Trustworthy Learning using Uncertain Interpretation of Data

Jack H. Good

October 23, 2024
CMU-RI-TR-24-69

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:
Artur Dubrawski, *Chair*
Jeff Schneider
Tom Mitchell
Gilles Clermont, *University of Pittsburgh*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

Motivated by the potential of Artificial Intelligence (AI) in high-cost and safety-critical applications, and recently also by the increasing presence of AI in our everyday lives, Trustworthy AI has grown in prominence as a broad area of research encompassing topics such as interpretability, robustness, verifiable safety, fairness, privacy, accountability, and more. This has created a tension between simple, transparent models with inherent trust-related benefits and complex, black-box models with unparalleled performance on many tasks. Towards closing this gap, we propose and study an uncertain interpretation of numerical data and apply it to tree-based models, resulting in a novel kind of fuzzy decision tree called Kernel Density Decision Trees (KDDTs) with improved performance, enhanced trustworthy qualities, and increased utility, enabling the use of these trees in broader applications. We group the contributions of this thesis into three pillars.

The first pillar is robustness and verification. The uncertain interpretation, by accounting for uncertainty in the data, and more generally as a kind of regularization on the function represented by a model, can improve the model with respect to various notions of robustness. We demonstrate its ability to improve robustness to noisy features and noisy labels, both of which are common in real-world data. Next, we show how efficiently verifiable adversarial robustness is achievable through the theory of randomized smoothing. Finally, we discuss the related topic of verification and propose the first verification algorithm for fuzzy decision trees.

The second pillar is interpretability. While decision trees are widely considered to be interpretable, good performance from tree-based models is often limited to tabular data and demands both feature engineering, which increases design effort, and ensemble methods, which severely diminish interpretability compared to single-tree models. By leveraging the efficient fitting and differentiability of KDDTs, we propose a system of learning parameterized feature transformations for decision trees. By choosing interpretable feature classes and applying sparsity regularization,we can obtain compact single-tree models with competitive performance. We demonstrate application to tabular, time series, and simple image data.

The third pillar is pragmatic advancements. Semi-supervised Learning (SSL) is motivated by the expense of labeling and learns from a mix of labeled and unlabeled data. SSL for trees is generally limited to black-box wrapper methods, for which trees are not well-suited. We propose as an alternative a novel intrinsic SSL method based on our uncertain interpretation of data. Federated Learning (FL) is motivated by data sharing limitations and learns from distributed data by communicating models. We introduce a new FL algorithm based on function space regularization, which borrows concepts and methods from our formalism of uncertain interpretation. Unlike prior FL methods, it supports non-parametric models and has convergence guarantees under mild assumptions. Finally, we show how our FL algorithm also provides a simple utility for ensemble merging.

Acknowledgements

There are many to whom I would like to express my gratitude for their part in the journey that has culminated in this thesis.

First and foremost, to my advisor, Professor Artur Dubrawski. He took me in as a naive undergraduate who knew a lot less than he thought he did, and then brought me back as a doctoral student. I will never forget his unexpected phone call in the middle of the night to let me know I was admitted to the doctoral program. He taught me to think big, yet stay grounded and realistic. He gave strong advice, but remained supportive regardless of my course. He trusted me with the freedom to pursue our research goals in my own way. He was patient when internships, responsibilities as a student, pandemic-related challenges to productivity and motivation, and other interruptions made progress uneven. And his unfailing optimism and commitment to the lab and our work has helped me stay strong and weather unpredictable progress, disappointing results, skeptical reception of work, and the competitiveness of the field. These can wear down a young researcher without support. Thank you sincerely for your guidance and your investment in me.

To my committee members, Professors Jeff Schneider, Tom Mitchell, and Gilles Clermont. Thanks for your expertise and valuable feedback towards the improvement of my work and its presentation.

To my mentors. To Kyle, whose technical guidance and pragmatic perspectives on research taught me a lot. And to Nick, who took me in as a new student and, through our collaboration, gave me the platform from which to spin off my own research. And to whom I know I can go for sound, compassionate advice and a shoulder to lean on.

To my other collaborators. To Torin, Shyla, Łukasz, Vlad, Kacper, and Daria, who worked with me to explore new directions for my work. To Abby, who caught mistakes and filled gaps in math that was sometimes well out of my depth. To Merritt, who has greatly enhanced the quality of my code and taught me a lot about principled programming along the way.

To the Auton Lab community: Ceci and Mono, my wonderful office mates; Yeeho, my good friend and informer of free food; Emma, who joined us for our D&D antics; Angela, who took the initiative to organize gatherings outside the office; and everyone else I've had the good fortune to get to know. Thanks to all of you, the lab has come to feel like home.

To my parents Mary and Bill, who look out for me still and never let me forget to stay in touch and visit home. And to my brothers Eli and Zachary, my gaming buddies and best friends that can never get rid of me. These years I have been physically further away, and yet felt closer to my family than ever.

And lastly, to my partner Macy, who has been by my side through it all. Time with her has been a much-needed escape from the world of work and an impetus to get out, try new things, and explore. She has been patient and understanding when deadlines, internships, and travel kept me from seeing her. And her presence tempers

my bittersweet departure from CMU and the Auton Lab with excitement for the future.

The journey has been harder than I thought it would be, and for different reasons than I would ever have anticipated. And so the support of everyone around me has been a larger part of my success than I could have imagined. Thank you all deeply for making this possible.

This work has been partially supported by the U.S. Department of Homeland Security (awards 14DNARI00005, 16DNARI00025, 17DNERP00002, 18DNARI00031), U.S. Department of Energy (via Lawrence Livermore National Laboratory collaborative projects B622976, B649543, B649585, B662844), National Institutes of Health (awards R01NS124642, R01HL141916), U.S. Army (awards W911NF1820218, W911NF20D0002, W519TC23C0031), U.S. Defense Threat Reduction Agency (award HDTRA11310026), and by the Space Technology Research Institutes grant from NASA's Space Technology Research Grants Program.

Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

1.1 The Case for Simplicity in Machine Learning

Machine Learning (ML) can be understood as a microcosm of the scientific method, as the distillation of usable knowledge from observations, partially automated to relieve the burden of the difficult and laborious search for a good explanatory hypothesis. While there is no one universally agreed interpretation of the scientific method, if you look on, e.g., Wikipedia, you will find the process paraphrased below, which we correspond to the process of machine learning.

The Scientific Method	**Machine Learning**
1. Define a question	1. Define a task
2. Gather information	2. Collect data
3. Form an explanatory hypothesis	3. Train a model
4. Test the hypothesis and collect experimental data	4. Validate the model on held-out data
5. Analyze the data	5. Analyze performance metrics
6. Interpret the data, draw conclusions, make new hypotheses	6. Optimize performance metrics via model selection and tuning
7. Publish results and retest (usually by other scientists)	7. Evaluate on test data, deploy, and monitor the model

At step 3, there are always many equally explanatory hypotheses. This raises the question: what makes a good explanatory hypothesis? The obvious answer is that it must generalize, that is, it must pass steps 4 though 6, lest we return to step 3 and try again. The No Free Lunch Theorem [188] tells us that no hypothesis selection strategy is better than random without some kind of prior or assumptions, but both science and machine learning tend to do quite a bit better than random guessing, so there is something more at play. In science, this is often credited to the heuristic of Occam's Razor, also known as the principle of parsimony, that is, the somewhat vague and subjective but nonetheless useful notion that "the simplest explanation

is usually the best one". Not only are hypotheses selected in accordance with this principle more likely to hold up to experimental scrutiny, but they also tend to better satisfy qualitative criteria for a hypothesis to be useful. On models that explain physical phenomena, Stephen Hawking writes [90]:

> A model is a good model if it:
>
> 1. Is elegant
> 2. Contains few arbitrary or adjustable elements
> 3. Agrees with and explains all existing observations
> 4. Makes detailed predictions about future observations that can disprove or falsify the model if they are not borne out.

Similarly, Albert Einstein said "the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience." Thomas Kuhn writes also that "a theory should be fruitful of new research findings: it should, that is, disclose new phenomena or previously unnoted relationships among those already known" [106].

A good machine learning model exhibits many of the same properties. Beyond making accurate predictions, it is lightweight, transferable, and transparent enough to explain observations and predictions, provide insights in its application domain, and spark further inquiry. There was a time when the Occam's Razor paradigm, whether it was viewed as such or not, was valued in the field. A combination of domain expertise and statistical techniques applied to data preparation minimized the knowledge gap to be filled by simple models with well-understood learning algorithms. Decision trees in particular exemplify the paradigm. In their most essential form, rather than fix the model complexity and search for the best fit, they fix the level of fit and search for the least complex model. Perhaps it can be partially credited to this paradigm that classical methods, despite their limitations in representation power compared to modern methods, have remained popular in high-cost-of-failure domains such as healthcare, military, and finance, and tree-based methods remain state-of-the-art for tabular data problems [81].

However, the incredible developments in deep learning have shifted values in the ML community, particularly among researchers, towards large, complex end-to-end models. Without the prior of simplicity, tactics like overparameterization, stochastic optimization, and immense training set size broadly achieve good generalization while removing the need to carefully balance simplicity vs. expressivity. While there has been some progress understanding on a fundamental level why these can effectively replace the Occam's Razor principle to achieve generalization, it has been outpaced by the rapid development of methods. Moreover, this approach comes at the cost of much of the simplicity, elegance, and fruitfulness that, like a scientific theory, make a model good.

Now we see another shift in popular values toward trustworthy models with properties such as interpretability, robustness, verifiable safety, fairness, privacy, accountability, and more. These are challenging topics that largely become tractable if the model itself is simple. We therefore posit that, in an age when ML is under more scrutiny than ever before, the field is well-positioned to benefit from a return to the values of parsimony that propel the principled practice of science. It is a challenging course; it is often the case that, the simpler a solution (in this case, a model), the more complex the process of finding it. Da Vinci is believed to have said "simplicity is the ultimate sophistication". It is a difficult road, but one worth traveling. We must not forget that, despite popular perceptions among ML researchers, classical methods and the problems they are well-suited to solve are ubiquitous in the real-world application of ML, and there is ample room for innovation. Especially in light of advancements in ML research and computing technology, there are numerous low-hanging fruit. As for problems where classical methods are not well-suited, there is hope that, however large the gap between simple and complex models may be, it can be closed. We already see the gap narrowing through a combination of works that, on one hand, simplify and improve the trustworthiness of deep models, and on the other hand, advance the utility of classical and classically-inspired models.

This work belongs to the latter category. By introducing a formalism of interpreting data with uncertainty during learning and inference, we advance both the trustworthiness and utility of models with a particular emphasis on decision trees. Beyond their exemplification of the principle of parsimony, trees are widely applied and familiar to users with various backgrounds, they offer inherent trust-related benefits such as interpretability, and their structure is simple, compact, and sparsely-activated. The last point makes them amenable to methods and analyses that would be prohibitively costly or require approximations for more complex models. We hope that this work will serve as the foundation of continued research to squeeze the trustworthiness gap in ML ever smaller.

1.2 Thesis Overview

In Chapter 2, we introduce our formalism for interpreting data with uncertainty and its interpretations. Among these, we highlight that it is a form of regularization of the function represented by a model itself, making it applicable to any model class, not just parametric models. We propose algorithms for the efficient application of this formalism to Decision Trees (DTs), resulting in a novel form of Fuzzy Decision Tree (FDT) called Kernel Density Decision Trees (KDDTs), and show their superior performance over conventional DTs. We also discuss its potential application to other models.

In Chapter 3, we discuss robustness and verification, two pillars of Trustworthy AI (TAI). We show that, compared to conventional decision trees, KDDTs have

- reduced sensitivity to small changes in the data
- improved robustness to additive feature noise
- improved robustness to label noise
- improved robustness to adversarial perturbation of features, with efficient lower bounds on the robustness of individual predictions.

Together, these help to explain why KDDTs outperform DTs and understand best practices for their application. Finally, we cover the closely related topic of verification, that is, the automated process of proving or finding counterexamples to safety properties, such as adversarial robustness, of models. We show that verification of FDTs, like for most model classes, is NP-Complete, propose the first algorithm for verification of FDTs, and demonstrate its practicality despite theoretical inefficiency in the worst case.

In Chapter 4, we focus on interpretability, another pillar of TAI. KDDTs are unique in that they efficiently fit an inherently differentiable tree by conventional greedy tree growth. We leverage this to alternatingly fit a tree to transformed features and use the gradient of the impurity function to optimize the feature transformation. This enables the learning of trees with more expressive decision rules, such as Oblique Decision Trees (ODTs), which have linear decision rules. By choosing the class of feature transformation and regularizing it to meet domain-specific interpretability needs, we can learn interpretable tree-based models with both small size and good performance. Our approach is more flexible than previous methods, yields smaller trees with better performance, and maintains familiarity and favorable structural properties of the tree since KDDT fitting is a generalization of the classic CART fitting algorithm. We demonstrate its application to the interpretable classification of tabular, image, and time series data.

In Chapter 5, we apply the uncertain interpretation paradigm to advance the utility of decision trees, improving their viability where existing methods focus on parametric models and non-parametric models such as trees fall short. First, we propose a method for Semi-supervised Learning (SSL), that is, learning from both labeled and unlabeled data, where there are few existing methods for decision trees. Next, we introduce a new method of Federated Learning (FL) by regularizing models for agreement in function space. Unlike prior methods, which are limited to parametric models, it is model-agnostic. Moreover, since loss functions are generally convex in function space, it has convergence guarantees with mild assumptions; when combined with our uncertain interpretation formalism and a quadratic loss function, we have fast convergence close to the consensus optimum. As a result, our approach learns performant models in very few iterations with none of the vulnerability to client heterogeneity that plagues other FL algorithms. Finally, we show that a natural utility arising from our FL algorithm is merging ensemble models into a single model and demonstrate its utility with decision trees.

In Chapter 6, we summarize the contributions and conclusions of this work, reiterate its most important limitations, and lay the foundations for future work to build upon it.

In Appendix A, we define acronyms used throughout the text. When viewing this document digitally, each acronym is a link to its definition in Appendix A.

In Appendix B, we present lengthy proofs omitted from the main text.

In Appendix C, we describe additional details of experiments for the purpose of reproducibility.

In Appendix D, we provide additional experiment results.

Chapter 2

Uncertain Interpretation of Data

2.1 Definition

In machine learning based on Empirical Risk Minimization (ERM), given hypothesis space $\mathcal{H}$ of predictive functions $h : \mathcal{X} \to \mathcal{Y}$, probability distribution $p_{\mathcal{X},\mathcal{Y}}$ over $\mathcal{X} \times \mathcal{Y}$, and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, we aim to find a hypothesis $h \in \mathcal{H}$ that minimizes the risk $R[h] = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\mathcal{X},\mathcal{Y}}}[\ell(h(\mathbf{x}), \mathbf{y})]$. For this work, we assume $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{Y} \subseteq \mathbb{R}^q$; categorical features and class labels are one-hot encoded. In reality, $p_{\mathcal{X},\mathcal{Y}}$ is unknown, and instead we observe samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ drawn from $p_{\mathcal{X},\mathcal{Y}}$ that comprise the training data; the distribution of such samples is the empirical distribution $\hat{p}_{\mathcal{X},\mathcal{Y}}$. Then we choose h to minimize the empirical risk $\hat{R}[h] = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\mathcal{X},\mathcal{Y}}}[\ell(h(\mathbf{x}), \mathbf{y})] = \frac{1}{n}\sum_i \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i)$.

Our uncertain interpretation formalism further interprets each observed feature vector $\boldsymbol{x}_i$ as a random variable $\mathbf{x}_i \sim k(\cdot, \boldsymbol{x}_i)$ for some chosen k. This gives the empirical distribution $\hat{p}_k$ with density $\hat{p}_k(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum_i \mathbf{1}\{\boldsymbol{y} = \boldsymbol{y}_i\}k(\boldsymbol{x}, \boldsymbol{x}_i)$. With this interpretation, the marginal distribution $\hat{p}_k(\boldsymbol{x}) = \frac{1}{n}\sum_i k(\boldsymbol{x}, \boldsymbol{x}_i)$ is Kernel Density Estimation (KDE), a popular method for estimation of the true distribution of feature values $\boldsymbol{x}$ given observations drawn from it. Therefore, we refer to k as a *kernel*. Then the empirical risk is

$$\hat{R}_k[h] = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_k}[\ell(h(\mathbf{x}_i), \mathbf{y}_i)] = \frac{1}{n}\sum_{i\in[n]} \int_{\mathcal{X}} \ell(h(\boldsymbol{z}), \boldsymbol{y}_i)k(\boldsymbol{z}, \boldsymbol{x}_i)\ d\boldsymbol{z} \tag{2.1}$$

and, at inference time, the input $\boldsymbol{x}$ is optionally also interpreted as random, resulting in the prediction

$$f(\boldsymbol{x}) = E_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[h(\mathbf{x})] = \int_{\mathcal{X}} h(\boldsymbol{z})k(\boldsymbol{z}, \boldsymbol{x})\ d\boldsymbol{z}. \tag{2.2}$$

To summarize, we choose the hypothesis that minimizes the risk on a kernel density estimate, then optionally smooth predictions using the same kernel.

This notion of a "kernel" comes purely from KDE and should not be confused with the kernels used for the "kernel trick" in learning algorithms such as kernel Support Vector Machine (SVM). The kernel trick formulates a learning algorithm such that its only dependency on the training data is only through inner products. Then, by replacing the inner product with a kernel function, one implicitly learns in a higher-dimensional or even infinite-dimensional space. There is no such implication in this work.

2.2 Interpretations

There are several different ways of interpreting the uncertain interpretation formalism defined in this section. These help to connect it to related concepts and research, and will also help to motivate its various uses throughout this work.

Modeling uncertainty. In Section 2.1, we define the formalism as interpreting the inputs to a model with uncertainty. This could, for example, model uncertainty in an observation due to known sources of noise. Another way to view it is that we estimate the underlying distribution using the KDE, rather than the usual empirical risk, to account for uncertainty resulting from random sampling of training data from the underlying distribution.

Function regularization. This formalism is a kind of regularization. Unlike typical regularization, which is applied to the parameters, this regularization is applied directly to the function itself represented by a model. Like parameter regularization, it can

- prefer simple (smooth) models
- reduce overfitting and improve robustness to noise
- reduce the variance of the estimate while introducing bias (shrinkage).

We study these topics in greater depth throughout the thesis. In particular, in Section 2.3.6, we benchmark the effect of this regularization on the performance of decision trees, and throughout Chapter 3, we study the the impact of this regularizing effect on various notions of model robustness. We also extend this concept of function regularization to penalize the difference of models, which we apply to decentralized federated learning in Section 5.2 and ensemble merging in Section 5.3.

Increasing margins. In classification, the term "margin" is sometimes used to refer to the distance from a point to the decision boundary. Large-margin predictions are usually desirable and associated with good generalization to unseen data. However, besides notable exceptions such SVM, most learning algorithms have no explicit objective to promote large-margin predictions, instead relying on low-power hypothesis classes (e.g. linear models), optimization tricks such as over-parametrization and

randomized optimization (e.g. neural networks), or ensemble algorithms such as bagging and boosting (e.g. tree ensembles). By spreading the risk over a locality of each training data point, we penalize any case where the decision boundary passes through that locality, creating a kind of large-margin objective. This can be formalized using the randomized smoothing framework, a technique for achieving and computing the adversarial robustness (large margin) of predictions. We study this topic in Section 3.4.

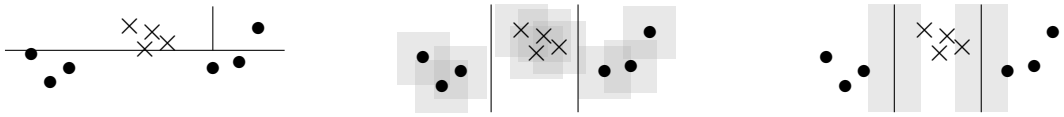
Smoothing models. The uncertain interpretation, especially during prediction, directly enforces smoothness of the model. If the kernel is stationary, that is, depends only on the difference between inputs, then the model is effectively smoothed by convolution with the kernel, a common strategy for smoothing of, for instance, image and time series data. Smooth models are useful in many ways. The scores output by a classifier are often interpreted as a probability distribution over the classes, indicating a kind of confidence of the prediction; if the highest score is close to 1, then the prediction is very confident, whereas if it is lower, and especially if it is close to the next-highest score, then the prediction is not confident. However, powerful models can tend to be overconfident when interpreted in this way. Neural networks are known to tend toward overconfidence [83]; decision trees, when fully grown, always predict with full confidence unless there are differently-labeled data with identical features in the training set, and even with pruning, the predicted score at a given leaf is discrete and often based on few samples. Smoothness means a continuous, gradual change in confidence when moving through the space, potentially tempering overconfidence and enforcing the idea that similar inputs should have similar predictions. Smoothness also ensures that a model is differentiable, and that the gradient is locally descriptive of the model, which improves its utility.

2.3 Decision Trees

Decision Trees (DT) are the most natural candidate for the first application of the uncertain interpretation formalism and are the focus of much of the content of this thesis. We apply our uncertain interpretation to define a generalization of Classification and Regression Trees (CART) [25] that we call KDDTs and propose accompanying fitting and prediction algorithms. This section describes KDDTs as originally published in [77] with some added discussion.

