mathbb{R}^{50}$) which were trained on Wikipedia 2014 and Gigaword 5. We obtain a word vector for each context word and simply concatenate them. For example, consider the

[4] http://hunch.net/~vw/odp_train.vw.gz, http://hunch.net/~vw/odp_test.vw.gz
[5] https://nlp.stanford.edu/software/

following sequence “black lives matter protest”. First, we extract 50 dimensional GloVe vectors for “black”, “lives”, “matter” and then concatenate them which results into 150 dimensional vector (i.e. this would be the total number of input features). The ground truth label would be a single integer: 4011 (assuming the index of the word “protest” is 4011 in our vocabulary).

20 newsgroups . We select a subset of 6 classes: ’rec.motorcycles’, ’rec.sport. hockey’, ’sci.crypt’, ’sci.space’, ’talk.politics.mideast’, ’talk.politics.guns’. The resulting documents are collected and transformed into tf-idf representations with top 1000 unigrams and bigrams. We use scikit-learn’s implementation for tf-ifd and set `min_df`= 3. Before that, we apply standard preprocessing steps: removing non-alphanumeric symbols, English stopwords and headers/footers from documents. To compute entropic affinities, we first project the data into 20 dimensions using PCA.

Breast cancer We normalize features to have values between [0,1]. Entropic affinities are computed directly on the original input features.

Bibliography

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

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

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. ACM*, 51(1):117–122, Jan. 2008.

[4] P. Auer, R. C. Holte, and W. Maass. Theory and applications of agnostic PAC–learning with small decision trees. In A. Prieditis and S. J. Russell, editors, *Proc. of the 12th Int. Conf. Machine Learning (ICML'95)*, pages 21–29, Tahoe City, CA, July 9–12 1995.

[5] J.-M. Begon, A. Joly, and P. Geurts. Globally induced forest: A prepruning compression scheme. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 420–428, Sydney, Australia, Aug. 6–11 2017.

[6] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Machine Learning Research*, 7:2399–2434, Nov. 2006.

[7] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 23, pages 163–171. MIT Press, Cambridge, MA, 2010.

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

[9] K. P. Bennett. Decision tree construction via linear programming. In *Proc. 4th Midwest Artificial Intelligence and Cognitive Sience Society Conference*, pages 97–101, 1992.

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

[11] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The Million Song Dataset. In *Proc. 12th Int. Society for Music Information Retrieval Conference (ISMIR 2011)*, pages 591–596, Miami, FL, Oct. 24–28 2011.

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

[13] A. Beygelzimer, J. Langford, and P. Ravikumar. Error-correcting tournaments. In *International Conference on Algorithmic Learning Theory (ALT 2009)*, pages 247–262, 2009.

[14] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. Sparse local embeddings for extreme multi-label classification. 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 730–738. MIT Press, Cambridge, MA, 2015.

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

[16] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer Series in Information Science and Statistics. Springer-Verlag, Berlin, 2006.

[17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Trans. Association for Computational Linguistics*, 5:135–146, 2017.

[18] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Application.* Springer Series in Statistics. Springer-Verlag, Berlin, second edition, 2005.

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

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

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

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

[23] P. Bühlmann and S. van de Geer. *Statistics for High-Dimensional Data.* Springer Series in Statistics. Springer-Verlag, 2011.

[24] M. Á. Carreira-Perpiñán. The elastic embedding algorithm for dimensionality reduction. In J. Fürnkranz and T. Joachims, editors, *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, pages 167–174, Haifa, Israel, June 21–25 2010.

[25] M. Á. Carreira-Perpiñán. The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models. arXiv, 2022.

[26] 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.

[27] M. Á. Carreira-Perpiñán and M. Vladymyrov. A fast, universal algorithm to learn parametric nonlinear embeddings. 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 253–261. MIT Press, Cambridge, MA, 2015.

[28] M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. arXiv:1212.5921, Dec. 24 2012.

[29] M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. In S. Kaski and J. Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 10–19, Reykjavik, Iceland, Apr. 22–25 2014.

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

[31] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intelligent Systems and Technology*, 2(3):27, Apr. 2011.

[32] J. Chen, J. Mueller, V. N. Ioannidis, S. Adeshina, Y. Wang, T. Goldstein, and D. Wipf. Does your graph need a confidence boost? convergent boosted smoothing on graphs with tabular node features. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proc. of the 39th Int. Conf. Machine Learning (ICML 2022)*, Baltimore, MD, July 17–23 2022.

[33] 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.