2.3.1 Background and Motivation

Decision trees are among the oldest and most universally known and implemented model classes for both classification and regression in machine learning. They offer numerous benefits: they are fast to train and make predictions without any specialized hardware, relatively small in memory usage, easy to apply and tune, flexible in their non-parametric, variable-resolution structure, easy to verify for robustness and safety



(a) The partition from fitting a crisp decision tree.

(b) The partition from fitting a KDDT with box kernels.

(c) Predicted value transitions smoothly in the bands.

Figure 2.1: An example to illustrate how KDDTs differ from crisp decision trees.

properties, and intuitive to humans, both holistically as a hierarchical partition of their input space and on the prediction level as a series of simple decision rules.

However, basic trees are rarely used in practice due to several weaknesses, foremost of which is their sensitivity to randomness of sampling and noisy observation, resulting in a tendency to overfit to training data. Moreover, they are limited in their compatibility with modern machine learning methods because of, for example, their non-parametric nature, their lack of meaningful differentiability, and their general limitation to tabular data.

The first limitation, at least, is typically resolved using ensembles of trees, which are a modern staple of tabular machine learning. These include ensembles with randomization strategies to reduce overfitting, such as random forests [26] and ExtraTrees [68], and boosted ensembles of small trees, such as AdaBoost [62], and XGBoost [35]. The tradeoff of ensembles is that their increased complexity results in longer training time, larger memory requirements, reduced interpretability, and slower safety verification (shown to be NP-Complete by [100]).

We propose KDDTs to augment, and sometimes even replace, these methods while relaxing additional limitations of tree-based models, providing trust-related benefits, and improving the practical usability of trees, as studied throughout the rest of this thesis. KDDTs belong to a less well-known family of tree-based methods wherein a decision may take multiple paths and a prediction is a weighted combination of leaves, variously called fuzzy decision trees, soft decision trees, differentiable decision trees, and neural decision trees. We use Fuzzy Decision Tree (FDT) as an umbrella term. Most use soft partitions at each node that smoothly transition the allocation of the decision from one subtree to another based on a learned splitting function. Many grow a tree using a standard DT algorithm, then replace splits with fuzzy splits. Depending on the algorithm, it may also freeze the tree structure, replace the splits with parametric splitting models, and train the parameters using gradient-based optimization. Many support fuzzy data, and some require it; prior methods can find optimal splits only for fuzzy categorical data because, unlike when fitting crisp trees, the gain for fuzzy splits on numerical data is continuous in the threshold value, ruling out exhaustive search for finding the optimal threshold. Alongside the KDDT formalism, we propose a fuzzy generalization of CART that is the first algorithm for efficiently finding optimal splits on continuous features for FDTs, including but not limited to KDDTs.

2.3.2 Related Work: Fuzzy Decision Trees

FDTs, unlike standard decision trees, are based on partitions that allocate a decision partially to multiple subtrees rather than wholly to one. The allocation is typically based on a learned splitting function. Starting with [32], many variations of the concept have been proposed over the years. We focus on those that build a tree greedily as with crisp trees and refer the reader to [36, 6, 161] for overviews of work in this area. In particular, we mention a few paradigms for fitting to highlight the difference in our approach. Most methods are based on fuzzy sets and only handle categorical features natively, so continuous data must be discretized [129]. Other approaches instead add fuzziness to partitions selected by algorithms for crisp trees [31] or fuzzify a crisp tree after it is completely fitted [41]. Our approach uses kernels to naturally represent fuzziness of continuous features without need for discretization and efficiently finds optimal partitions for the estimated distributions of data. The fuzziness is native to the learning process; there is no separate step to add it afterwards.

We also mention that there are numerous works learning fuzzy decision trees with a variety of splitting functions using gradient-based optimization. For example, Oblique Decision Trees (ODTs) use linear splits. These are less relevant to vanilla KDDTs; see Section 4.1.1 for more.

There are a few cases where KDE has been integrated with decision trees, but not to define fuzzy trees as in KDDTs. [159] propose a technique whereby prediction paths in a conventionally trained decision tree are used to choose features for KDE-based classification. This produces continuous predicted class probabilities (like KDE classifiers) and resists performance degradation due to the curse of dimensionality (like decision trees). [96] use one-dimensional KDE as features for training decision trees for one-class classification, which is used for tasks like outlier or anomaly detection. Other works use KDE only in the leaves and not to determine partitions [179, 135]. Our approach, unlike these, modifies the basic decision tree to fit directly to the density estimate and optionally make kernel-smoothed predictions.

2.3.3 Algorithms for Fuzzy Decision Trees

Recall that, given training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, we aim to fit a model that minimizes the risk on the class-dependent kernel density estimate

$$\hat{p}_k(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{\boldsymbol{y} = \boldsymbol{y}_i\} k(\boldsymbol{x}, \boldsymbol{x}_i)$$

where k is a chosen kernel. We propose here a generalization of CART for fitting to $\hat{p}_k$. It is a generalization in the sense that, if we let $k(\boldsymbol{z}, \boldsymbol{x}) = \delta(\boldsymbol{z} - \boldsymbol{x})$, then the fitting algorithm is just standard CART. Here δ is the Dirac delta function in $\mathbb{R}^p$, which has value zero everywhere except at zero, and which has integral 1 over $\mathbb{R}^p$. We call such k the "delta kernel" or "δ kernel". It can also be viewed as a generalization

of CART in the sense that it constructs the tree we would get in the limit as the number of samples approaches infinity by sampling from $\hat{p}_k$ and fitting using CART.

In order to do this efficiently, we make the following assumptions.

Assumption 2.1. For all $\boldsymbol{x} \in \mathbb{R}^p$, the function $k(\cdot, \boldsymbol{x})$ is a probability distribution on $\mathbb{R}^p$, that is, $k(\boldsymbol{z}, \boldsymbol{x}) \geq 0$ for all $\boldsymbol{z}$ and $\int_{\mathbb{R}^p} k(\boldsymbol{z}, \boldsymbol{x})\, d\boldsymbol{z} = 1$.

Assumption 2.2. For all $\boldsymbol{x} \in \mathbb{R}^p$, the distribution $k(\cdot, \boldsymbol{x})$ is isotropic, that is, it can be written in terms of its marginal distributions k_j as

$$k(\boldsymbol{z}, \boldsymbol{x}) = \prod_{j=1}^{p} k_j(z_j, \boldsymbol{x}).$$

To meet these assumptions, we typically use the following marginal kernels with a bandwidth parameter h:

- Delta kernel $k_j(z_j, \boldsymbol{x}) = \delta(z_j - x_j)$. This interprets an observation x_j without uncertainty and results in the conventional CART algorithm.
- Box kernel $k_j(z_j, \boldsymbol{x}) \propto \mathbf{1}\{z_j \in [x_j - h, x_j + h]\}$. This interprets an observation x_j as a uniform random variable $\mathcal{U}[x_j - h, x_j + h]$.
- Gaussian kernel $k_j(z_j, \boldsymbol{x}) \propto \exp\left(\frac{-(z_j - x_j)^2}{2h}\right)$. This interprets an observation x_j as a normal random variable $\mathcal{N}(x_j, h)$.

Marginals k_j and their respective bandwidths can be mixed and matched to form the multidimensional kernel k.

For fitting, but not for prediction, we need one additional assumption.

Assumption 2.3. For all $\boldsymbol{x} \in \mathbb{R}^p$, $j \in [p]$, $k_j(\cdot, \boldsymbol{x})$ is piecewise-constant.

Assumption 2.3 motivates the notion of using different kernels for fitting and prediction. For example, one may use a histogram approximation of a Gaussian kernel for fitting, then an equivalently scaled Gaussian kernel for prediction. One may also use, for instance, a box kernel for fitting and a delta kernel for prediction if they require unsmoothed predictions.

General Fuzzy Decision Trees

Before specifying the fitting and prediction algorithms, we define a more general notion of FDTs with which our algorithms are compatible. It encompasses both KDDTs and the majority of prior formulations of FDTs.

An FDT T is a hierarchical set nodes where each *internal* node i is characterized by the following:

- a left child $\ell_i \in [|T|]$

- a right child $r_i \in [|T|]$
- a splitting function $\sigma_i : \mathbb{R}^p \to [0, 1]$
- a value $\boldsymbol{v}_i \in \mathbb{R}^q$.

Each *leaf* node has only a value.

The splitting function determines the allocation of the decision to each subtree at a split. The tree's prediction is defined recursively as

$$f_i(\boldsymbol{x}) = \begin{cases} (1 - \sigma_i(\boldsymbol{x}))f_{\ell_i}(\boldsymbol{x}) + \sigma_i(\boldsymbol{x})f_{r_i}(\boldsymbol{x}) & i \text{ is internal} \\ \boldsymbol{v}_i & i \text{ is a leaf} \end{cases} \tag{2.3}$$

with recursion starting at the root. For conventional crisp decision trees, the splitting function is $\sigma_i(\boldsymbol{x}) = \mathbf{1}\{x_{a_i} > t_i\}$ for attribute index $a_i \in [p]$ and threshold $t_i \in \mathbb{R}$.

A common type of FDT, like conventional trees, uses a single feature and threshold at each node, but makes the transition of the decision allocation gradual. These axis-aligned FDTs typically have a single base splitting function $\sigma : \mathbb{R} \to [0, 1]$, for example, a sigmoid function, and define the node splitting functions as $\sigma_i(\boldsymbol{x}) = \sigma(x_{a_i} - t_i)$. As for KDDTs, for a node i with lower and upper bounds $\boldsymbol{l}, \boldsymbol{u} \in (\mathbb{R} \cup \{-\infty, +\infty\})^p$ defined by its ancestors' splits,

$$\sigma_i(\boldsymbol{x}) = \frac{K_j(t_i, \boldsymbol{x}) - K_j(l_i, \boldsymbol{x})}{K_j(u_i, \boldsymbol{x}) - K_j(l_i, \boldsymbol{x})} \tag{2.4}$$

where $K_j(\cdot, \boldsymbol{x})$ is the Cumulative Distribution Function (CDF)

$$K_j(t, \boldsymbol{x}) = \int_{-\infty}^{t} k_j(z, \boldsymbol{x})\, dz$$

of $k_j(\cdot, \boldsymbol{x})$. Since the bounds $\boldsymbol{l}$ and $\boldsymbol{u}$ depend on the splits of ancestor nodes, KDDTs have a dependency between splitting functions not seen in other FDTs.

FDTs can also be represented using *membership functions* $\mu_i : \mathbb{R}^p \to [0, 1]$. Let A_i be the set of ancestor nodes of i; further, let $A_{\ell,i}$ be the ancestors of which i is a left descendent and $A_{r,i}$ the ancestors of which i is a right descendent. Then the membership function is defined

$$\mu_i(\boldsymbol{x}) = \prod_{j \in A_{\ell,i}} (1 - \sigma_j(\boldsymbol{x})) \prod_{j \in A_{r,i}} \sigma_j(\boldsymbol{x}) \tag{2.5}$$

and indicates the degree of membership of input $\boldsymbol{x}$ to node i. Accordingly, it is easily shown that $\sum_{i \in \text{leaves}} \mu_i(\boldsymbol{x}) = 1$ for all $\boldsymbol{x}$.

For KDDTs, the membership value is the probability of membership in a node for a conventional decision tree given the uncertain interpretation of the input. In

particular, for a node i with lower and upper bounds $\boldsymbol{l}, \boldsymbol{u} \in (\mathbb{R} \cup \{-\infty, +\infty\})^p$,

$$\begin{aligned} \mu_i(\boldsymbol{x}) &= \mathbb{P}_{\mathbf{x} \sim k(\cdot, \boldsymbol{x})}[\mathbf{x} \in \mathcal{R}_i] \\ &= \int_{\mathcal{R}_i} k(\boldsymbol{z}, \boldsymbol{x}) \, d\boldsymbol{z} \\ &= \prod_{j \in [p]} K_j(u_j, \boldsymbol{x}) - K_j(l_j, \boldsymbol{x}). \end{aligned} \tag{2.6}$$

where $\mathcal{R}_i$ is the (possibly unbounded) hyperrectangle

$$\mathcal{R}_i = (l_1, u_1] \times \cdots \times (l_p, u_p]. \tag{2.7}$$

Thus the isotropic k, as in Assumption 2.2, makes computation of membership values efficient.

Given the membership value, the predictions can be written simply as

$$f(\boldsymbol{x}) = \sum_{i \in \text{leaves}} \mu_i(\boldsymbol{x}) \boldsymbol{v}_i \tag{2.8}$$

as an equivalent alternative to Equation 2.3.

Fitting Fuzzy Decision Trees

Here we present an efficient algorithm for fitting FDTs with a generalization of CART. While we present it generally, it is easily adapted for KDDTs using the equivalence specified in Equations 2.4 and 2.6.

Given training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ and stopping conditions on tree growth, the goal is to construct a tree and assign features $\boldsymbol{a}$ and thresholds $\boldsymbol{t}$ to minimize an impurity function, used like a risk functional, as

$$R[h] = \frac{1}{n} \sum_{i \in \text{leaves}} w_i \text{Impurity}(\boldsymbol{v}_i) \tag{2.9}$$

for tree-based hypothesis h, where w_i is the sample weight

$$w_i = \sum_{j \in [n]} \mu_i(\boldsymbol{x}_j). \tag{2.10}$$

For now, it is sufficient to assume that node values are always assigned as the weighted mean

$$\boldsymbol{v}_i = \frac{1}{w_i} \sum_{j \in [n]} \mu_i(\boldsymbol{x}_j) \boldsymbol{y}_j. \tag{2.11}$$

This is actually a consequence of the equivalence this impurity minimization scheme to ERM, as discussed in Section 2.3.5. For KDDTs, this gives the identity $\boldsymbol{v}_i = \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_k}[\mathbf{y} \mid \mathbf{x} \in \mathcal{R}_i]$ with $\mathcal{R}_i$ as in Equation 2.7.

Common impurity functions for classification are the Gini impurity

$$\text{Gini}(\boldsymbol{v}) = 1 - \sum_{j \in [p]} v_j^2 \tag{2.12}$$

and entropy

$$\text{Entropy}(\boldsymbol{v}) = -\sum_{j \in [p]} v_j \log v_j. \tag{2.13}$$

For regression, the typical impurity function is Mean Squared Error (MSE) impurity, which cannot be written in terms of $\boldsymbol{v}$ alone, but as formalized in this work, it is equivalent to the Gini impurity up to constants, so Gini impurity can be used for the sake of optimization.

As in CART, a tree is constructed using a greedy top-down algorithm whereby, at each node i, the feature a_i and threshold t_i are chosen to minimize the total impurity. This is usually described in terms of the maximization of the gain

$$\text{Gain}(a, t) = \text{Impurity}(\boldsymbol{v}_i) - \frac{w_{\ell_i}}{w_i}\text{Impurity}(\boldsymbol{v}_\ell) - \frac{w_{r_i}}{w_i}\text{Impurity}(\boldsymbol{v}_r) \tag{2.14}$$

where the weights and values of the child nodes depend on a and t. This proceeds until one of several possible stopping conditions is met.

In CART, for each $a \in [p]$, a set of candidate thresholds is generated by bisecting each consecutive pair of data values, then the total of each target value on either side of each candidate threshold is computed using a cumulative sum. This allows the efficient evaluation of every possible partition of the training data. However, for FDTs, where σ_i is also given, the gain may be continuous, and such an exhaustive evaluation is generally not possible. To tackle this, we make the following assumption, which is implied by the more specific Assumption 2.3 for KDDTs.

Assumption 2.4. Each splitting function $\sigma_i(\boldsymbol{x}_j; a, t)$ is continuous and piecewise-linear in t. In particular, it is linear on intervals $(-\infty, c_{j,1}), [c_{j,1}, c_{j,2}], \ldots, [c_{j,r_i}, \infty)$.

We also make an assumption about the impurity function. Entropy, Gini, and MSE all satisfy this condition.

Assumption 2.5. The Hessian of the impurity is negative semidefinite everywhere.

Then Theorem 2.1, which we prove in [77], provides a set of candidate thresholds.

Theorem 2.1. *If Assumptions 2.4 and 2.5 hold, then given a feature $a \in [p]$, a threshold $t \in \mathbb{R}$ that maximizes Gain(a, t) is in $\{c_{j,k} \mid j \in [n], k \in [r_i]\}$.*

Now that we have a set of candidate thresholds, Since the membership values μ_i are linear in t, we can iterate over the ordered candidates $(t_1, t_2, \dots)$ as in CART and keep a running sum of target values

$$\boldsymbol{s}_{\ell_i} = \sum_{j \in [n]} \mu_{\ell_i}(\boldsymbol{x}_j)\boldsymbol{y}_j \tag{2.15}$$

using the update

$$\boldsymbol{s}_{\ell_i}(a, t_{k+1}) = \boldsymbol{s}_{\ell_i}(a, t_k) + (t_{k+1} - t_k)\boldsymbol{s}'_{\ell_i}(a, t_k) \tag{2.16}$$

where the rate of change is updated

$$\boldsymbol{s}'_{\ell_i}(a, t_{k+1}) = \boldsymbol{s}'_{\ell_i}(a, t_k) + (\sigma'(a, c_{j,m}, \boldsymbol{x}_j) - \sigma'(a, c_{j,m-1}, \boldsymbol{x}_j))\mu_i(\boldsymbol{x}_j)\boldsymbol{y}_j \tag{2.17}$$

where $c_{j,m}$ is the partition bound corresponding to threshold candidate t^{k+1}. When p is out of range, the term is omitted. Then we compute gain using Equation 2.14 with the identities $\boldsymbol{v}_i = \boldsymbol{s}_i / w_i$, $w_{r_i} = w_i - w_{\ell_i}$, and $\boldsymbol{s}_{r_i} = \boldsymbol{s}_i - \boldsymbol{s}_{\ell_i}$.

Stopping Conditions

Various stopping conditions are possible. The following are straightforward to implement in a recursive fitting algorithm:

- maximum depth
- minimum sample weight allowed in any node
- Cost-Complexity Pruning (CCP) α, the minimum gain, weighted by the proportion of samples at the current node, required to create a new split
- timeout.

If splits are added to the tree in order of highest gain, for example, using a priority queue, additional stopping conditions are possible:

- maximum number of nodes
- maximum number of leaves.

In the absence of equal-gain splits, these are equivalent to CCP, but offer more transparent control of tree size. All of the above stopping conditions are supported in our published implementation of KDDTs.

Computational Complexity

The threshold search for a single node in CART with n incoming samples iterates over n thresholds for each of p features, computing impurity on q targets for each, resulting $O(npq)$. Our FDT algorithm multiplies the number of threshold candidates by the number of splitting function pieces r minus one and adds constant additional computation to each step, resulting in $O(npqr)$. In the context of KDDTs, the complexity of CART is multiplied by the number of pieces in the fitting kernel. For simple kernels like a box kernel, the additional cost is negligible. For approximated kernels, such as a histogram approximation of a Gaussian kernel, the number of bins (pieces) can be set as a tradeoff of runtime vs. fidelity.

Another factor to consider is that, in classic CART, a sample may only belong to one path, that is, one node at any depth level of the tree. This means that, for balanced trees, the complexity of fitting the entire tree is $O(npq \log|T|)$, as opposed to $O(npq|T|)$ for highly unbalanced trees. Meanwhile, in FDTs, data may belong to multiple paths, and in the worst case, all of them. This worst case gives $O(npqr|T|)$ for any tree shape. However, in reality, it is a spectrum. In the context of KDDTs, using a small bandwidth results in sparse membership; as kernel bandwidth approaches zero, the worst-case runtime approaches $O(npqr \log|T|)$ for balanced trees. Using large bandwidth results in dense membership, with worst-case runtime approaching $O(npqr|T|)$. Indeed, in practice, we see that fitting kernel bandwidth has a dramatic effect on fitting time.

Likewise, for prediction, classic CART is $O(\log|T|)$ for balanced trees and $O(|T|)$ for highly unbalanced trees, whereas FDT prediction ranges from $O(\log|T|)$ to $O(|T|)$ depending on membership sparsity for balanced trees.

2.3.4 Visualization on Toy Data

Figures 2.2 and 2.3 visualize tree-based classifiers and regressors, respectively, on synthetic 2-dimensional toy data sets, each with 50 samples, then 1000 samples. The DTs have size selected via the CCP-α parameter by 10-fold cross-validation, and the KDDTs use a box kernel with bandwidth likewise selected by cross-validation.

KDDTs tend to have smoother decision boundaries with larger margins. They also tend to grow larger when size is selected in this way, but can often be pruned without significantly affecting predictions.

2.3.5 Tree Growth as Empirical Risk Minimization

Decision tree growth is often understood to be a fundamentally different learning process from the typical ERM since it aims to maximize gain functions, such as Gini gain or information gain, that are uniquely defined for trees. However, these processes can actually be viewed through the lens of ERM with common loss functions, as

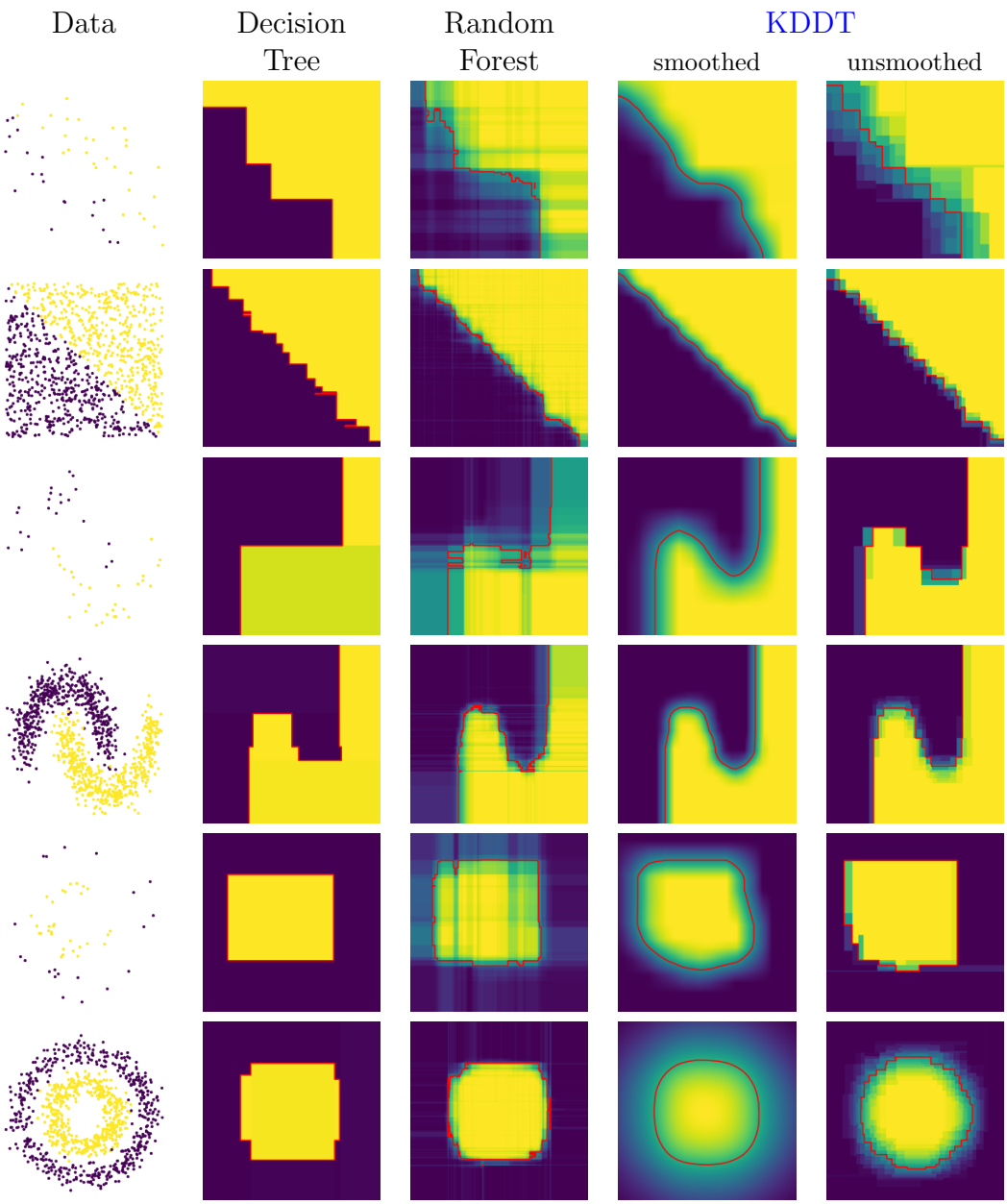


Figure 2.2: Visualization of tree-based classifiers on toy data.

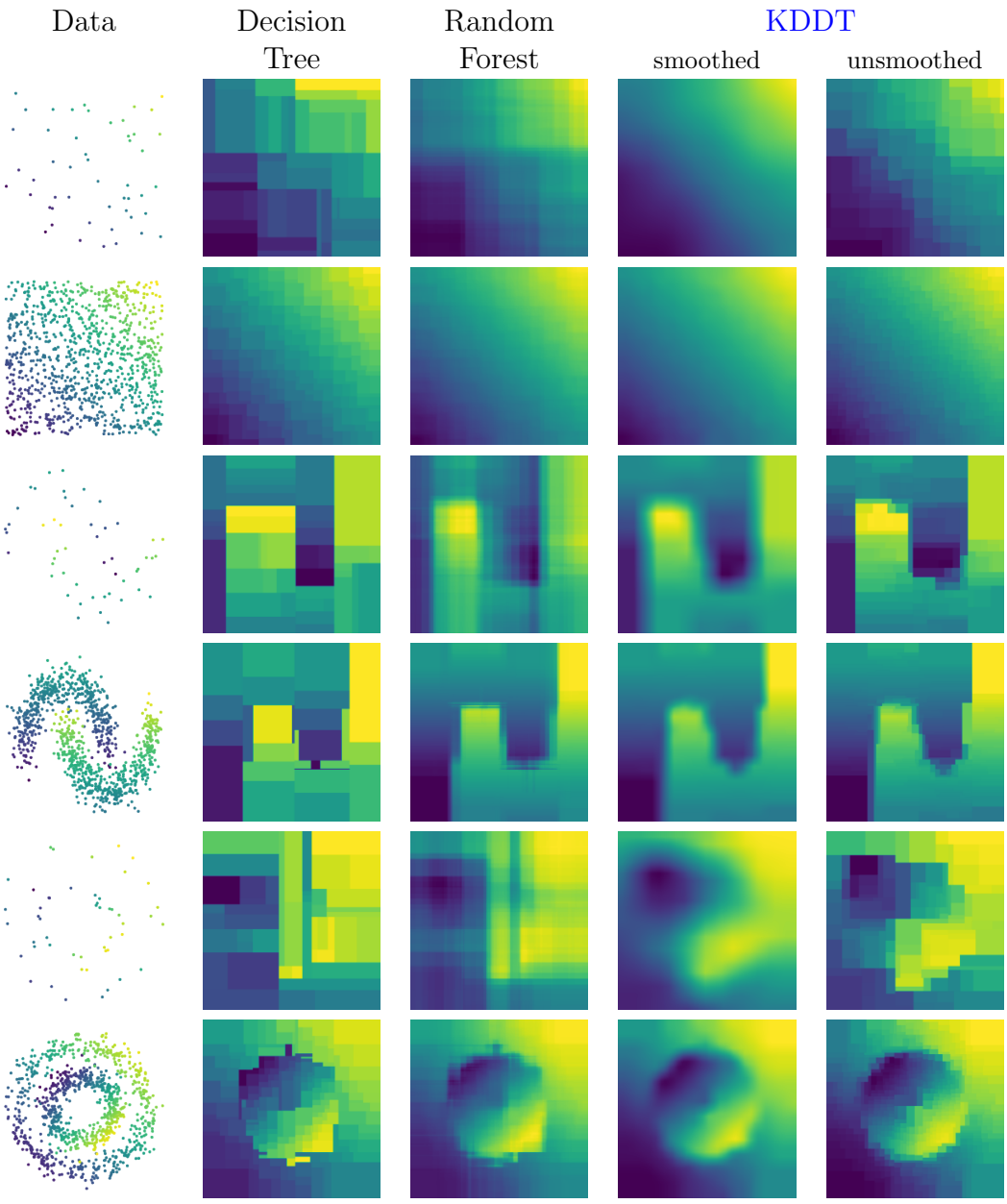


Figure 2.3: Visualization of tree-based regressors on toy data.

full name	short name	labels	features	samples
Iris	iris	3	4	150
Wine	wine	3	13	178
Glass Identification	glass	6	9	214
Optical Recognition of Handwritten Digits	optdigits	10	64	5620
Ionosphere	ionosphere	2	34	351
Pen-Based Recognition of Handwritten Digits	pendigits	10	16	10992
Image Segmentation	segmentation	7	19	210
Letter Recognition	letter	26	16	20000
Yeast	yeast	10	8	1484
Spambase	spambase	2	57	4601
Connectionist Bench (Sonar, Mines vs. Rocks)	sonar	2	60	208
Statlog (Landsat Satellite)	satimage	6	36	6435

Table 2.1: Information about data sets.

established by the following theorems. These will be used throughout this work. See Appendix B for the proofs.

Theorem 2.2. *For KDDTs, Gini impurity is equivalent to Mean Squared Error (MSE) risk on the smoothed empirical distribution $\hat{p}_k$.*

Theorem 2.3. *For KDDTs, Entropy impurity is equivalent to cross-entropy risk on the smoothed empirical distribution $\hat{p}_k$.*

Since conventional DTs are a subclass of KDDTs, we get the following corollaries by letting $k(\boldsymbol{z}, \boldsymbol{x}) = \delta(\boldsymbol{z} - \boldsymbol{x})$.

Corollary 2.1. *For DTs, Gini impurity is equivalent to MSE risk on the empirical distribution $\hat{p}_{\mathcal{X},\mathcal{Y}}$.*

Corollary 2.2. *For DTs, Entropy impurity is equivalent to cross-entropy risk on the empirical distribution $\hat{p}_{\mathcal{X},\mathcal{Y}}$.*

From these, we conclude that tree fitting to maximize Gini gain and Information gain minimizes empirical MSE and cross-entropy risk, respectively.

Furthermore, through this framing of tree fitting as risk minimization, it is straightforward to show that the risk-minimizing leaf values are as in Equation 2.11 for both cross-entropy and MSE. However, this it may be different for other loss functions.

2.3.6 Benchmarks

We compare KDDTs to conventional decision trees in the form of basic Decision Trees (DTs), Random Forests (RFs), and ExtraTrees (ET). We include Extreme Gradient Boosting (XGB) [35] as a representative of boosted ensemble methods, but

	decision tree			random forest			ExtraTrees			boost
model	DT	KDDT		DT	KDDT		DT	KDDT		XGB
smoothed	-	yes	no	-	yes	no	-	yes	no	-
iris	94.0	96.7	**97.3**	94.0	94.7	**95.3**	95.3	95.3	**96.0**	94.0
wine	88.2	**97.7**	94.3	97.7	**98.9**	**98.9**	**98.3**	**98.3**	**98.3**	96.0
glass	71.0	**73.3**	71.0	78.6	**80.9**	77.6	**77.1**	76.7	76.3	78.1
optdigits	90.8	**97.7**	94.9	98.3	**98.6**	**98.6**	98.5	**98.6**	**98.6**	97.8
ionosphere	87.8	92.3	**93.2**	93.7	**94.9**	**94.9**	**94.3**	**94.3**	**94.3**	93.2
pendigits	96.3	**98.9**	98.0	99.1	**99.3**	**99.3**	**99.4**	**99.4**	99.3	99.1
segmentation	88.6	**89.5**	89.0	91.9	**92.9**	92.4	91.4	**92.9**	91.9	87.6
letter	88.1	**94.5**	88.1	**96.8**	96.6	96.6	97.4	**97.5**	97.4	96.5
yeast	58.6	**61.3**	60.2	61.9	61.3	**62.3**	60.9	**61.5**	60.5	59.4
spambase	92.1	**93.5**	92.2	95.4	**95.6**	95.3	**96.0**	95.6	95.5	95.6
sonar	72.0	**83.1**	78.3	83.6	86.0	**89.4**	88.0	**89.4**	**89.4**	84.6
satimage	87.4	**90.6**	88.7	**92.0**	91.7	91.7	**91.8**	91.7	91.6	92.1

Table 2.2: 10-fold cross-validation accuracy for tree-based models. Best in category is bold. Best overall is underlined.

do not implement boosted KDDTs. We also include as baselines Linear (or Logistic) Regression (LR) and a small Multi-Layer Perceptron (MLP). The data are the most popular tabular classficiation data sets with continuous features from the UCI Machine learning Repository [49] at the time of writing. Data sets are summarized in Table 2.1. We report accuracy from 10-fold cross validation, and hyperparameters including kernel bandwidth and CCP-α are selected for each fold by an additional 10-fold cross validation. We also set a minimum sample weight of 1 for all models. The KDDT-based models use a simple box kernel, that is, a uniform distribution on a hypercube centered at the input. Additional details are available in Appendix C.2.

The results are shown in Table 2.2. They show that KDDTs are a powerful enhancement of conventional trees, usually providing a significant boost to single-tree performance or a modest boost to random forest performance. There is less evidence of benefit to ExtraTrees models, where the altered choice of threshold value is mostly lost due to random threshold selection. KDDTs outperform decision trees on every data set and come close to or exceed the conventional tree ensembles on several, a notable feat for single trees with feature-aligned splits. With random forests, KDDTs outperform standard trees on 10 data sets. With ExtraTrees, they outperform on 6 and tie on 3. A model using KDDTs outperforms all baseline models on 9 data sets and ties on 1. Usually, KDDT performance is better with smoothed predictions than without.

2.3.7 Discussion

KDDTs offer a simple and efficient way to improve the generalization of decision trees by interpreting inputs as uncertain, or viewed differently, by regularizing the predictive function they represent through smoothing. This significantly boosts the performance of single-tree models, sometimes coming close to or matching ensemble performance, and can slightly increases the performance of tree-ensembles.

The regularizing effect also improves various aspects of the robustness of trees, as we explore in Chapter 3. Moreover, the use of prediction kernels augments KDDTs with the additional utilities of smooth prediction and differentiability, which we apply for gradient-based learning in Chapter 4 and semi-supervised learning in Section 5.1. Finally, the KDDT fitting algorithm is generally useful for fitting trees to any kind of piecewise-constant distribution, not just smoothed empirical risk. We explore its application toward decentralized federated learning and ensemble merging in Sections 5.2 and 5.3, respectively.

The main limitations are the following. First, there are limitations on the kernel choice. For both fitting and prediction, we require that the kernel is isotropic, that is, that it has diagonal covariance, and for fitting, we require the marginal distributions to be piecewise-constant. Second, the possibility for samples to take multiple paths increases computational cost; if the kernel support is large, the data membership in tree nodes becomes dense and the computation cost becomes very high. Third, KDDTs introduce a rather complex design choice in the shape and size of the kernel. We find that, in the general case, the best strategy is usually to choose a simple, symmetrical kernel such as a box kernel or Gaussian kerenl and select bandwidth from a set of candidate values, usually in the range of .01 to 1 for normalized data, by cross-validation. Of course, tuning the bandwidth by cross-validation also increases the cost of applying the models; however, since tree ensembles and cross-validation are easily parallelizable, and since trees have a small memory footprint and do not require any special hardware such as a GPU, KDDTs still remain practical to apply.

2.4 Other Models

We briefly discuss the potential for application of this uncertain interpretation formalism to a number of other model classes besides trees.

2.4.1 Partitioning Models

The key to applying the formalism to trees is that trees partition the domain and have constant value on each subdomain, so the smoothing reduces to integrating the kernel on each subdomain. Under certain assumptions on the model structure and kernel, the integral is efficient to compute. For trees, we rely on axis-aligned splits, that is,

hyperrectangular partitioning, and isotropic kernels. Other partitioning models are also promising candidates for efficient application.

There are various generalizations of decision trees that use different decision rules from the typical feature-threshold rules. We discuss some of these in Section 4.1. A popular example is Oblique Decision Trees (ODTs), where the decision rules compare a linear combination of features to a threshold. Then the resulting subdomains are convex polytopes. Even if the kernel is uniform on a convex polytope such that the integral reduces to computing the measure of a convex polytope, it is hard to compute [51], but fast approximations have been developed [66, 53, 30]. This does not guarantee, however, that they can be used effectively for fitting ODTs, for which there are several methods.

There are various algorithms for learning logical rule-based models, of which trees are just one example. Like trees, they are often valued for their intuitive logic-based decision making, and like trees, they are good candidates for efficient application of our uncertain interpretation formalism, particularly with simple feature-threshold rules. Though they are among the oldest kinds of learned models, innovation on rule-based models and learning algorithms continues to this day; recent examples include [195, 140]. Some, such as certain types of rule lists, are special cases of trees or tree ensembles. [124] recently survey explainability and interpretability in machine learning and include a section on rule-based models. [117] broadly study the intersection of rule-based systems and machine learning. [59] survey genetic algorithms for learning rule-based models, including trees.

A particularly interesting class of models are those based on Directed Acyclic Graphs (DAGs), of which trees are a special case. DAGs allow a node to have multiple parents, meaning that they can be much more compact and computationally efficient than trees by consolidating redundant structures. As an extreme example, consider the mapping $\boldsymbol{x} \mapsto \mathbf{1}\{\prod_{i \in [p]} x_i \geq 0\}$. While a decision tree takes $O(2^p)$ nodes to represent this mapping, a DAG with similar feature-threshold rules can represent it in just $O(p)$ nodes. Meanwhile, the node membership computation strategy we use for smoothing trees can be applied efficiently to more general DAGs with very little modification. Much of the literature in DAG is toward building Bayesian networks. However, these model discrete data, whereas the focus of this work is continuous data. Though we are not aware of any, if there exists work viewing DAGs as an alternative to decision tree learning, particularly learning algorithms based on a greedy heuristic similar to CART for learning trees, then it is possible that our KDDT fitting algorithm could be generalized for DAGs.

2.4.2 Linear Models

We begin with linear regression, which has the form $h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ for some $\boldsymbol{A} \in \mathbb{R}q \times p, \boldsymbol{b} \in \mathbb{R}^q$. The usual loss function is mean squared error. Consider the risk for one output component $i \in [q]$ for training sample $\boldsymbol{x}, \boldsymbol{y}$. If we reasonably assume that

$\mathbb{E}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[\mathbf{x}] = \boldsymbol{x}$, then the risk is

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[(\boldsymbol{A}_i^\top \mathbf{x} + b_i - y_i)^2] &= \mathbb{E}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[\boldsymbol{A}_i^\top \mathbf{x} + b_i - y_i]^2 + \mathrm{Var}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[\boldsymbol{A}_i^\top \mathbf{x} + b_i - y_i] \\ &= (\boldsymbol{A}_i^\top \boldsymbol{x} + b_i - y_i)^2 + \boldsymbol{A}_i^\top \Sigma \boldsymbol{A}_i\end{aligned}$$

where Σ is the covariance matrix $\Sigma_{i,j} = \mathrm{Cov}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}(\mathbf{x}_i, \mathbf{x}_j)$. This is just squared error with regularization on the coefficients $\boldsymbol{A}_i$; in particular, if $\Sigma = \lambda I$, then it is a standard L2 regularization with coefficient λ. This reflects the notion that our formalism is a kind of regularization. At inference time, $f_i(\boldsymbol{x}) = \mathbb{E}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x})}[\boldsymbol{A}_i^\top \mathbf{x} + b_i] = \boldsymbol{A}_i^\top \boldsymbol{x} + b_i$, so the smoothing has no effect.

For logistic regression, the use of the softmax function to map the linear function into $[0,1]^q$ for classification makes computation of the expectation over the kernel difficult.

2.4.3 Kernel Trick Models

The "kernel trick" writes a model in terms of inner products with the training data, then replaces the inner products with a kernel function that represents an inner product in a higher or even infinite dimensional transformation of the data. The best known example is the kernel Support Vector Machine (SVM), which is used for binary classification only with output dimension $q = 1$; differently from our usual notation, this means that labels y_i are scalars -1 or 1 instead of the usual indicator vector. In particular, kernel SVM has the form $h(\boldsymbol{x}) = \sum_i a_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b$ for some learned parameters $\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$ and chosen kernel k. While typically soft-margin SVM is interpreted as a margin maximization problem, it can also be viewed through the lens of ERM with hinge loss $\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$. Suppose we apply our formalism with another kernel k'. Then we have

$$\hat{R}_{k'}[h] = \frac{1}{n} \sum_{i\in[n]} \int_{\mathcal{X}} \max\left(0, 1 - \sum_{j\in[n]} a_j y_i y_j k(\boldsymbol{z}, \boldsymbol{x}_j)\right) k'(\boldsymbol{z}, \boldsymbol{x}_i)\, d\boldsymbol{z}$$

and there is no obvious efficient and precise way to compute this. This highlights the difference between the kernel trick and our formalism despite some surface-level similarity.

2.4.4 Black-box Models

For model classes where there is no efficient deterministic approach for applying our uncertain interpretation formalism, Monte Carlo methods can be applied. The most basic approach is to sample points from $k(\cdot, \boldsymbol{x})$ to compute unbiased estimates of the expected risk and smoothed predictions during training and inference, respectively. For instance, during minibatch optimization of a neural network, one can replace

each instance $\boldsymbol{x}$ in a batch with a random sample from $k(\cdot, \boldsymbol{x})$, and when making a prediction at an input $\boldsymbol{x}$, the output can be averaged over many samples from $k(\cdot, \boldsymbol{x})$. This is ultimately a kind of data augmentation. While Monte Carlo methods are simple and general, they are of course approximate, and there is a tradeoff of sample size (and therefore computational cost) vs. variance of the approximation.

Another approach is multidimensional numerical integration, for which Monte Carlo methods are one family of methods that do not suffer so much as others from a curse of dimensionality, which is important in this context. Viewed through the lens of numerical integration, there may be methods, Monte Carlo or otherwise, that better take advantage of the characteristics of a particular model class and kernel.

2.5 Choosing Kernels

The choice of the size and shape of the kernel is important and nontrivial. Even if we fix the shape and choose just the size, too small results in little or no difference from the default, unsmoothed model, and too large over-smooths the model and causes it to underfit. In reality, one must also choose the shape of the kernel, including the possibility for different scaling along different features.