[34] A. E. Choromanska and J. Langford. Logarithmic time online multiclass prediction. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28. MIT Press, Cambridge, MA, 2015.

[35] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, 2013.

[36] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012.

[37] H. Daumé III, N. Karampatziakis, J. Langford, and P. Mineiro. Logarithmic time one-against-some. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 923–932, Sydney, Australia, Aug. 6–11 2017.

[38] 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.

[39] A. Defazio, F. R. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 1646–1654. MIT Press, Cambridge, MA, 2014.

[40] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the *EM* algorithm. *J. Statistical Society B*, 39(1):1–38, 1977.

[41] M. Denil, D. Matheson, and N. de Freitas. Narrowing the gap: Random forests in theory and in practice. In E. P. Xing and T. Jebara, editors, *Proc. of the 31st Int. Conf. Machine Learning (ICML 2014)*, pages 665–673, Beijing, China, June 21–26 2014.

[42] J. Eisenstein. *Introduction to Natural Language Processing*. MIT Press, 2019.

[43] 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.

[44] J. Feng, Y. Yu, and Z.-H. Zhou. Multi-layered gradient boosting decision 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 3554–3564. MIT Press, Cambridge, MA, 2018.

[45] E. Frank and S. Kramer. Ensembles of nested dichotomies for multi-class problems. In *Proc. of the 21st Int. Conf. Machine Learning (ICML'04)*, pages 305–312, Banff, Canada, July 4–8 2004.

[46] 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.

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

[48] 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)*, New Orleans, LA, June 19–24 2022.

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

[50] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG–TR–96–1, University of Toronto, May 21 1996.

[51] 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.

[52] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2019.

[53] L. Han, Y. Huang, and T. Zhang. Candidates vs. noises estimation for large multi-class classification problem. In J. Dy and A. Krause, editors, *Proc. of the 35th Int. Conf. Machine Learning (ICML 2018)*, pages 1890–1899, Stockholm, Sweden, July 10–15 2018.

[54] T. Hancock, T. Jiang, M. Li, and J. Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, May 1 1996.

[55] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2015.

[56] 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.

[57] 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.

[58] G. Hinton and S. T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 857–864. MIT Press, Cambridge, MA, 2003.

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

[60] K.-U. Hoffgen, H. U. Simon, and K. S. Vanhorn. Robust trainability of single neurons. *J. Computer and System Sciences*, 50(1):114–125, Feb. 1995.

[61] 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.

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

[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] O. İrsoy and E. Alpaydın. Unsupervised feature extraction with autoencoder trees. *Neurocomputing*, 258:63–73, Oct. 4 2017.

[65] O. İrsoy, O. T. Yıldız, and E. Alpaydın. Soft decision trees. In *Proc. 21st Int. Conf. Pattern Recognition (ICPR'12)*, pages 1819–1822, Tsukuba Science City, Japan, Nov. 11–15 2012.

[66] S. Ivanov and L. Prokhorenkova. Boost then convolve: Gradient boosting meets graph neural networks. In *Proc. of the 9th Int. Conf. Learning Representations (ICLR 2021)*, Online, Apr. 25–29 2021.

[67] Y. Jernite, A. Choromanska, and D. Sontag. Simultaneous learning of trees and representations for extreme classification and density estimation. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 1665–1674, Sydney, Australia, Aug. 6–11 2017.

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

[69] B. Joshi, M. R. Amini, I. Partalas, F. Iutzeler, and Y. Maximov. Aggressive sampling for multi-class to binary reduction with applications to text classification. 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. MIT Press, Cambridge, MA, 2017.

[70] N. Kambhatla and T. K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, Oct. 1997.

[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] C. Kemp, T. L. Griffiths, S. Stromsten, and J. B. Tenenbaum. Semi-supervised learning with trees. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 16, pages 257–264. MIT Press, Cambridge, MA, 2004.

[73] P. Koehn. *Neural Machine Translation*. Cambridge University Press, 2020.

[74] 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.

[75] M. Kuhn, S. Weston, M. Culp, N. Coulter, and R. Quinlan. C50: C5.0 decision trees and rule-based models. R package version 0.1.2, May 22 2018. Available online at https://cran.r-project.org/package=C50.

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

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

[78] J. Levatić, M. Ceci, D. Kocev, and S. Džeroski. Semi-supervised classification trees. *J. Intelligent Information Systems*, 49:461–486, 2017.

[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] A. H. Li and A. Martin. Forest-type regression with general losses and robust forest. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 2091–2100, Sydney, Australia, Aug. 6–11 2017.