In short, the best approach we have found that requires no input from the user is to simply choose a simple, general kernel shape (we use box kernels because they are efficient with KDDTs), then select the bandwidth from a candidate list using cross-validation. This requires that either the data is normalized or the bandwidth is scaled by some measure of variation for each feature. Usually, for data with standard deviation 1, a good range of bandwidths is on the order of 0.01 to 1.0.

This approach, despite being the best balance of practicality and efficacy, has some important limitations. First, cross-validation increases the cost of model design, and for very large models and/or data, or when fitting very many models as in experimental settings, this can become a practical challenge. Second, though it may be reasonable to choose different bandwidths for different features, especially for e.g. tabular data where different features may have completely different properties, it is not practical to choose different bandwidths for different features using this exhaustive cross-validation approach because the number of combinations increases exponentially with the number of features. This motivates exploration into other strategies for kernel selection. We will overview the approaches we have considered so far, as well as the benefits and limitations of each.

First, there is a huge amount of research into kernel selection for KDE that we may leverage to kernels in a model-agnostic fashion. Some examples include the following.

- Rules of thumb such as Silverman's [156] for bandwidth selection. These are an extremely low-cost option that may give a starting point for cross-validation, but rarely achieve the best performance on their own.

- Likelihood maximization methods, that is, choosing a kernel that maximizes cross-validation or leave-one-out likelihood of the KDE. While this still involves cross-validation, it can be significantly faster in practice than cross-validation involving fitting a model.

- Adaptive kernel methods, that is, methods where properties of the kernel, such as the bandwidth, are chosen locally. See for example [150] for an in-depth study.

Some of these scale unfavorably with the size of the data and may not be practical for large data sets. But the main problem with these methods is that they ignore the model completely, which can be harmful. For instance, with KDDTs, the tree acts as a kind of automatic local dimension reduction. This is beneficial; it eliminates the curse of dimensionality issue present in purely kernel-based models, helps to generalize out of distribution, and makes even small kernels useful. However, it means that bandwidths chosen in a pure KDE context, where there is no dimension reduction, become increasingly too large. Indeed, we find that likelihood maximization and nearest-neighbor based adaptive methods work exceedingly well in low-dimensional data, but for even moderately many features, the selected bandwidth is always too large and the tree badly underfits. While trees may be an extreme example, we expect to see some degree of this issue in any model class. This can be partially addressed by scaling heuristics, for instance, dividing the bandwidth by the square root of the number of features, but this is generally unreliable. The main takeaway is that an ideal kernel selection method must consider the model.

There is one model-agnostic method that, while not ideal, is sometimes useful because it does not suffer from the issue of selecting overly large bandwidth as the dimensionality increases: selecting the bandwidth independently for each feature, for instance, using a likelihood maximizing method. In particular, when using a kernel with piecewise-constant marginal distributions, as required by KDDT fitting, it is possible to compute the leave-one-out likelihood for a sample in linear time; the density at each point can be computed by a running sum over the kernel pieces as in the KDDT threshold search. This can select a bandwidth much faster than cross-validation while also enabling different bandwidth for each feature, sometimes even outperforming the cross-validation approach while generally being less reliable. The main limitation here, besides that it is still model-agnostic, is that it treats the features as completely independent, which is usually not a reasonable assumption.

It is not possible to learn the kernel size and shape using training loss; just as increasing a regularization coefficient does not decrease loss, it is the same for kernel size. For instance, it is straightforward to see that, in KDDTs, increasing the kernel bandwidth never decreases the impurity. As a result, any loss-minimizing approach will just shrink the kernel toward size zero.

One last approach is to leverage expert knowledge to guide the kernel selection process. There are often known sources of noise in data, such as quantization, sensor

error, or inherent randomness of the observed process. In some cases, the noise model may be understood; in the best case, the exact range or distribution of uncertainty may be known. In this case, the kernels can be designed to model these specific sources of noise to make the model aware of them during training and prediction. In Section 3.2.1, we add noise to data and study its effect on the kernel bandwidth that maximizes performance; usually, the optimal kernel bandwidth increases linearly with the amount of noise, suggesting that knowledge of the noise distribution can indeed inform kernel choice.

Chapter 3

Robustness and Verification

Robustness refers generally to the ability to remain consistent in the presence of adverse conditions. In machine learning, it is core to the trustworthiness of a model to perform reliably in real-world deployment. There are various adverse conditions to which we expect a good model to be robust, depending on the application context. Examples include missing values, noisy features or labels, adversarial perturbation of features, distribution shift, class or group imbalance, data poisoning, and confounding patterns in the data. Several recent works [61, 184, 24] broadly discuss the various notions of robustness in ML and their importance for models to be reliable, safe, predictable, and generally trustworthy.

In this chapter, we examine the effect of using KDDTs on the robustness of decision trees to a number of adverse conditions. We begin in Section 3.1 by showing that KDDTs reduce the well-known sensitivity of trees to small, random changes in the data. Next, in Sections 3.2 and 3.3, we show how KDDTs improve robustness to noisy features and noisy labels, respectively. In Section 3.4, we study the robustness of KDDTs and their ensembles to adversarial perturbation. While adversarial perturbations are not actually present in most applications, it is a useful framework for analyzing the robustness of models when the natural perturbation distribution is unknown. Since these kinds of data issues are common, these help to better understand why KDDTs can so greatly outperform decision trees and better inform their application.

Finally, in Section 3.5, we discuss the related topic of verification, that is, the process of proving or finding counterexamples to safety properties of models, of which the most popular property among researchers is adversarial robustness. We propose the first verification algorithm for FDTs that is sound and complete in finite precision and show that, despite the NP-Completeness of the verification task, it scales to reasonably sized models in practice.

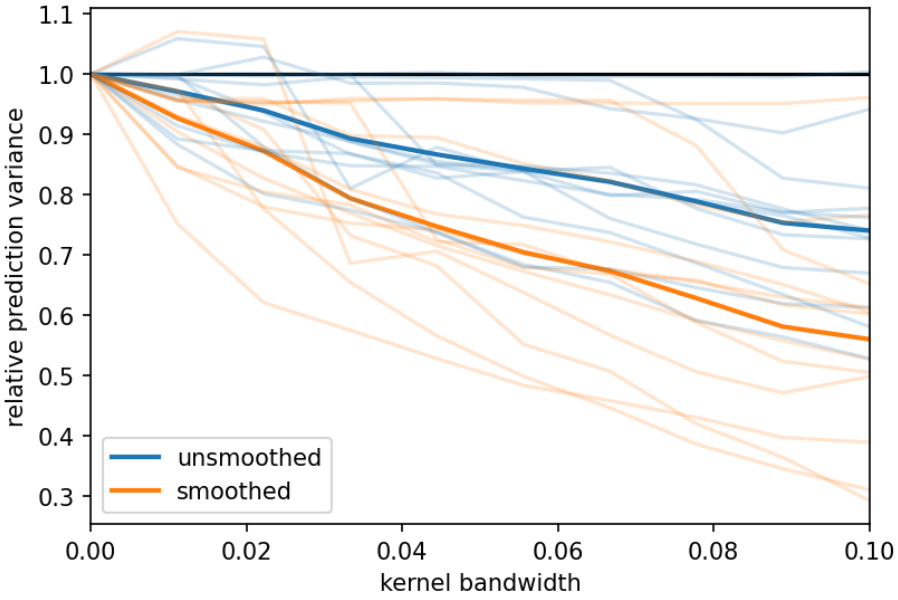


Figure 3.1: Test set prediction variance of KDDTs trained on 100 bootstrap samples, relative to that of DTs, for 12 data sets. Average over data sets in high opacity.

3.1 Decision Tree Stability

One of the main drawbacks of decision trees is that they are unstable or sensitive to small changes in the training data [113, 144, 89]. Since the growth algorithm is greedy, and since each subtree depends on its ancestor splits, a change in split can result in an entirely different subtree. This is one interpretation of why bagging is so effective with trees; by averaging over randomized trees, it reduces the variance to zero as the ensemble grows large.

One of the known effects of regularization is to reduce the variance of a learning algorithm at the cost of increased bias, which motivates the question of to what extent our uncertain interpretation formalism, which we view as a kind of regularization, has this effect on trees. To test this hypothesis, we split each of the 12 data sets summarized in Table 2.1 into 50% training and 50% test data, then fit DTs and KDDTs with box kernels of various bandwidth to 100 bootstrap samples of each training set, as if fitting a random forest but without feature randomization. Then we measure the variance of the predicted scores over the 100 trees, averaged over the samples in the test set. In Figure 3.1, we show the evolution of the average prediction variance of the KDDTs relative to that of the DTs. As the bandwidth increases, the variance of the predictions reduces, especially when using smoothing during prediction. This is evidence that our uncertain interpretation formalism is a kind of regularization, and that KDDTs are effectively regularized decision trees with improved robustness to the inherent randomness of sampled data.

3.2 Feature Noise

For any data set, various sources of noise that are not informative of the target value, whether it be a class label or regression target, may affect the observed feature values. These can range from sensor error to precision issues to the inherent randomness of the process under observation. In some cases, the noise model may be understood; for instance, sensors may have known noise distributions and failure modes, and random physical processes, such as radiation, may be well-understood. In other cases, sources of error may be unknown. The relationship between feature noise and performance is well-studied with numerous works showing a steady decrease in performance as the noise severity increases. See [84] for a survey of such works.

Trees in particular can be vulnerable to feature noise. When fully grown, they are highly vulnerable to overfitting. In addition, they are known to be sensitive to small changes in the data, as demonstrated in Section 3.1; even a small change, especially if it affects the choice of split early in the tree, can have a butterfly effect on the rest of tree that depends on it. Since KDDTs interpret data features with uncertainty, we expect them to be more robust to feature noise if the size and shape of the kernel accurately reflect the noise present in the data. This can also be seen more generally as a regularizing effect that reduces overfitting to noisy data. In this section, we test these hypotheses by evaluating KDDTs trained on data with random noise added to the features.

3.2.1 Experiments

We study the effect of KDDTs on the robustness of decision trees to feature noise. The data is described in Table 2.1. For each data set, we apply additive uniform or Gaussian noise to features, and for the KDDTs, we test with box and Gaussian kernels. We test combinations of noise scaling and kernel bandwidth with both ranging from 0 to 1 in increments of 0.1. When the kernel and noise distributions match and the noise scaling is equal to the bandwidth, then the noise distribution is identical to the kernel. Additional details are in Appendix C.3. For each trial, we measure accuracy on a 20% test split which is also affected by the same additive feature noise.

Figures 3.2 and 3.3 show the results for Uniform and Gaussian noise, respectively. Figures 3.4 and 3.5 show a breakdown of performance by bandwidth for each noise level, relative to the best performance of the respective noise level, for smoothed KDDTs. Additional results of this type are in Appendix D.1.

Performance

In figures 3.2 and 3.3 we see that, across all data sets and at all noise levels, we see that KDDTs outperform conventional decision trees, and that smoothed KDDTs outperform unsmoothed KDDTs. On most data sets, we also see that the gap widens as the magnitude of feature noise increases.

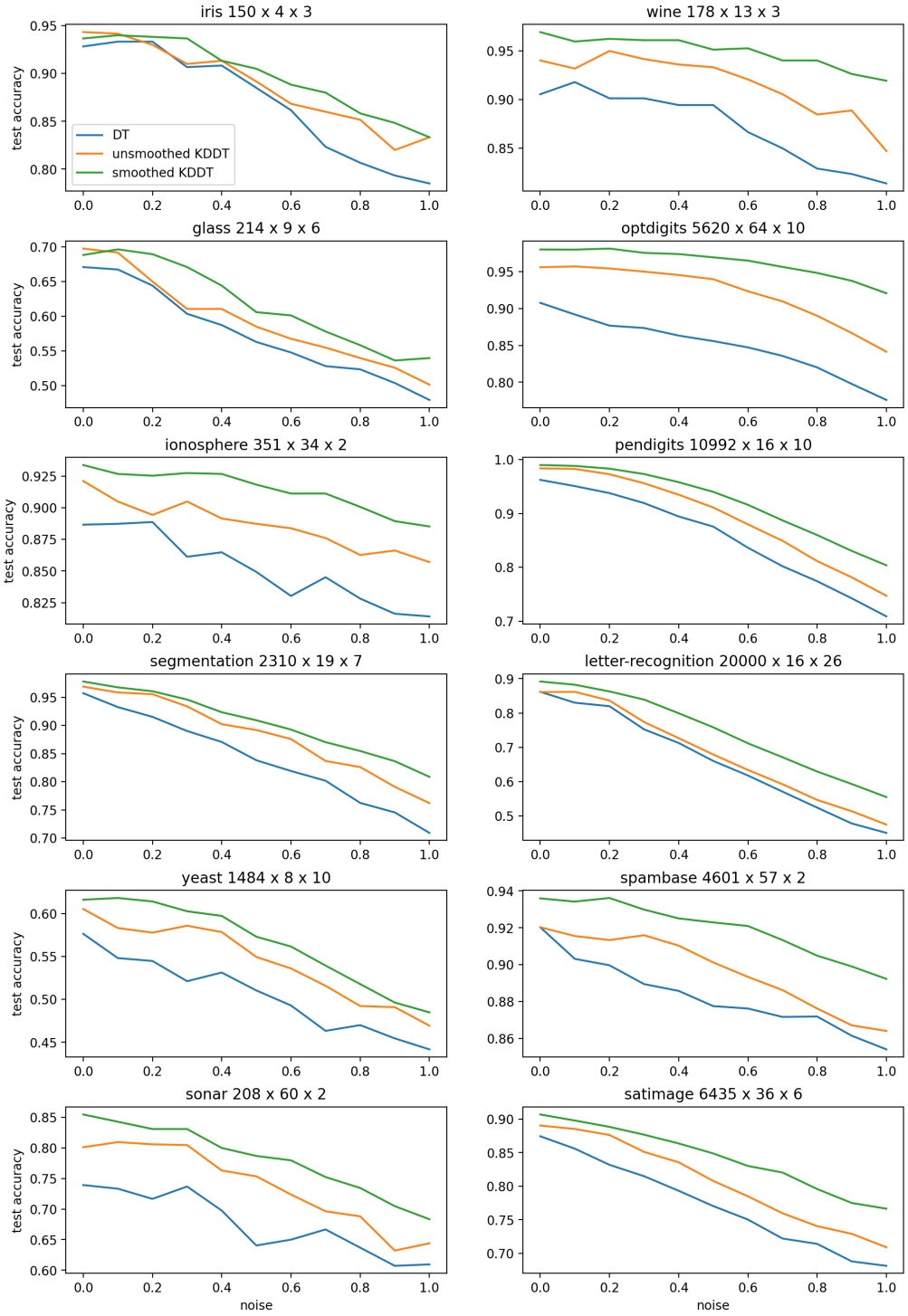


Figure 3.2: Performance of tree-based models on data where the features are each normalized to standard deviation 1, then random noise is added from a uniform distribution with radius shown on the x-axis.

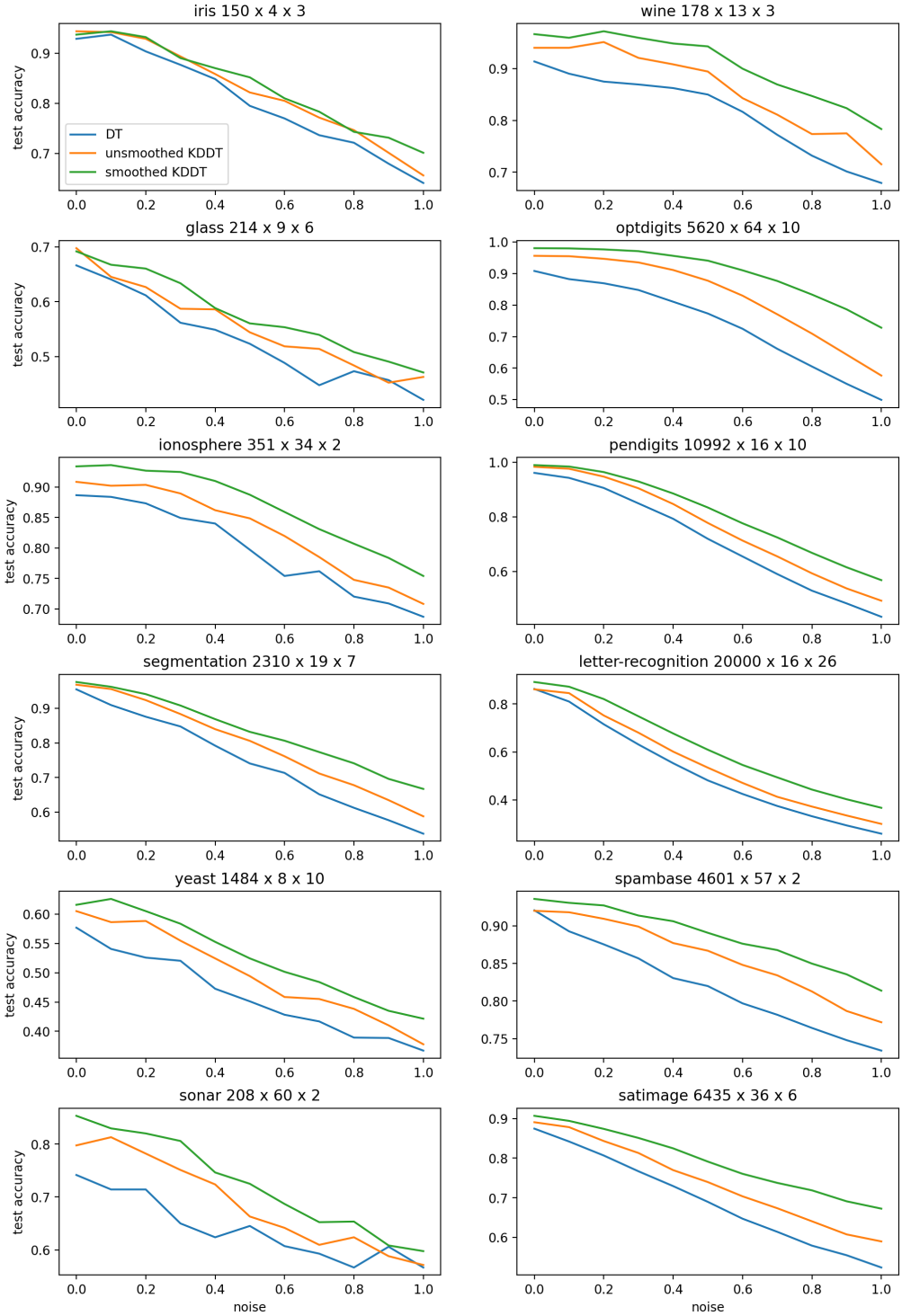


Figure 3.3: Performance of tree-based models on data where the features are each normalized to standard deviation 1, then random noise is added from a Gaussian distribution with standard deviation shown on the x-axis.

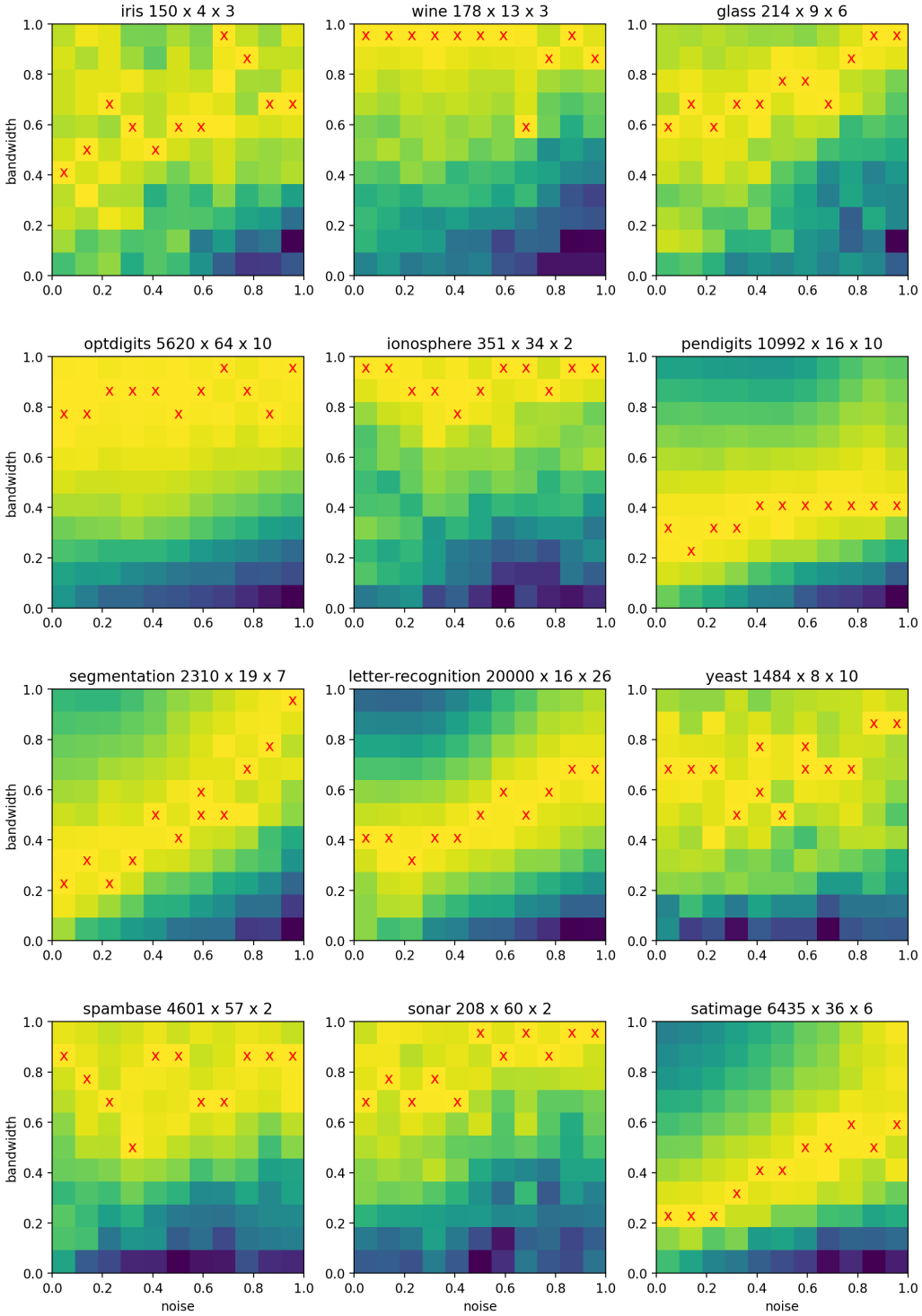


Figure 3.4: Test accuracy relative to best of each noise level (marked with x) for KDDTs trained on the data with added uniform noise.