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

[82] 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.

[83] Z. C. Lipton. The mythos of model interpretability. *Comm. ACM*, 81(10):36–43, Oct. 2018.

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

[85] D. Lowe and M. E. Tipping. Feed-forward neural networks and topographic mappings for exploratory data analysis. *Neural Computing & Applications*, 4(2):83–95, June 1996.

[86] T. K. R. Medini, Q. Huang, Y. Wang, V. Mohan, and A. Shrivastava. Extreme classification in log memory using count-min sketch: A case study of Amazon search with 50M products. 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 13265–13275. MIT Press, Cambridge, MA, 2019.

[87] B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. On oblique random forests. In R. L. de Mántaras and E. Plaza, editors, *Proc. of the 11st European Conf. Machine Learning (ECML–00)*, pages 453–469, Barcelona, Spain, May 31 – June 2 2000.

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

[89] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černocký. Empirical evaluation and combination of advanced language modeling techniques. In P. Cosi, R. De Mori, G. Di Fabbrizio, and R. Pieraccini, editors, *Proc. of Interspeech'11*, pages 605–608, Florence, Italy, Sept. 27–31 2011.

[90] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur. Recurrent neural network based language model. In T. Kobayashi, K. Hirose, and S. Nakamura, editors, *Proc. of Interspeech'10*, pages 1045–1048, Makuhari, Japan, Sept. 26–30 2010.

[91] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C.

Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 3111–3119. MIT Press, Cambridge, MA, 2013.

[92] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In D. Koller, Y. Bengio, D. Schuurmans, L. Bottou, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 21, pages 1081–1088. MIT Press, Cambridge, MA, 2009.

[93] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In R. G. Cowell and Z. Ghahramani, editors, *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 246–252, Barbados, Jan. 6–8 2005.

[94] K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* Adaptive Computation and Machine Learning Series. MIT Press, 2012.

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

[96] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

[97] M. Norouzi, M. Collins, D. J. Fleet, and P. Kohli. CO2 forest: Improved random forest by continuous optimization of oblique splits. arXiv:1506.06155, June 24 2015.

[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] I. Partalas, A. Kosmopoulos, N. Baskiotis, T. Artières, G. Paliouras, É. Gaussier, I. Androutsopoulos, M.-R. Amini, and P. Gallinari. LSHTC: A benchmark for large-scale text classification. arXiv:1503.08581, 2015.

[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] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *Proc. ACL–14 Conf. Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543, Doha, Qatar, 2014.

[102] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, Oct. 1988.

[103] Y. Prabhu and M. Varma. FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proc. of the 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2014)*, pages 263–272, New York, NY, Aug. 24–27 2014.

[104] 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.

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

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

[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] J. J. Rodríguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, Oct. 2006.

[109] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers—a survey. *IEEE Trans. Systems, Man, and Cybernetics Part C—Applications and Reviews*, 35(4):476–487, Nov. 2005.

[110] L. Rokach and O. Maimon. *Data Mining With Decision Trees. Theory and Applications.* Number 69 in Series in Machine Perception and Artificial Intelligence. World Scientific, 2007.

[111] S. Roweis, L. Saul, and G. Hinton. Global coordination of local linear models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pages 889–896. MIT Press, Cambridge, MA, 2002.

[112] J. W. Sammon, Jr. A nonlinear mapping for data structure analysis. *IEEE Trans. Computers*, C–18(5):401–409, May 1969.

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

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

[115] M. Schmidt, N. L. Roux, and F. R. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162:83–112, 2017.

[116] S. Schulter, C. Leistner, P. Wohlhart, P. M. Roth, and H. Bischof. Alternating regression forests for object detection and pose estimation. In *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, pages 417–424, Sydney, Australia, Dec. 1–8 2013.

[117] 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.

[118] W. Sun, A. Beygelzimer, H. Daumé III, J. Langford, and P. Mineiro. Contextual memory trees. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 6026–6035, Long Beach, CA, June 9–15 2019.

[119] J. Tanha, M. van Someren, and H. Afsarmanesh. Semi-supervised self-training for decision tree classifiers. *Int. J. Machine Learning and Cybernetics*, 8:355–370, 2017.

[120] 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.

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

[122] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, Feb. 1999.

[123] P. Tzirakis and C. Tjortjis. T3C: Improving a decision tree classification algorithm's interval splits on continuous attributes. *Advances in Data Analysis and Classification*, 11(2):353–370, June 2017.