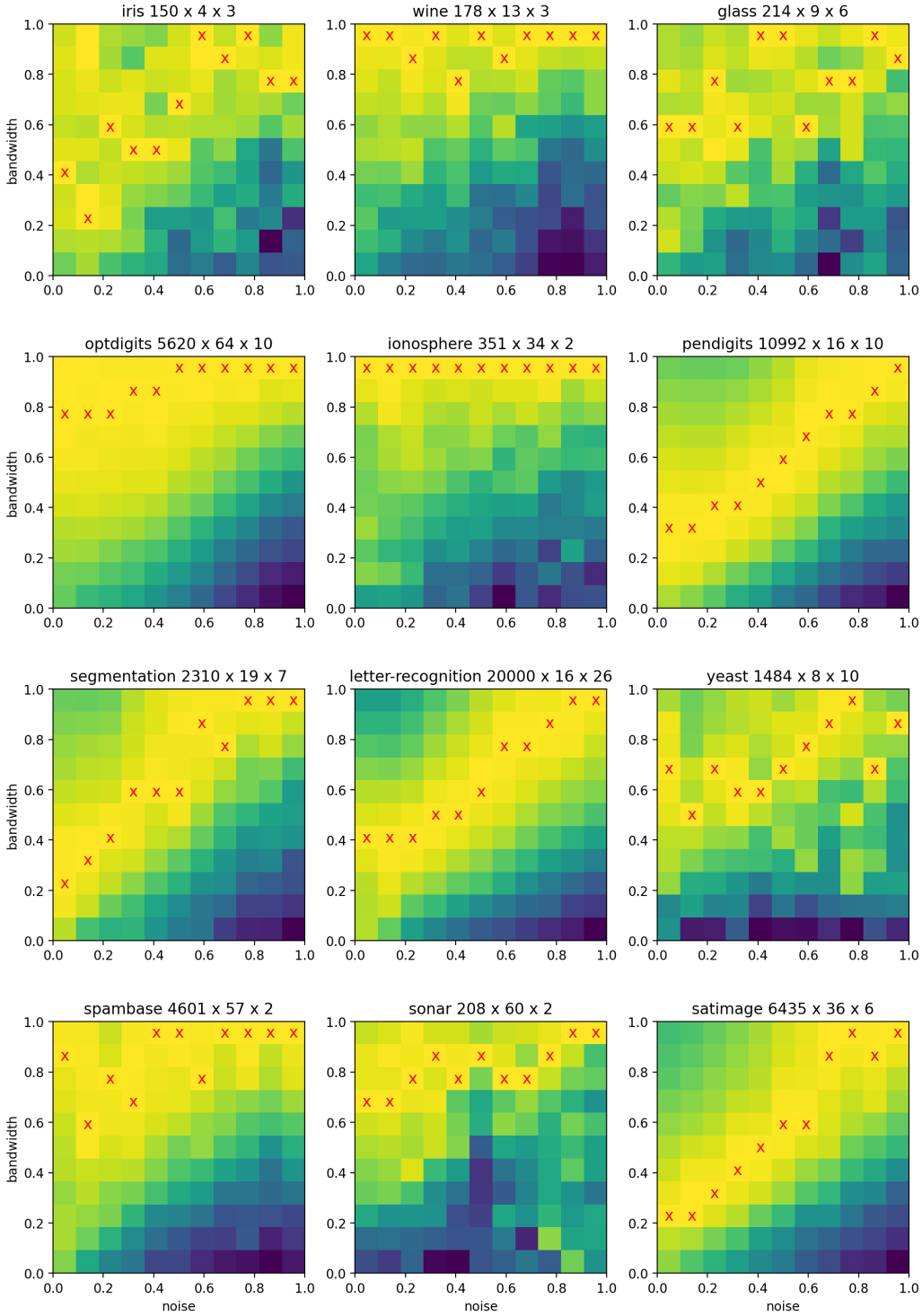


Figure 3.5: Test accuracy relative to best of noise level (marked with x) for KDDTs trained on the data with added Gaussian noise.

Kernel Size

In figures 3.4 and 3.5, we see that performance is highly dependent on kernel bandwidth and deviation from the optimal bandwidth steadily reduces performance. Sensitivity to choice of bandwidth is generally greater with more noise. In most cases, the more noise present in the data, the higher the bandwidth that achieves the best performance, and the relationship between noise level and best bandwidth is approximately linear. In the other cases, the optimal bandwidth does not depend much on noise; there are no cases where the optimal bandwidth clearly gets smaller with more noise. This is consistent with the interpretation of KDDTs as accounting for uncertainty in the input. This suggests that there is merit in the idea of setting bandwidths based on knowledge of noise in the data; for instance, one might set a very large bandwidth for a feature that is known to be dominated by noise and a small bandwidth for a feature that is known to be precise.

Kernel Shape

The KDDT results shown in this section use box kernels. We also ran a smaller number of trials with Gaussian kernels for comparison and found near-identical performance across the board, regardless of whether the added noise was uniform or Gaussian. This suggests that, while the kernel size may be important, the shape is not important for these kinds of noise.

3.3 Label Noise

In classification, labels are often provided by humans via expert annotation or crowdsourcing, and while most ML methods assume that labels are clean and correct, errors are actually common. Recently it has been estimated that public research data sets contain an average of 3% and up to 10% label errors [134]. Moreover, such label errors are generally more harmful to a trained model than feature noise [205], motivating the study of techniques for detecting and improving robustness to label errors.

While neural network robustness to label errors has received a great deal of attention [88, 160], work on decision trees remains limited. Some show trees' innate theoretical resilience to certain kinds of label errors in the sample limit, for example, [71]; however, such work does not provide actionable methods for improving robustness in real applications. Others focus on ensemble methods such as random forests [193, 203] or gradient boosting [127], which are not applicable to single trees.

In this section, we provide some no-go theory that shows that popular loss design techniques used in deep learning for robustness to label noise are not applicable to decision trees. This motivates the use of methods specialized for decision trees. We show that the regularizing effect of KDDTs improves robustness to certain kinds of label noise. We also adapt and enhance a specialized noise-robust impurity criterion

from prior work.

3.3.1 Types of Noisy Labels

For each random sample $\boldsymbol{x}, \boldsymbol{y}$, we assume that the observed label $\tilde{\boldsymbol{y}}$ may differ from $\boldsymbol{y}$ due to noise. Label noise is categorized by the nature of the conditional probability $\mathbb{P}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{y})$. We described types of label noise as follows. If $\mathbb{P}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{y}) = \eta$ for all $\tilde{\boldsymbol{y}} \neq \boldsymbol{y}$, then it is called *uniform* label noise. More generally, if $\mathbb{P}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{y}) = T_{\boldsymbol{y},\tilde{\boldsymbol{y}}}$ for some $\boldsymbol{T} \in [0,1]^{p \times p}$, then it is called *class-dependent* label noise with transition matrix $\boldsymbol{T}$. If $\mathbb{P}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{y})$ depends on $\boldsymbol{x}$, then it is called *feature-dependent* label noise. While a variety of successful methods improve robustness to uniform and class-dependent noise, feature-dependent noise is more challenging.

3.3.2 No-go Results for Loss Design Methods with Trees

A popular method for improving robustness to label noise in parametric machine learning is to modify the loss function to achieve theoretical noise tolerance. While it is not usually considered possible to apply such methods to non-parametric models like trees, the equivalence of tree fitting to ERM, as we show in Section 2.3.5, provides a framework by which to plug in loss design methods to decision trees. However, we find that two popular methods, when analyzed this way, turn out not to be useful for trees. As in Section 2.3.5, we prove these results for KDDTs, but since DTs are a subclass of KDDTs, the conclusions apply to conventional trees as well. Proofs are in Appendix B.

Symmetric Losses

Symmetric loss functions are known to be noise-tolerant to various types of label noise under mild assumptions and improve robustness to noise labels in practice [123, 72, 70, 33]. Adopting the definition from [70] into our notation, a loss function ℓ is symmetric if there exists some constant c such that, for any prediction $\hat{\boldsymbol{y}} \in [0,1]^q$, $\sum_{i \in [q]} \ell(\hat{\boldsymbol{y}}, \boldsymbol{e}_i) = c$. Recall that q is the number of classes and $\boldsymbol{e}_i$ is the indicator of class i, which is how we represent class labels consistently with regression targets in this work. Put differently, a symmetric loss is one where the sum of loss values over every possible ground-truth label is constant over all possible predictions. Examples include 0-1 loss and Mean Absolute Error (MAE) loss.

Theorem 3.1 shows that, when using any symmetric loss to fit a decision tree, the plurality indicator, that is, a one-hot vector with 1 in the position of the most frequent label, is always an optimal leaf value. This frequently results in cases where, despite high impurity, both children of every possible split have the same leaf value, resulting in zero gain and terminating tree growth early. This is why, for instance, we do not use accuracy as an objective for tree growth. Consequently, symmetric loss functions are not suitable for tree growth.

Theorem 3.1. *For any decision tree and symmetric, non-negative loss function, there exist loss-minimizing leaf values that are plurality indicators.*

Loss Correction

Loss correction [137] is a method for class-dependent noise robustness that assumes knowledge of the transition matrix $\boldsymbol{T}$ and incorporates it into the loss so that, in expectation, the risk minimizer is the same as if training on clean data. In particular, *forward correction* uses corrected loss $\ell_{\boldsymbol{T}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \ell(\boldsymbol{T}\hat{\boldsymbol{y}}, \boldsymbol{y})$. Theorem 3.2 shows that this has no effect on the structure of a learned tree. Therefore, one may simply learn a tree with the loss ℓ, then correct the leaf values. However, if the correct label is observed most frequently, a reasonable assumption, then this is unlikely to change the plurality label at many leaves. This reflects trees' inherent tolerance to certain kinds of noise. As in [137], we assume $\boldsymbol{T}$ is invertible.

Theorem 3.2. *The learned structure of a decision tree is invariant to forward loss correction.*

There is also a related method called *backward correction*. Unlike forward correction, this can result in a different tree structure; however, we show empirically in [166] that this has no influence on the performance of trees.

3.3.3 Credal Impurity for Fuzzy Trees

Credal decision trees [1, 122] augment the C4.5 tree fitting algorithm by incorporating imprecise probabilities into the splitting criterion. This method is shown by [122] to improve robustness to noisy labels. Here we generalize the method for FDTs and the CART fitting algorithm.

Credal sets are convex sets of categorical distributions estimated from data based on Walley's Imprecise Dirichlet Model [177]. Let $\boldsymbol{n} = (n_1, \ldots, n_q)$ be the number of samples in each class, where q is the total number of classes, and $N = \sum_i n_i$ the total number of samples. Then the credal set is

$$C(\boldsymbol{n}) = \left\{ \boldsymbol{p} \in [0,1]^q \;\middle|\; p_i \in \left[\frac{n_i}{N+s}, \frac{n_i+s}{N+s} \right] \;\; \forall i \in [q], \sum_{i \in [q]} p_i = 1 \right\} \tag{3.1}$$

where $s \in \mathbb{R}_{\geq 0}$ is a hyperparameter with larger s leading to more conservative inference. One can think of a credal set as the set of distributions one could achieve by adding s additional labeled samples to a population. This has the property that larger sample sizes yield a smaller, more confident range of probabilities.

Credal decision trees simply replace the impurity function with its maximum over the credal set formed by the samples at a given node. In particular, for a node value

$\boldsymbol{v}$ and total sample weight w as defined in Equations 2.10 and 2.11, the resulting imprecise impurity is

$$\text{ImpreciseImpurity}(\boldsymbol{v}, w) = \max_{\boldsymbol{p} \in C(w\boldsymbol{v})} \text{Impurity}(\boldsymbol{p}). \tag{3.2}$$

Notice that the impurity now depends on sample size (weight) w in addition to the node value $\boldsymbol{v}$; for very small sample weights, the impurity is always high, but as the weight increases, the imprecise impurity converges to standard impurity. Imprecise impurity also creates the possibility that a split may have negative gain, so it may be used as a stopping condition for tree growth.

While the adaptation of this definition for FDTs is simple, amounting only to allowing sample counts to be non-integer, the computation of imprecise impurity is more complicated. Standard application, as in [122], relies on a simple solution to the maximization in Equation 3.2 when the sample counts are integer and $s \in (0, 2]$. We propose an efficient method for computing imprecise impurity for integer or non-integer sample counts and any $s \in \mathbb{R}_{\geq 0}$. The efficiency is particularly important for tree fitting based on CART, which supports continuous numerical features and may require the evaluation of very many candidate splits for each node.

The problem of selecting the optimal distribution from the credal set can be viewed as the allocation of s sample weight to each of q categories to maximize the impurity. Put differently, it is a perturbation of the probability distribution of limited magnitude to maximize impurity. Consider the gradient of common impurity functions as defined in Equations 2.12 and 2.13.

$$\nabla \text{Gini}(\boldsymbol{v}) = 1 - 2\boldsymbol{v} \tag{3.3}$$

$$\nabla \text{Entropy}(\boldsymbol{v}) = -1 - \log \boldsymbol{v} \tag{3.4}$$

From these, it is evident that for any $v_i < v_j$, increasing v_i and decreasing v_j increases the impurity. Then the optimal perturbation allocates the sample weight to the lowest-weight class(es). This can be accomplished efficiently by sorting the values, iterating with a cumulative sum to determine at which point the s sample weight is fully allocated, assigning the new values, and restoring the values to their original order. The process is outlined in Algorithm 1.

This has $O(q)$ cost, asymptotically no greater than computing impurity. In practice, the loop can be replaced with efficient cumulative sum operations, and the whole procedure can be vectorized over candidate thresholds, resulting in negligible practical cost.

3.3.4 Experiments

We study the effect of KDDTs and credal impurity on the robustness of decision trees to labeling errors. The data is described in Table 2.1. For each data set, we apply uniform label noise with the probability of mislabeling η ranging from 0 to 0.3. We also test a selection of the s parameter for credal impurity from 0 (normal impurity)

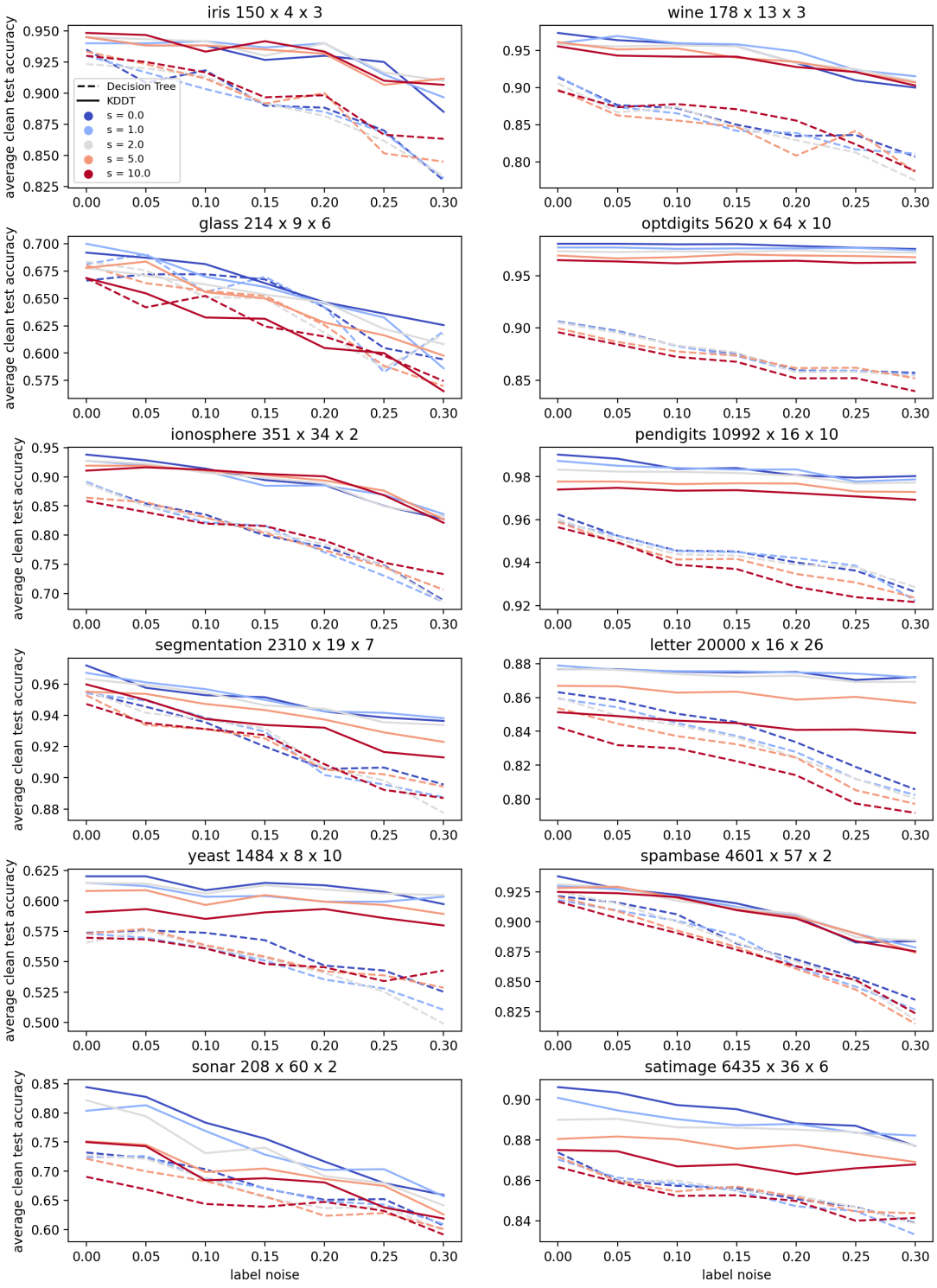


Figure 3.6: Performance of trees with label noise in the training data. The hyperparameter s is for credal impurity.

Algorithm 1 Compute the maximum-impurity member of a credal set.

Input: Node value $\boldsymbol{v} \in [0,1]^q$, sample weight $w \in \mathbb{R}_{>0}$, credal set parameter $s \in \mathbb{R}_{\geq 0}$
Output: Maximal-impurity node value $\boldsymbol{v} \in [0,1]^q$

1: $\boldsymbol{v} \leftarrow w\boldsymbol{v}/(w+s)$ ▷ Scale so that final result is a distribution
2: $s \leftarrow s/(w+s)$ ▷ Scale so that final result is a distribution
3: $\boldsymbol{v} \leftarrow \textsc{Sort}(\boldsymbol{v})$ ▷ Ascending order
4: $\boldsymbol{t} \leftarrow \boldsymbol{0}$ ▷ Total cumulative allocation
5: $i \leftarrow 1$ ▷ Class index
6: **while** $t_i < s$ **do**
7: $\quad t_{i+1} \leftarrow t_i + i(v_{i+1} - v_i)$ ▷ Compute new total allocation
8: $\quad i \leftarrow i + 1$ ▷ Increment class index
9: $\alpha \leftarrow (s - t_{i-1})/(t_i - t_{i-1})$ ▷ Interpolate to correct total allocation
10: $p \leftarrow (1-\alpha)v_{i-1} + \alpha v_i$
11: **for** $j \in [i-1]$ **do** ▷ Set perturbed ratios
12: $\quad v_j \leftarrow p$
13: $\boldsymbol{v} \leftarrow \textsc{RestoreOrder}(\boldsymbol{v})$ ▷ Revert the sort on line 3

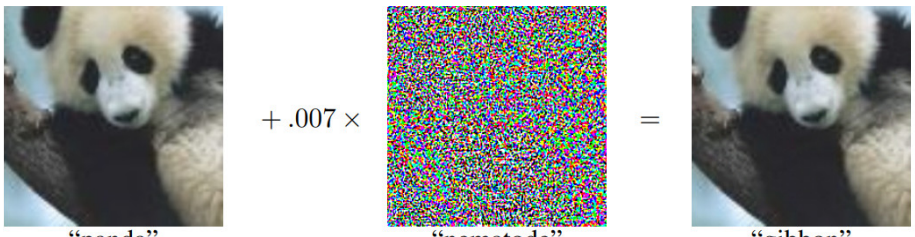
to 10. For context, [122] use $s = 2$. Additional details are in Appendix C.4. For each trial, we measure accuracy on a 20% test split that is clean, that is, not affected by label noise.

The results are shown in Figure 3.6. KDDTs universally outperform DTs and nearly always show less degradation in performance as the label noise increases. On a few data sets, namely optdigits, pendigits, and letter, the KDDTs are hardly affected by label noise at all.

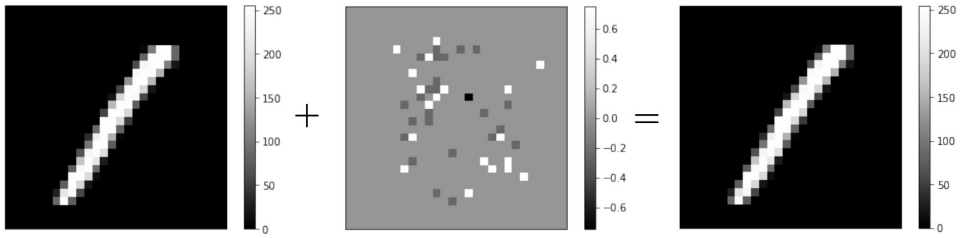
Despite some intial results indicating that the credal impurity could improve robustness, in these results, there not strong evidence that it is beneficial except maybe in some cases for very strong noise. If s is too large, it usually worsens performance. We suspect the reason is that, here, the tree size is tuned by cross-validation. This reduces overfitting to noise and appears to largely negate the benefits of credal impurity.

3.4 Adversarial Perturbation

It was discovered in 2013 by [165] that neural networks are vulnerable to imperceptibly small, but intentional perturbations of the input that cause a change in prediction. Examples are shown in Figure 3.7. These became known as *adversarial examples* or *adversarial perturbations* and spurred a lot of research in model verification, which is used to find, among other safety-related properties, Minimum Adversarial Perturbation (MAP), that is, the minimum-magnitude change in an input to change the prediction. It has also motivated research into building models that are robust to such



(a) An adversarial example from a neural network, from [78].



(b) An adversarial example from a random forest, from [74].

Figure 3.7: Adversarial examples with imperceptibly small perturbation.

perturbations. Random forests and other tree-based models are also vulnerable to adversarial perturbation [74], but have received relatively little attention despite their widespread application, motivating the development of adversarially robust methods for tree-based models.

Randomized smoothing [38] is a model-agnostic method for efficiently training robust models and lower-bounding the robustness of predictions. It is known mainly as the state-of-the-art approach for L2 adversarial robustness, that is, robustness to perturbation bounded by the L2 norm; it is also useful for L1 adversarial robustness, but its practicality diminishes for p-norms with $p > 2$, especially the popular L∞ norm [194]. To train a model with randomized smoothing, choose a smoothing distribution (usually Gaussian for L2 robustness), then train the model on data with random augmentations sampled from the smoothing distribution; then, average predictions over many augmentations from the same distribution. Then the average prediction can be used to compute a high-probability lower bound on the radius of adversarial robustness. This useful because verifying adversarial robustness of a prediction is NP-Hard for popular models, including neural networks [100] and tree ensembles [98]. We show in Section 3.5.3 that robustness verification is also NP-Hard for FDTs.

Our uncertain interpretation formalism is very similar; the only difference is that we smooth the predicted scores instead of the predicted class. Therefore, by simply smoothing predictions instead, our formalism produces adversarially robust models

with an efficient means of estimating robustness. In particular, if smoothing predictions is deterministic as in KDDTs, that is, the smoothed prediction can be computed exactly (see Section 2.3), then there is *no need for randomization*. As a result, there is no inefficiency from sampling many random augmentations, and the robustness bounds are *guaranteed* to hold rather than holding with high probability.

This capability of trees was first discovered by [93], who dubbed it (De-)Randomized Smoothing (DRS) and showed that deterministic smoothing reduces cost and provides larger robustness bounds due to the exponential need for random samples. Lacking a suitable deterministic tree fitting algorithm, their approach is limited to ensembles of decision stumps, that is, trees with only a single split; this both simplifies fitting and makes it possible to smooth the predictions of the ensemble as a whole. Here we show how the KDDT fitting algorithm can be used for DRS of more general tree-based models, propose improved methods for ensemble smoothing, and demonstrate the certifiable adversarial robustness achieved in this manner.

3.4.1 (De-)Randomized Smoothing with KDDTs

Like [93], we will focus on L1 and L2 adversarial robustness by smoothing with box-uniform and Gaussian distributions, respectively[1].

Let $\phi : \mathbb{R}^p \to \mathbb{R}$ be a smoothing distribution and $f : \mathbb{R}^p \to [0,1]^q$ a classifier, and write $\hat{y}(\boldsymbol{x}) = \arg\max_{i \in [q]} f_i(\boldsymbol{x})$ the class predicted by f at $\boldsymbol{x}$. Then define the smoothed predictor $p : \mathbb{R}^p \to [0,1]^q$ as

$$p_c(x) = \mathbb{P}_{\epsilon \sim \phi}[\hat{y}(\mathbf{x} + \epsilon) = c] = \int_{\mathbb{R}^p} \mathbf{1}\{\hat{y}(\mathbf{x} + \epsilon) = c\} \phi(\epsilon)\, d\epsilon$$

for each output index $c \in [q]$. If f is a tree, then p is a KDDT with kernel $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}' - \mathbf{x})$ where each leaf value is set to its argmax indicator after fitting. Then, by the following theorems adapted from [38, 194], we have certified robustness bounds for KDDTs with Gaussian and box kernels.

Theorem 3.3. *Let $\phi \sim \mathcal{N}(0, \sigma\mathbf{I})$ and $\|\epsilon\|_2 \leq r$. Then, for all c,*

$$p_c(\mathbf{x} + \epsilon) \geq \Phi\left(\Phi^{-1}(p_c(\mathbf{x})) - \frac{r}{\sigma}\right) \tag{3.5}$$

where Φ is the Gaussian CDF.

For a model smoothed with an isotropic Gaussian distribution, this provides a means of computing a radius of certified L2 robustness given a smoothed prediction.

Theorem 3.4. *Let $\phi \sim \mathcal{U}_{[-\lambda,\lambda]^p}$ and $\|\epsilon\|_1 \leq r$. Then, for all c,*

$$p_c(\mathbf{x} + \epsilon) \geq p_c(x) - \frac{r}{2\lambda}. \tag{3.6}$$

[1] See [194] for a broader discussion of perturbation norms and smoothing distributions; as usual, we can use any isotropic distribution as a KDDT kernel.

For a model smoothed with a isotropic uniform distribution, this provides a means of computing a radius of certified L1 robustness given a smoothed prediction.

Moreover, for Gaussian smoothing, [110] provides the following.

Theorem 3.5. *Let $\phi \sim \mathcal{N}(0, \sigma\mathbf{I})$ and $\|\epsilon\|_2 \leq r$. Then, for all c,*

$$p_c(\mathbf{x}+\epsilon) \geq \Phi\left(\Phi^{-1}(a + p_c(\mathbf{x})) - \frac{r}{\sigma}\right) - \Phi\left(\Phi^{-1}(a) - \frac{r}{\sigma}\right) \tag{3.7}$$

where a is the (unique) solution to

$$\Phi'(\Phi^{-1}(a)) - \Phi'(\Phi^{-1}(a + p_c(\mathbf{x}))) = -\sigma\|\nabla p(\mathbf{x})\|_2. \tag{3.8}$$

For a model smoothed with an isotropic Gaussian distribution, this provides a means of computing a larger radius of certified L2 robustness given both a smoothed prediction *and* the gradient of the smoothed model at that input.

These provide a practical method for obtaining bounds on the robust radius of KDDT predictions. While setting the leaf values to indicators is not required for these bounds to hold, it does result in larger bounds and has little effect on the natural accuracy, so it is recommended.

3.4.2 Certification of Ensembles

While the KDDT prediction algorithm offers an efficient mechanism for DRS of single-tree models, it does not offer such a mechanism for ensembles. It is safe to simply use the average smoothed prediction, since it is equivalent to the smoothed prediction of the averaged models, but this can result in very poor bounds. As a worst-case example, consider an ensemble of $2n+1$ classifiers in which n classifiers always predict class 0 and $n+1$ classifiers always predict class 1; then, as n grows large, their average prediction approaches 0.5, so the certifiable robust radius approaches 0. However, the true robust radius is infinite for all n. Figure 3.8 shows an example of this gap on the moons data set. Despite nearly identical predictions, the RF has poor certifiable robustness due to averaging of predictions. If we were able to smooth the predicted label and not the predicted score of the ensemble, this gap would vanish. Though a tree ensemble is piecewise-constant, the number of pieces is at worst exponential in the number of trees, so there is no general, efficient way to accomplish this. In fact, we show that the problem of smoothing tree ensembles is NP-Hard by reduction from 3-SAT. This is proven in Appendix B.

Theorem 3.6. *Computing the smoothed prediction of a tree ensemble is NP-Hard.*

This motivates the development of a better approach for computing robustness bounds when it is efficient to smooth predictions of individual models, but not ensembles. We propose the following. Write the smoothed ensemble $p(\mathbf{x}) = \frac{1}{n}\sum_{i\in[n]} p^{(i)}(\mathbf{x})$

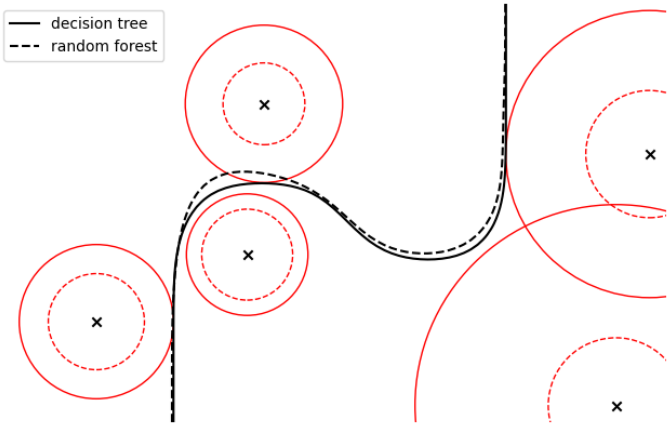


Figure 3.8: Several L2 robustness bounds computed using Equation 3.5 for a KDDT DT and KDDT RF trained on the moons data set. Despite their near-identical predictions, the certifiable robustness bounds are smaller for the RF because of the averaging of its trees' predictions.

for smoothed constituents $p^{(i)}$, $i \in [n]$. For a given input $\mathbf{x}$ and class c, let $\rho^{(i)}(r)$ denote the lower bound on $p_c^{(i)}(\mathbf{x}+\epsilon)$ for $\|\epsilon\| \leq r$. Then we have ensemble bound

$$\rho(r) \geq \frac{1}{n} \sum_{i \in [n]} \rho^{(i)}(r) \tag{3.9}$$

where each $\rho^{(i)}$ is nonincreasing in r and efficient to evaluate. Therefore we can simply use binary search over r to determine at what radius the ensemble prediction is tightly bounded above a decision threshold (typically 0.5). In general, this approach makes no difference if the ensemble predictions are similar, but improves the bound more and more as the variance of the ensemble predictions increases, as shown in Figure 3.9. In the example given at the beginning of this section, where the naively computed robustness bound is arbitrarily poor, this approach completely recovers the true level of robustness for Gaussian smoothing, showing that there is potential to greatly improve the provable bounds for ensembles.

Despite this, there are significant limitations. First, the fact that Equation 3.6 is linear means that this method rarely makes a difference when smoothing with box kernels; it simply informs the bound computation that any tree cannot predict a value less than zero. Moreover, when using gradients, it can actually yield a worse bound compared to the naive method of averaging predictions and gradients and using Equation 3.7. Consider an ensemble of two trees with equal predictions and maximum-magnitude gradients in opposite directions; then our proposed method cannot use the gradients to improve the bound. However, the naive method leverages the fact that the average gradient is zero to improve the bound. Another example is when ensemble constituents are vulnerable to perturbation, but in orthogonal directions.

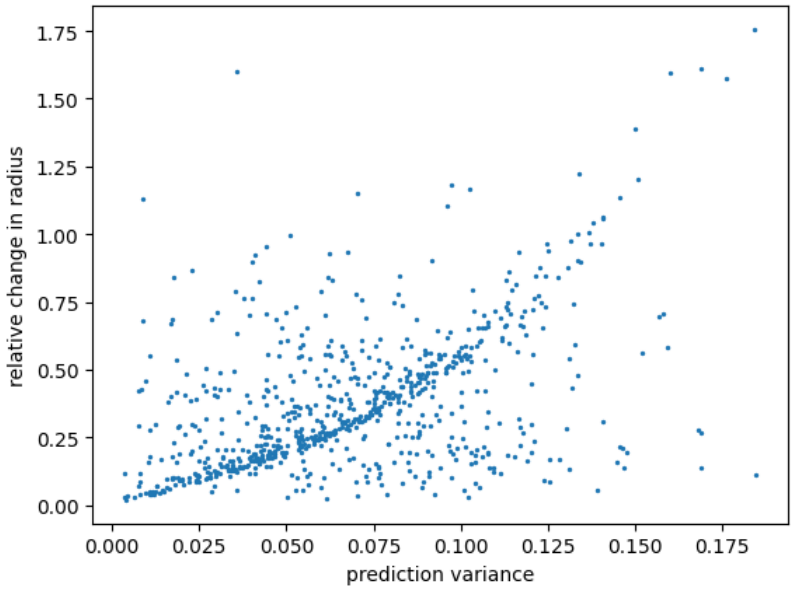


Figure 3.9: Change in certified radius by using Equation 3.9 with 3.5 for 1000 sets of 5 uniformly random predictions.

In both cases, the ensemble is more robust than its most robust constituent, but it is clear from Equation 3.9 that this approach can never certify that an ensemble is more robust than its most robust constituent, whereas the naive approach potentially can.

These cases demonstrate that there is still room for improvement, particularly by incorporating gradient directions. One approach to accomplish this would be to make the perturbation bound directional: for direction vector $\mathbf{v} \in \mathbb{R}^p$ with $\|\mathbf{v}\| = 1$, let $\rho^{(i)}(r, \mathbf{v})$ lower bound $p_c(\mathbf{x} + \epsilon\mathbf{v})$ for $\epsilon \in [0, r]$. If each $\rho^{(i)}(r, \cdot)$ is convex or log-convex, then at each r in the binary search, one can solve for the direction $\mathbf{v}$ that minimizes the ensemble bound. We conjecture that such a bound exists for the smoothing distributions discussed here, but leave their discovery to future work.

3.4.3 Experiments

To evaluate our approach, we imitate the experimental setup of [93], who first proposed DRS with trees. In particular, we use five binary classification data sets and, for each, train models for L1 robustness with box kernel with radius λ and for L2 robustness with Gaussian kernel with standard deviation σ, as shown in Table 3.1. Additional details are in Appendix C.5. We omit standard random forests as a baseline because L1 and L2 robustness verification is difficult.

Data Set	n	p	λ (L1)	σ (L2)
Breast Cancer Wisconsin (Diagnostic)	683	9	2.00	0.25
Pima Indians Diabetes Database	768	8	0.28	0.15
MNIST 1 vs. 5	14,000	784	4.00	0.25
MNIST 2 vs. 6	14,000	784	4.00	0.25
FMNIST Shoes	14,000	784	4.00	0.25

Table 3.1: Description of data sets and smoothing parameters, from [93].

Comparison against Decision Stump Ensembles

We first compare against Decision Stump Ensemble (DSE) [93], to our knowledge the only existing method using DRS with trees. We evaluate using the same two metrics: first, the Average Certified Radius (ACR), that is, the average certifiable radius of predictions on the test set, where incorrect predictions have radius 0; second, the certified accuracy at several radii, that is, the percentage of test data at which the prediction is correct and the certifiable robust radius is sufficiently large. The results are shown in Table 3.2.

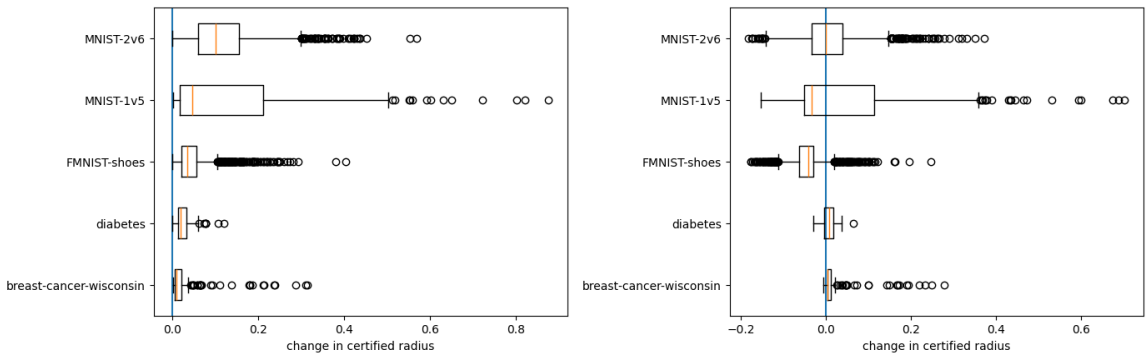
If we compare KDDT-based methods to DSE, for L1 robustness, KDDTs achieve higher certified accuracy than DSE at all the examined radii, but sometimes fall slightly short in ACR. For L2 robustness, KDDT-based methods achieve higher certified accuracy at smaller radii, but lower at larger radii, and their ACR is much lower on the MNIST data sets. KDDTs also achieve better clean accuracy across the board. From these observations, we conclude that KDDTs achieve better robust performance at small to moderate radius, but robust performance drops off at the "tails" of the smoothing distribution. If a higher average radius is desired, one can simply make the smoothing kernel larger. The fact that this effect is more severe in the L2 cases is likely due to the approximation of the smoothing kernel used for fitting, where the tails of the Gaussian distribution are truncated. Using a less truncated kernel with more pieces would improve it, but increase fitting cost, which is already fairly high due to the large kernels being used.

If we compare KDDT trees and forests, we see that they are usually close in performance; this is consistent with our understanding of the regularizing effect of smoothing, which reduces model variance and decreases the need for ensemble methods, as discussed in Section 3.1. This is a particularly extreme case where, since the kernels are larger than one would use when optimizing for clean accuracy, the gap between trees and ensembles becomes very small. The forests are also at a disadvantage since their predictions cannot be smoothed in the ideal sense, as established by Theorem 3.6. We see that trees always match or outperform forests in the L1 robustness experiments, but forests often take the lead in the L2 robustness experiments.

The standard decision tree always has very poor certified accuracy, but sometimes has higher clean accuracy than any of the robust models, pointing to an accuracy-

Norm	Data Set	Method	ACR	Certified Accuracy at Radius				
				0.00	0.10	0.25	0.50	1.00
L1	Breast Cancer	DT	0.255	94.2	87.6	20.4	9.5	0.0
		KDDT DT	**1.601**	**100.**	**100.**	**98.5**	**92.7**	**81.0**
		KDDT RF	1.596	**100.**	**100.**	**98.5**	**92.7**	**81.0**
		DSE	1.396	95.2	94.1	92.1	87.1	72.8
	Diabetes	DT	0.079	**77.9**	36.4	4.5	**0.0**	**0.0**
		KDDT DT	0.149	74.0	**59.7**	**33.8**	**0.0**	**0.0**
		KDDT RF	0.146	73.4	59.1	29.2	**0.0**	**0.0**
		DSE	**0.153**	73.7	58.3	30.1	**0.0**	**0.0**
	MNIST 1 vs. 5	DT	0.066	**98.9**	38.9	0.0	0.0	0.0
		KDDT DT	**3.632**	98.7	**98.7**	**98.6**	**98.4**	**97.5**
		KDDT RF	3.532	98.4	98.4	97.9	97.7	97.1
		DSE	3.425	96.2	95.9	95.4	94.7	93.0
	MNIST 2 vs. 6	DT	0.152	**98.2**	46.7	17.8	7.6	0.0
		KDDT DT	3.032	97.5	**97.3**	**97.0**	**96.2**	**94.1**
		KDDT RF	2.888	96.6	96.5	96.1	95.5	92.0
		DSE	**3.243**	95.7	95.5	95.1	94.5	92.8
	FMNIST-Shoes	DT	0.057	**95.2**	16.4	0.4	0.1	0.0
		KDDT DT	2.517	89.5	**89.0**	**88.2**	**86.4**	**82.2**
		KDDT RF	2.447	87.4	86.7	85.8	83.8	79.8
		DSE	**2.731**	84.4	83.8	83.1	81.7	79.1
L2	Breast Cancer	DT	0.221	94.2	80.3	18.2	8.0	0.0
		KDDT DT	0.631	**100.**	98.5	95.6	**82.5**	3.6
		KDDT RF	0.610	**100.**	**99.3**	**96.4**	81.8	2.9
		DSE	**0.653**	93.3	90.6	87.3	73.8	**15.5**
	Diabetes	DT	0.070	**77.9**	31.8	0.6	**0.0**	**0.0**
		KDDT DT	**0.153**	73.4	58.4	**27.9**	**0.0**	**0.0**
		KDDT RF	0.150	74.0	**59.7**	**27.9**	**0.0**	**0.0**
		DSE	0.124	72.7	53.0	15.6	**0.0**	**0.0**
	MNIST 1 vs. 5	DT	0.066	98.8	39.5	0.0	0.0	0.0
		KDDT DT	0.691	**99.6**	**99.4**	98.7	**95.0**	0.3
		KDDT RF	0.664	99.5	**99.4**	**98.9**	92.2	0.0
		DSE	**1.720**	95.3	94.8	94.0	92.3	**87.9**
	MNIST 2 vs. 6	DT	0.088	98.0	30.5	1.2	0.6	0.0
		KDDT DT	0.664	98.8	98.4	97.3	91.1	0.0
		KDDT RF	0.594	**99.4**	**99.2**	**97.7**	79.5	0.0
		DSE	**1.613**	95.5	94.9	93.9	**91.7**	**84.9**
	FMNIST Shoes	DT	0.047	**95.0**	8.8	0.1	0.0	0.0
		KDDT DT	0.486	94.4	**92.6**	**86.2**	58.0	0.0
		KDDT RF	0.459	94.0	92.3	84.0	49.4	0.0
		DSE	**1.334**	85.0	83.7	81.5	**78.1**	**68.6**

Table 3.2: Comparison of robust accuracy. DSE results from [93].