[124] L. J. P. van der Maaten. Learning a parametric embedding by preserving local structure. In D. van Dyk and M. Welling, editors, *Proc. of the 12th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2009)*, pages 384–391, Clearwater Beach, FL, Mar. 21–24 2009.

[125] L. J. P. van der Maaten. Barnes-Hut-SNE. In *Proc. of the 1st Int. Conf. Learning Representations (ICLR 2013)*, Scottsdale, AZ, May 2–4 2013.

[126] L. J. P. van der Maaten. Accelerating t-SNE using tree-based algorithms. *J. Machine Learning Research*, 15:3221–3245, Oct. 2014.

[127] L. J. P. van der Maaten and G. E. Hinton. Visualizing data using t-SNE. *J. Machine Learning Research*, 9:2579–2605, Nov. 2008.

[128] J. E. van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109:373–440, 2020.

[129] A. Verikas, A. Gelzinis, and M. Bacauskiene. Mining data with random forests: A survey and results of new tests. *Pattern Recognition*, 44(2):330–349, 2011.

[130] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Computer Vision*, 57(2):137–154, May 2004.

[131] M. Vladymyrov and M. Á. Carreira-Perpiñán. Partial-Hessian strategies for fast learning of nonlinear embeddings. In J. Langford and J. Pineau, editors, *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, pages 345–352, Edinburgh, Scotland, June 26 – July 1 2012.

[132] M. Vladymyrov and M. Á. Carreira-Perpiñán. Linear-time training of nonlinear low-dimensional embeddings. In S. Kaski and J. Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 968–977, Reykjavik, Iceland, Apr. 22–25 2014.

[133] A. R. Webb. Multidimensional scaling by iterative majorization using radial basis functions. *Pattern Recognition*, 28(5):753–759, May 1995.

[134] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In A. McCallum and S. Roweis, editors, *Proc. of the 25th Int. Conf. Machine Learning (ICML'08)*, pages 1168–1175, Helsinki, Finland, July 5–9 2008.

[135] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, fourth edition, 2016.

[136] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747, Sept. 15 2017.

[137] Z. Yang, J. Peltonen, and S. Kaski. Scalable optimization for neighbor embedding for visualization. In S. Dasgupta and D. McAllester, editors, *Proc. of the 30th Int. Conf. Machine Learning (ICML 2013)*, pages 127–135, Atlanta, GA, June 16–21 2013.

[138] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995)*, pages 189–196, 1995.

[139] O. T. Yıldız and E. Alpaydın. Omnivariate decision trees. *IEEE Trans. Neural Networks*, 12(6):1539–1546, Nov. 2001.

[140] L. Zhang, J. Varadarajan, P. N. Suganthan, N. Ahuja, and P. Moulin. Robust visual tracking using oblique random forests. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 5825–5834, Honolulu, HI, July 21–26 2017.

[141] 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.

[142] 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.

[143] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Learning a tree of neural nets. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21)*, pages 3140–3144, Toronto, Canada, June 6–11 2021.

[144] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Learning interpretable, tree-based projection mappings for nonlinear embeddings. In *Proc. of the 25th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2022)*, pages 9550–9570, Online, Mar. 28–30 2022.

[145] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Semi-supervised learning with decision trees: Graph Laplacian tree alternating optimization. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 35. MIT Press, Cambridge, MA, 2022.

[146] 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.

[147] 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.

[148] 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.

[149] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 16, pages 321–328. MIT Press, Cambridge, MA, 2004.

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

[151] Z.-H. Zhou and J. Feng. Deep forest: Towards an alternative to deep neural networks. In *Proc. of the 17th Int. Joint Conf. Artificial Intelligence (IJCAI'01)*, pages 3553–3559, Seattle, Washington, USA, Aug. 4–10 2001.

[152] H. Zhu, P. Murali, D. Phan, L. Nguyen, and J. Kalagnanam. A scalable MIP-based method for learning optimal multivariate decision trees. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 33, pages 1771–1781. MIT Press, Cambridge, MA, 2020.

[153] X. Zhu. Semi-supervised learning literature survey. Technical Report TR–1530, Dept. of Computer Science, University of Wisconsin-Madison, July 19 2008.

[154] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU–CALD–02–107, School of Computer Science, Carnegie-Mellon University, June 2002.

[155] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In T. Fawcett and N. Mishra, editors, *Proc. of the 20th Int. Conf. Machine Learning (ICML'03)*, pages 912–919, Washington, DC, Aug. 21–24 2003.