(a) Without gradient (Equation 3.5). (b) With gradient (Equation 3.7).

Figure 3.10: Change in certified radius by using our method, normalized by bandwidth.

robustness tradeoff when using heavy smoothing. The enormous gap in certified accuracy highlights the efficacy of the smoothed models.

Improvement of Ensemble Smoothing

We next report the distribution of the difference in certified radius over the test set, normalized by the smoothing kernel bandwidth, by using our method based on Equation 3.9. As this mainly benefits L2 bounds with Gaussian kernels, we focus on those, both with and without incorporating gradients. The results are in Figure 3.10.

As expected, without using gradients, we see that our method improves the bound for most samples, sometimes by quite a lot. In particular, the benefit is potentially large for the many-featured MNIST data sets, but smaller for the tabular data sets.

Also as expected, we see that with gradients, our approach can either improve the bound, or make it worse, since it discards gradient direction and only uses gradient magnitude, as discussed in Section 3.4.2. However, the benefit is still large for a nontrivial number of samples, so it is best used to supplement, but not replace, the conventional approach for computing a bound.

3.5 Verification

While robustness methods aim to improve a model's conformity to very specific safety specifications, *verification* is the more general problem of proving or finding counterexamples, usually by framing as a satisfiability problem, for a wide variety of specifications. These often include robustness properties, particularly adversarial robustness, where general verification algorithms are a popular approach for precisely measuring robustness. Others include application-specific safety properties; for instance, [100] verify properties of neural networks for aircraft collision avoidance, such as *if the*

intruder is near and approaching from the left, the network advises "strong right". These have since become a benchmark for neural network verification algorithms.

This kind of verification task is often inherently intractable; it was discovered that verification is NP-Complete for tree ensembles [98] and for neural networks [100], two of the most common model classes in modern machine learning. However, it is understood that satisfiability problems with an underlying structure are often much faster to solve in practice than the worst case, and a huge amount of work by the ML verification & validation research community has made it possible to verify both kinds of models at fairly large scale in practice.

In particular, abstraction-refinement is a verification paradigm whereby an efficient over-approximation of the model, that is, an upper and lower bound on the model that are easier to analyze than the model itself, is used to check a property. If the approximation fails to conclusively determine the truth of the property, usually due to a spurious counterexample, the approximation is refined and checked again. This can speed up verification of some models [169] and can make possible the verification of models that are not amenable to non-approximating strategies, such as neural networks with activation functions that are not piecewise linear [141].

Such is also the case for Fuzzy Decision Tree (FDT)s, as we show in this work. FDTs are a strong candidate for trustworthy AI applications; their continuous decision allocation can improve the expressiveness of a single tree, enable smooth regression, reduce overfitting, and provide differentiability, enabling both fully gradient-based optimization as in [163] and hybrid strategies, as we propose in Chapter 4. Compared to neural networks, such trees have fewer design choices since the architecture can be chosen automatically, and they can be less sensitive to hyperparameters [155]. In addition, FDTs leveraging oblique splits consisting of a linear combination of features can be both smaller and more performant than axis-aligned trees [197, 121, 158, 75]. Less complex models are also generally faster to verify. However, prior to our work, there has been no practical method for verification of FDTs, creating a major gap in their trustworthy application.

FDTs are superficially similar to neural networks; both propagate values through a layered graph structure and can backpropagate to get loss gradients and update parameters, and the splitting functions of FDTs resemble activation functions of neural networks. However, neural networks directly transform their input, while FDTs instead partition it using soft boundaries. As a result, verification of FDTs presents different challenges compared to verification of neural networks. A neural network is a composition of its activation functions, whereas an FDT is a product of its splitting functions. This means that, for example, a neural network with piecewise-linear activations is piecewise linear, whereas an FDT with piecewise-linear splitting functions is piecewise-polynomial with degree up to the tree depth, which is relatively difficult to analyze. As a result, while the hardness of verification of ReLU networks and tree ensembles is a result of combinatorial challenges, the hardness of verification of FDTs arises from both combinatorial challenges and their ability to directly represent

non-convex functions. We discuss these challenges in Section 3.5.3 and show that, as a result, verification of FDTs is NP-Complete, similarly to other learned models.

In this section, we propose an iterative abstraction-refinement algorithm that can refine upper and lower bounds of the extreme value of an FDT in a given region until the desired property is determined, as originally published in [76]. Specifically, given an FDT $f : \mathbb{R}^p \to \mathbb{R}^q$, convex domain $D \subseteq \mathbb{R}^p$, $\boldsymbol{a} \in \mathbb{R}^q$, and $b \in \mathbb{R}$, by iteratively partitioning D, it can

1. prove $\boldsymbol{a}^\top f(\boldsymbol{x}) \leq b$ or $\boldsymbol{a}^\top f(\boldsymbol{x}) < b$ for all $\boldsymbol{x} \in D$, or find a counterexample
2. find $\max_{\boldsymbol{x} \in D} \boldsymbol{a}^\top f(\boldsymbol{x})$ within given tolerance or timeout.

The first is our form for verification, which can be applied to prove or disprove a variety of safety properties, including the popularly studied property of local adversarial robustness; additional properties can be verified by manually altering f, as discussed in the Additional Properties subsection of 3.5.2. The latter is a utility that arises naturally from the verification methodology. In finite precision, the algorithm is sound and complete, that is, guaranteed to terminate with a correct result in finite time.

We demonstrate the usefulness of our algorithm by benchmarking time for two verification tasks: finding minimum adversarial perturbation and testing global adversarial robustness. For the minimum adversarial perturbation tests, we also compare against a Satisfiability Modulo Theories (SMT) solver. Since SMT queries with nonlinear arithmetic are not necessarily decidable, SMT is generally both limited to FDTs with piecewise-linear splitting functions and incomplete for FDT verification, but due to the lack of prior work in FDT verification, it is the best baseline available for this task. As an additional baseline, we use a variation of our algorithm with a refinement strategy based on the most similar methods for neural network verification. We use a selection of 10 public data sets with varying sizes and complexities. Ultimately, we find that our method is much faster than the baselines, even when comparing only on cases where the baselines terminate with a solution. Our implementation and experiment code is publicly available[2].

3.5.1 Related Work: Abstraction-Refinement Algorithms

As we cover related work in FDTs in Section 2.3.2 and 4.1.1, here we will cover work related work in the verification of machine learning models, specifically, those based on the abstraction-refinement paradigm.

Most of such work has focused on neural networks. As one of the first works in machine learning verification as a whole, [141] are also the first to apply abstraction-refinement to neural networks. Their approach partitions the domain of the activation function into intervals of equal size and uses constant bounds on each interval to

[2] https://github.com/autonlab/fdt_verification

enable an MLP to be represented as Boolean combinations of arithmetic constraints and checked with the solver HySAT. If a counterexample is found, it is checked against the real model and, if it does not actually violate the property, the interval size is decreased, refining the approximation, and the procedure repeated until the property is proven or a true counterexample is found. The approach is only demonstrated on extremely small networks.

Starting in 2018, in response to the limited scalability of complete, satisfiability-based methods such as [100], many abstraction methods were developed to scale to larger networks. [67] leverage extensive work in abstract interpretation and propose the use of abstract domains, such as boxes, zonotopes, and polyhedra, with abstract transformers for common neural network structures, including feedforward and convolutional layers with ReLU activations as well as max pooling layers. [128] adapt this concept to be used during training to produce networks that are more verifiably robust compared to approaches using projected gradient descent to find adversarial examples for training. [7] then introduce optimization techniques to automatically refine the abstraction. The algorithm is shown to be δ-complete, meaning that for a given δ, any returned counterexample is within δ of a true counterexample. [171] propose to represent domains as star sets, which enables both complete verification and less conservative over-approximation compared to zonotopes. [170] then introduce ImageStars, a variant of star sets for representing sets of images, for the verification of convolutional neural networks. [172] use zonotopes as an efficient overapproximation of star sets in a pre-filtering process, effectively introducing an abstraction-refinement strategy and improving the scalability of the star set approach, and extends star set methods to a wider class of piecewise-linear activation functions. [10] introduce data structure and algorithmic changes, including several optimizations, achieving unprecedented performance on verification benchmarks.

Another family of methods, which has come to be called LiRPA (Linear Relaxation Perturbation Analysis), admit general activations by bounding them with simpler functions, usually linear. [199] and [157] are optimized for the fast, but incomplete setting. [190] extend LiRPA to general architectures such as convolutional layers. [191] then introduce refinement by combining the approach with existing branch-and-bound methods, with the result being both complete and faster than existing complete methods based on branch-and-bound. Similarly, [182] improve over [199] by eliminating the need for LP solvers and adding an optimization-based refinement capability. They demonstrate superior speed in the complete setting and tightness of bound in the incomplete setting.

Differently from any of the aforementioned abstraction approaches, [52] perform abstraction by merging neurons to form a smaller network that over-approximates the original network and refinement by splitting the merged nodes, with the choice guided by spurious counterexamples. The smaller network is then used with other verification methods, thereby introducing an abstraction-refinement capability. The authors use their method with the toolset proposed by [102], a more recent satisfiability-based

verifier based on [100].

With the greatest resemblance to our approach, [181] use constant upper and lower bounds of activation functions, like [141], but instead of producing constraints to a solver, computes network bounds directly, an approach they call symbolic interval analysis. The approximation is refined by bisecting the input range. They show that the process converges in finite steps for Lipschitz continuous networks. [180] combine this symbolic interval analysis with linear rather than constant bounds, called symbolic linear relaxation. They also propose an alternative iteration approach that refines the approximation at the most over-approximated nodes. For FDTs, we use constant, rather than linear, bounds on the splitting function; since the model consists of successive multiplications of these functions, bounding the output is difficult even with a linear relaxation.

In the domain of tree ensemble verification, where the main challenge is the combinatorial explosion resulting from the number of trees, [169] propose an abstraction-refinement approach that refines a hyperrectangle approximation by splitting the input region in such a way that a tree can be eliminated from the ensemble. In the worst case, this continues until no trees remain, resulting in a list of input hyperrectangles and associated output values, which fully specifies the tree ensemble and can be checked exhaustively for the property of interest.

We are aware of no prior work addressing the verification of FDTs. We believe that this is because they are not as commonly applied as the very popular neural networks and tree ensembles, and, more importantly, because the problem of verifying them presents unique challenges. For example, the multiplication of splitting functions means that linear relaxations are not clearly useful, and it causes SMT-based approaches, a popular branch of complete verification methods for neural networks, to be incomplete even for linear splitting functions. Our overall framework is most similar to that of [181] in that both approaches use approximations of individual model components to directly compute overall output bounds, then split the input to refine those bounds, making them easily parallelizable. What makes our approach unique is our novel, flexible, and theoretically grounded methods for computing the bounds and selecting the split for FDTs, which have fundamentally distinct structure from neural networks, as well as the accompanying theory and analysis. In particular, we (1) introduce an efficient interval approximation for FDTs, whereas [181] propose an approximation for neural networks; (2) propose a method for choosing linear domain splits that maximally reduce overapproximation under worst-case structural assumptions, whereas [181] use a heuristic splitting rule that bisects on the feature with the largest gradient; (3) support any convex domain for verification, enabling robustness verification for any p-norm, whereas [181] use hyper-interval domains, limiting robustness verification to L∞ only; (4) support any nondecreasing splitting function, whereas [181] as proposed supports only a few piecewise-linear activation functions; and (5) provide a flexible framework for verifying complex properties involving modifications to both the input and output, such as global adversarial robustness, whereas

[181] do not.

3.5.2 Fuzzy Decision Tree Verification Algorithm

Here we consider a subclass of the FDTs specified in Section 2.3.3 where splits are based on a linear combination of features and a threshold, which encompasses axis-aligned FDTs with a single feature and threshold at each node. In particular, given a nondecreasing splitting function σ, the decision allocation function σ_i at each node i is defined as

$$\sigma_i(\boldsymbol{x}) = \sigma(\boldsymbol{a}_i^\top \boldsymbol{x} + b_i)$$

for learned parameters $\boldsymbol{a}_i \in \mathbb{R}^p$ and $b_i \in \mathbb{R}$. We do not actually require that σ be the same at all nodes, but this is the common practice, so we present it as such. The prediction is then as in Equations 2.3 and 2.8. When the structure is fixed, the model is parametric and differentiable and therefore can be trained using gradients. The structure and initial parameters are usually taken from a trained crisp tree.

Our overall procedure first uses an efficient abstraction to bound the output of f on a domain D. If the bound is sufficiently precise, we are finished; otherwise, we refine the approximation by splitting D into two subdomains and repeat until the required precision is attained. In the case of verification of the property "$\boldsymbol{a}^\top f(\boldsymbol{x}) \leq b$ for all $\boldsymbol{x} \in D$", this occurs when either the upper bound is at most b for all subdomains of D and the property is proven, or the lower bound is greater than b for some subdomain and a counterexample is found. Likewise for properties of the form "$\boldsymbol{a}^\top f(\boldsymbol{x}) < b$ for all $\boldsymbol{x} \in D$". The following sections describe the processes of abstraction and refinement, then discuss properties of the problem and algorithm to justify the approach.

Abstraction

Given an FDT f, vector $\boldsymbol{a}$, and bounded convex domain D, we aim to find upper and lower bounds of $\boldsymbol{a}^\top f(\boldsymbol{x})$ on $\boldsymbol{x} \in D$ that are ideally close to the true minimum and maximum while remaining efficient to compute. To do this, we first find the extreme values of σ_i on D for each node.

$$\begin{aligned} \sigma_i^{\min} &= \min_{\boldsymbol{x} \in D} \sigma_i(\boldsymbol{x}) = \sigma\left(\min_{\boldsymbol{x} \in D} \boldsymbol{a}_i^\top \boldsymbol{x} + b_i\right) \\ \sigma_i^{\max} &= \max_{\boldsymbol{x} \in D} \sigma_i(\boldsymbol{x}) = \sigma\left(\max_{\boldsymbol{x} \in D} \boldsymbol{a}_i^\top \boldsymbol{x} + b_i\right) \end{aligned} \tag{3.10}$$

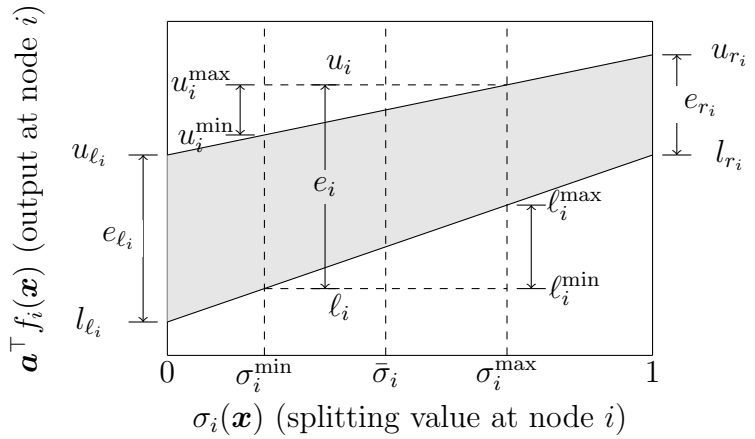


Figure 3.11: Geometry of bounding at an internal node. The gray band shows the output range given the childrens' output ranges.

These can be computed by convex optimization. Then a straightforward recursion determines the lower bound l_i and upper bound u_i from the children ℓ_i and r_i of i.

$$\begin{aligned} l_i^{\min} &= (1-\sigma_i^{\min})l_{\ell_i} + \sigma_i^{\min} l_{r_i} \\ l_i^{\max} &= (1-\sigma_i^{\max})l_{\ell_i} + \sigma_i^{\max} l_{r_i} \\ u_i^{\min} &= (1-\sigma_i^{\min})u_{\ell_i} + \sigma_i^{\min} u_{r_i} \\ u_i^{\max} &= (1-\sigma_i^{\max})u_{\ell_i} + \sigma_i^{\max} u_{r_i} \\ l_i &= \min(l_i^{\min}, l_i^{\max}) \\ u_i &= \max(u_i^{\min}, u_i^{\max}) \end{aligned} \tag{3.11}$$

Here $l_i^{\min}$ and $l_i^{\max}$ are lower bound candidates computed using the minimum and maximum split value, respectively, and likewise for $u_i^{\min}$ and $u_i^{\max}$. These will be used again later since the difference in each pair indicates the contribution of the node's split to the overall range of the bound. For leaves, the output is constant, so $l_i = u_i = \boldsymbol{a}^\top \boldsymbol{v}_i$. A geometric interpretation of this bounding scheme is illustrated in Figure 3.11.

Refinement

The overapproximation error in the abstraction comes from using a constant bound of σ_i at each node. To reduce the error, we split the domain such that, for some node i, the range of $(\sigma_i^{\min}, \sigma_i^{\max})$ becomes smaller on the new subdomains, thereby improving the approximation, as shown in Figure 3.12. In particular, we split along a hyperplane satisfying $\boldsymbol{x}^\top \boldsymbol{a}_i + b_i = c$, where $\sigma(c-\epsilon) \le \bar{\sigma}_i \le \sigma(c+\epsilon)$ for all $\epsilon > 0$ and

$$\bar{\sigma}_i = \frac{1}{2}(\sigma_i^{\min} + \sigma_i^{\max}). \tag{3.12}$$

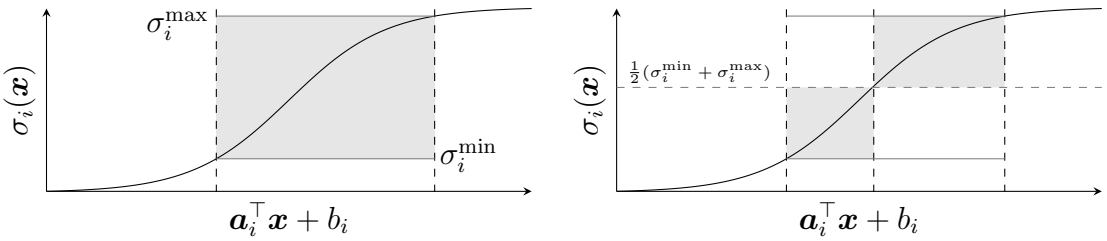


Figure 3.12: Splitting the domain refines the approximation.

This formulation is designed to account for non-invertible σ; if σ is invertible, then we can simply use $c = \sigma^{-1}(\bar{\sigma}_i)$. Splitting at $\bar{\sigma}_i$ causes the resulting subdomains to have equal approximation errors at node i.

To select at which node to perform the split, we define $v(i, j)$ that measures the amount of variation in the approximation at node i coming from internal j.

$$v(i,j) = \begin{cases} \frac{1}{2}\left(\left|l_i^{\min} - l_i^{\max}\right| + \left|u_i^{\min} - u_i^{\max}\right|\right) & \text{if } i = j \\ (1-\bar{\sigma}_i)v(\ell_i, j) & \text{if } j \in T_{\ell_i} \setminus T_{r_i} \\ \bar{\sigma}_i v(r_i, j) & \text{if } j \in T_{r_i} \setminus T_{\ell_i} \\ (1-\bar{\sigma}_i)v(\ell_i, j) + \bar{\sigma}_i v(r_i, j) & \text{if } j \in T_{\ell_i} \cap T_{r_i} \end{cases} \tag{3.13}$$

Here T_i is the set of nodes in the subtree rooted i. We include $T_{\ell_i} \cap Tr_i$ even though this is not possible in trees to support DAG models that enable verification of additional properties, as discussed in the Additional Properties section. With v defined, we refine our approximation by splitting D at the node j where $v(i, j)$ is maximized when i is the root. Theorem 3.8 justifies this strategy.

Since we are interested in the maximum value of $\boldsymbol{a}^\top f(\boldsymbol{x})$ on D, we keep a set of the subdomains partitioning D (which initially contains only D), remove the one with the highest upper bound, split it, bound on the resulting domains, and add them back to the set. Prioritizing domains with the highest upper bound allows us to efficiently hone in on the maximum over D. The set is implemented as an efficient priority queue such as a max heap. During the process, bounds on $\max_{\boldsymbol{x} \in D} a^\top f(\boldsymbol{x})$ are continuously updated and the refinement is repeated until a stopping condition is met. This could be a desired tolerance, a timeout, a number of iterations, or a property $\boldsymbol{a}^\top f(\boldsymbol{x}) \leq b$ or $\boldsymbol{a}^\top f(\boldsymbol{x}) < b$ to be proven or disproven. Such a property is disproven by finding a domain with lower bound greater than (or equal to, depending on the property) b, so a counterexample can be produced by taking any point in that domain, or the domain itself can be interpreted as a counterexample. If more counterexamples are desired, refinement can be allowed to continue. The process is outlined in Algorithm 2.

The bounding scheme involves optimizing linear functions on convex domains many times at each refinement, which is the vast majority of the computation time spent by the algorithm. To reduce this as much as possible, we use a strategy that redefines the model's subtree split to allow dynamic pruning by allowing splits to

Algorithm 2 Refine bounds of $\max_D \boldsymbol{a}^\top f(\boldsymbol{x})$.

$H \leftarrow$ max heap
$l, u \leftarrow \textsc{Bound}(f, D)$
while stopping condition on l, u is not met **do**
    $D^-, D^+ \leftarrow \textsc{Split}(f, D)$
    $l^-, u^- \leftarrow \textsc{Bound}(f, D^-)$
    $l^+, u^+ \leftarrow \textsc{Bound}(f, D^+)$
    $\textsc{Push}(H, u^-, D^-)$
    $\textsc{Push}(H, u^+, D^+)$
    $l \leftarrow \max(l, l^-, l^+)$
    $u, D \leftarrow \textsc{Pop}(H)$

have value exactly 0 or 1, if they can not already.

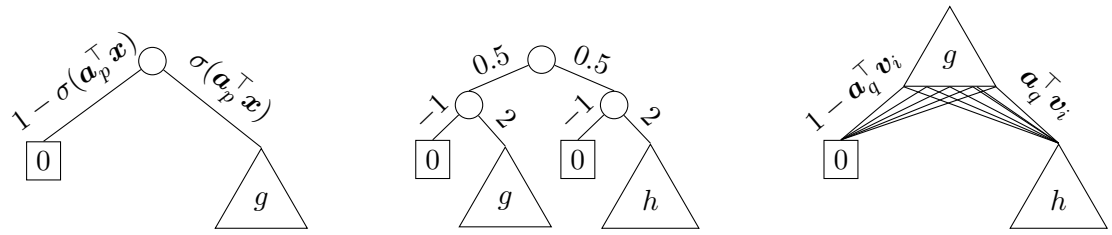
$$\sigma_i(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \sigma(\boldsymbol{a}_i^\top \boldsymbol{x} + b_i) < \epsilon \\ 1 & \text{if } \sigma(\boldsymbol{z}_i^\top \boldsymbol{x} + b_i) > 1 - \epsilon \\ \sigma(\boldsymbol{a}_i^\top \boldsymbol{x} + b_i) & \text{otherwise} \end{cases}$$

Here ϵ is a small chosen value. Defined as such, subtrees that have exactly zero weight within a domain can be removed and the resulting pruned tree stored along with the domain in the priority queue, reducing computation for future bounds. The greater the threshold chosen, the greater the potential to save time by pruning, but too large a value may negatively affect the performance of the FDT.

Additional Properties

Though the presented algorithm can only verify properties of the form $\boldsymbol{a}^\top f(\boldsymbol{x}) \leq b$ or $\boldsymbol{a}^\top f(\boldsymbol{x}) < b$ directly, we can actually verify a much broader class of properties by constructing a new tree that contains f as a subtree, then verifying it. In fact, we can generalize further by, within the constructed part, allowing a node to have multiple parent nodes, resulting in a Directed Acyclic Graph (DAG) containing one or more copies of f. We refer to this kind of structure as a Fuzzy Decision DAG (FDD). Though there are similar decision DAG models used for prediction, here we use it only as a tool to verify more complex properties of FDTs; an FDD can efficiently represent a broader class of functions than an FDT by avoiding exponential size increase due to redundant subtrees. The only necessary change to the methodology to verify FDDs is to ensure that the bounds and the v values are computed with memoization, a strategy of storing intermediate results to avoid redundant computation. This extension is the purpose of the $j \in T_{\ell_i} \cap T_{r_i}$ case in the definition of v.

In each constructed node, we may choose whatever σ, $\boldsymbol{a}_i$, and b_i we want, as long as σ is nondecreasing and defined on $\mathbb{R}$. It need not be the same at every node, and unlike the σ in the learned model, its range is not restricted to $[0, 1]$.



(a) $f(\boldsymbol{x}) = \sigma(\boldsymbol{a}_p^\top \boldsymbol{x})g(\boldsymbol{x})$ (b) $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ (c) $f(\boldsymbol{x}) = (\boldsymbol{a}_q^\top g(\boldsymbol{x}))h(\boldsymbol{x})$

Figure 3.13: Basic structures for combining FDTs and FDDs.

Theorem 3.7 describes building blocks for functions that can be represented in this way, Figure 3.13 gives a visual summary, and our public code includes functions to construct FDDs according to these operations and demonstrations of their use. We note that Theorem 3.7 additionally implies that our method can verify ensembles of FDTs.

Theorem 3.7. *Given FDDs g and h, for any $\boldsymbol{a}_p \in \mathbb{R}^p$, $\boldsymbol{a}_q \in \mathbb{R}^q$, and $\sigma : \mathbb{R} \to \mathbb{R}$, there exist FDDs $f(\boldsymbol{x}) = \sigma(\boldsymbol{a}_p^\top \boldsymbol{x})g(\boldsymbol{x})$, $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$, and $f(\boldsymbol{x}) = (\boldsymbol{a}_q^\top g(\boldsymbol{x}))h(\boldsymbol{x})$ with size $O(\text{size}(g) + \text{size}(h))$.*

Proof. Construct $f(\boldsymbol{x}) = \sigma(\boldsymbol{a}_p^\top x)g(\boldsymbol{x})$ by creating root node i with $\boldsymbol{a}_i = \boldsymbol{a}_p$, $b_i = 0$, splitting function σ, left child the zero leaf, and right child g. This introduces two additional nodes for total size $O(\text{size}(g))$.

Construct $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ by creating root node i with $\sigma_i(\boldsymbol{x}) = 1/2$, left child $2g$, and right child $2h$, with $2g$ and $2h$ constructed as above. This introduces five additional nodes for total size of $O(\text{size}(g) + \text{size}(h))$.

Construct $f(\boldsymbol{x}) = (\boldsymbol{a}_q^\top g(\boldsymbol{x}))h(\boldsymbol{x})$ by changing each leaf i of g into an internal node with $\sigma_i(\boldsymbol{x}) = \boldsymbol{a}_q^\top \boldsymbol{v}_i$, left child the zero leaf, and right child h. Now f is a DAG. This adds one node for a total size of $O(\text{size}(g) + \text{size}(h))$. □

This may seem not to include verification of properties with operations such as division and non-integer exponents. We indeed cannot maximize such functions, but we often can verify them by manipulating the property itself. For example, $(\boldsymbol{a}_1^\top f(\boldsymbol{x}))/(\boldsymbol{a}_2^\top f(\boldsymbol{x})) \leq b$ can be verified as $(\boldsymbol{a}_1 - b\boldsymbol{a}_2)^\top f(\boldsymbol{x}) \leq 0$, and $(\boldsymbol{a}^\top f(\boldsymbol{x}))^{c_1/c_2} \leq b$ for positive integers c_1 and c_2 can be verified as $(\boldsymbol{a}^\top f(x))^{c_1} \leq b^{c_2}$. Examples of functions that cannot be precisely represented are irrational exponents or functions whose computation by these operations demands an infinite series, such as exponentials or trigonometry functions. Moreover, properties that require a very large FDD to represent may be prohibitively slow to verify.

This method also enables verification of a property spanning multiple models. For instance, one might verify that two trained FDTs g and h are sufficiently similar on a domain by constructing $f_1 = g - h$ and $f_2 = h - g$ and verifying that each output of f_1 and f_2 is small on the domain.

We can analyze yet additional properties by augmenting the input space with additional dimensions. This approach could, for instance, compare properties of an FDT at multiple different inputs. Consider the concept of global adversarial robustness defined by [100], an extension of the concept of local adversarial robustness: f is globally robust for given δ and ϵ if, for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^p$ with $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\| \leq \delta$, $\|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)\| \leq \epsilon$. We can verify global robustness of an FDT under any convex norm on a bounded convex domain $D \subset \mathbb{R}^p$ as follows. Let f_1 and f_2 be copies of f with input dimension $2p$, where f_1 operates on the first p input values and f_2 operates on the last p input values. Construct $g = f_1 - f_2$ and, for each class index i, verify $\boldsymbol{e}_i^\top g(\boldsymbol{x}) \leq \epsilon$ on the domain $\{\boldsymbol{x} \in \mathbb{R}^{2p} \mid \boldsymbol{x}_{1:p} \in D,\ \boldsymbol{x}_{p+1:2p} \in D,\ \|\boldsymbol{x}_{1:p} - \boldsymbol{x}_{p+1:2p}\| \leq \delta\}$, where $\boldsymbol{e}_i$ is the indicator of i with 1 at position i and 0 elsewhere. If this holds for all $i \in [q]$, then f is globally robust on the domain. Otherwise, a counterexample to global robustness is the pair of inputs $(\boldsymbol{x}_{1:p}, \boldsymbol{x}_{p+1:2p})$, where $\boldsymbol{x}$ is a counterexample to the verification of g at any i.

3.5.3 Theoretical Results

We provide a theoretical analysis of our verification algorithm and the problems it addresses. Proofs not in the main text are in Appendix B. First, we focus on the algorithm. Let $e_i = u_i - l_i$ be the size of the bounds at node n, a worst-case notion of approximation error. Lemma 3.1 shows how e can be expressed in terms of v, where $v(i, j)$ measures the amount of variation in the approximation at node i coming from node j. T_i is the subtree rooted at node i.

Lemma 3.1. *For any node i, $e_i = \sum_{j \in T_i} v(i, j)$.*

Theorem 3.8 justifies the choice to split the node that maximizes v. Here we assume a simplified worst-case setting where any node-aligned split of the domain results in a reduction of the splitting value range at only that node, which is possible, for example, if $\boldsymbol{a}_i \perp \boldsymbol{a}_j$ for all $i \neq j$ and the original domain's boundaries are either parallel or orthogonal to each $\boldsymbol{a}_i$. In this sense, v indicates the optimal split in the worst case.

Theorem 3.8. *If splitting the domain affects only the chosen node, then choosing to split at the node that maximizes $v(i, \cdot)$, where i is the root, minimizes error summed over the resulting subdomains.*

Theorem 3.9 shows that the refinement reaches an approximation of given precision in finite steps. Because of machine precision, this implies that the verification methodology always eventually terminates and is complete in practice, but it does not necessarily imply efficient convergence; the number of domains increases at each step, so it may take an extreme number of steps before all domains have small enough error. However, we do not need good approximation everywhere, but only good enough to verify the desired property, which is why we prioritize domains with the highest upper

bound. The splitting of the domains also naturally enables parallel computation by allowing several processes to each take from and place onto the queue while simultaneously computing bounds for different domains, which helps make verification of complex models practical.

Theorem 3.9. *For any $\epsilon > 0$, an approximation with error at most ϵ is produced in finite steps.*

Next, we shift focus to the complexity of the problems addressed by this algorithm. Lemma 3.2 shows that an FDT can represent a specific polynomial used in the argument by [149] that quadratic programming is NP-Hard. Lemma 3.3 shows that an FDT can represent any 3-SAT formula. Theorems 3.10 and 3.11 then show that the problems addressed by this algorithm are NP-Complete and NP-Hard, respectively. This means that, similarly to verification of neural networks and standard tree ensembles, we can never expect FDT verification to scale efficiently in the worst case; however, in practice, models trained on real data with meaningful structure rarely result in the worst case, and a good algorithm can finish in reasonable time for a considerable range of problems, as shown in Section 3.5.4.

We prove Theorems 3.10 and 3.11 each using Lemma 3.2, then using Lemma 3.3. The interesting implication of this is that the hardness of optimizing and verifying FDTs can be viewed as resulting from two different challenges: in Lemma 3.2, the constructed FDT represents a single polynomial on the domain, with the challenge arising from a nonconvex function and not combinatorial explosion; in Lemma 3.3, however, the constructed FDT is piecewise-constant, with the challenge arising only from combinatorial explosion. In practice, both challenges may be present. This is as opposed to, for instance, tree ensembles and ReLU networks, which are piecewise-constant and piecewise-linear, respectively, and thus only present the challenge of combinatorial explosion.

Lemma 3.2. *An FDT with $\sigma(z) = \max(0, \min(1, z))$ can be constructed to represent the polynomial $\sum_{i=1}^{k} x_i(x_i - 1) + \sum_{i=1}^{k} x_i s_i$ for $0 \leq x_i \leq 1$, $s_i \in \mathbb{R}$ in $O(k^2)$ time.*

Lemma 3.3. *An FDT with $\sigma(z) = \frac{1}{2}(1 + sign(z))$ can be constructed to represent any 3-SAT formula with m variables and n clauses in $O(mn)$ time.*

Theorem 3.10. *Verification of an FDT $f : \mathbb{R}^p \to \mathbb{R}^q$ as defined in this work, that is, determining whether a property $\boldsymbol{a}^\top f(\boldsymbol{x}) \leq b$ or $\boldsymbol{a}^\top f(\boldsymbol{x}) < b$ holds for all $\boldsymbol{x} \in D$ for convex D, is NP-Complete.*

Proof. We first show that it is NP. A witness to the decision that a property does not hold is a counterexample $\boldsymbol{x}$. It is verified by computing $f(\boldsymbol{x})$ and checking it against the property. The required time is at most linear in the size of f.

We next show that it is NP-Hard via Lemma 3.2 by reduction from the subset sum problem in a slight modification of the argument from [149]. The subset sum

problem is a well-known NP-Complete problem to determine whether some subset of $S = \{s_1, \ldots, s_k\}$, $s_i \in \mathbb{Z}$ sums to $M \in \mathbb{Z}$. Construct an FDT $f(\boldsymbol{x}) = \sum_{i=1}^{k} x_i(x_i - 1) + \sum_{i=1}^{k} x_i s_i$ on $0 \leq x_i \leq 1$ as in Lemma 3.2. Next, let D be the domain described by $0 \leq x_i \leq 1$, $\sum_{i=1}^{k} x_i s_i \leq M$ and check whether $f(\boldsymbol{x}) < M$ for all $\boldsymbol{x} D$. If the subset sum problem has a solution, then a counterexample $\boldsymbol{x}$ exists with $x_i = 1$ if s_i is in the sum and $x_i = 0$ otherwise; likewise, since $x_i(x_i - 1) < 0$ when x_i is not 0 or 1, any $\boldsymbol{x}$ returned as a counterexample by the verification must consist of integers and therefore represents a solution to subset sum. We thus conclude that FDT verification is NP-Hard by polynomial-time reduction from subset sum.

We show again that it is NP-Hard, now via Lemma 3.3, by reduction from the 3-SAT satisfiability problem. Construct an FDT to represent the 3-SAT formula as in Lemma 3.3. Next, let D be $[-1, 1]^m$, with m the number of variables, and verify $-f(\boldsymbol{x}) < 0$. If this holds, then there is no $\boldsymbol{x} \in D$ such that $f(\boldsymbol{x}) = 0$, so by Lemma 3.3, the 3-SAT formula is unsatisfiable; otherwise, the counterexample generates a satisfying assignment as described in Lemma 3.3. Thus FDT verification is NP-Hard by polynomial-time reduction from 3-SAT □

Theorem 3.11. *Maximizing an FDT on convex domain is NP-Hard.*

Proof. We first show reduction from the subset sub problem via Lemma 3.2. The subset sum problem reduces to maximizing $\sum_{i=1}^{k} x_i(x_i - 1) + \sum_{i=1}^{k} x_i s_i$ subject to $\sum_{i=1}^{k} x_i s_i \leq M$, $0 \leq x_i \leq 1$ with $s_i \in \mathbb{Z}$, $M \in \mathbb{Z}$ as shown by [149]. By Lemma 3.2, an FDT can efficiently represent this function on this domain, so maximizing an FDT is also NP-Hard by polynomial-time reduction from subset sum.

We next show reduction from 3-SAT via Lemma 3.3. Construct an FDT to represent the 3-SAT formula as in Lemma 3.3. Next, let $\mathcal{D}$ be $[-1, 1]^m$, with m the number of variables, and maximize $-f(\boldsymbol{x})$ on D. If the maximum is less than 0, then there is no $\boldsymbol{x} \in D$ such that $f(\boldsymbol{x}) = 0$, so by Lemma 3.3, the 3-SAT formula is unsatisfiable; otherwise, the maximum value is 0 and the maximizing input generates a satisfying assignment as described in the proof of Lemma 3.3. Thus FDT maximization is NP-Hard by polynomial-time reduction from 3-SAT. □

3.5.4 Experiments

We present two sets of experiments: the first benchmarks our method and compares against alternatives to find Minimum Adversarial Perturbation (MAP) at test points on a selection of data sets from OpenML; the second tests global adversarial robustness for the same FDTs. Since scalability is the main limitation of verification, we are mainly interested in measuring the time taken to complete the tasks.

The OpenML data sets used for these tests are chosen to include a variety of number of features and labels. For each, we randomly select 90% of the samples to train FDTs with both sigmoid and linear splitting functions. The linear split here is defined as $\sigma(z) = \max(0, \min(1, (z + 1)/2))$ so that it transitions from 0 to 1 on

OpenML data set				FDT size		acc. (sig)		acc. (linear)	
ID	name	feat	label	depth	nodes	train	test	train	test
40685	shuttle	9	7	4	13	99.83	99.81	99.85	99.83
1036	sylva_agnostic	216	2	7	25	99.85	99.44	99.80	99.51
1497	wall-robot-nav	24	4	8	33	97.72	94.50	98.15	95.96
1462	banknote-auth	4	2	6	39	99.84	99.27	99.43	100.0
41169	helena	27	100	7	49	28.12	26.45	29.48	27.72
1120	MagicTelescope	11	2	7	57	87.03	88.16	88.00	88.64
182	satimage	36	6	11	107	93.28	90.65	93.17	89.25
40499	texture	40	11	14	163	99.76	100.0	99.80	100.0
375	JapaneseVowels	14	9	12	251	99.06	97.99	99.01	97.79
1479	hill-valley	100	2	59	305	65.78	59.50	70.18	66.12

Table 3.3: Information about data sets and their corresponding FDTs.

$[-1, 1]$. The other 10% of the samples are test data from which we randomly draw 100 points for the minimum adversarial perturbations. Each verification is run with a 3600 second timeout. Information about each data set and the corresponding FDTs is displayed in Table 3.3. To help associate results with FDT size, all tables and figures order the data sets by the number of nodes in the corresponding FDT.

We use CVXPY [48, 3] for convex optimization, which adds some overhead to the time to solve, but acts as a convenient interface to many different convex optimization solvers. For these experiments, we choose Gurobi, which is robust to solver failures on the more complex data sets, but which we find to take about twice as long as other solvers such as ECOS for this application. All experiments use $\epsilon = 10^{-2}$ for dynamic pruning. We train FDTs by first initializing from a fitted `sklearn.tree.DecisionTreeClassifier` [138] with `ccp_alpha=1e-3` and then training using PyTorch's Adam optimizer with batch size 500 and learning rate 10^{-3} for 5000 epochs. No special effort was made to individually tune these parameters for each data set since achieving the best possible prediction performance is not important to these experiments. However, we do report prediction accuracy in Table 3.3 to show that the FDTs are indeed resonably fitting the data. For the sake of benchmarking times, we did not parallelize the verification algorithm.

Our verification code is available on GitHub[3] along with Jupyter notebooks that demonstrate its features and run the experiments. We also include the saved FDT models and the test points used for experiments, with the data normalized to mean 0 and standard deviation 1, as well as the full results of the verifications run for the experiments.

[3] https://github.com/autonlab/fdt_verification

data set	Ours (sigmoid splitting)			Adapted from ReluVal (linear splitting)				
	failed	all cases		failed	all cases		Z3 succeeded	
	of 100	mean	median	of 100	mean	median	mean	median
shuttle	0	4.65	4.09	5	299.92	31.48	331.48	26.66
sylva_agnostic	3	256.76	61.27	97	3520.16	3600.00	-	-
wall-robot-nav	0	12.40	8.20	33	1396.74	315.89	15.21	11.39
banknote-auth	0	20.71	14.03	3	167.43	26.17	8.89	6.91
helena	0	112.59	90.08	42	1997.78	2369.87	988.37	508.50
MagicTelescope	2	256.34	54.30	43	1845.36	1836.69	604.36	6.57
satimage	4	224.08	31.72	91	3307.17	3600.00	1211.21	21.58
texture	2	385.24	163.96	92	3352.58	3600.00	-	-
JapaneseVowels	0	186.28	33.87	61	2364.41	3600.00	137.73	48.89
hill-valley	2	175.04	46.64	99	3564.18	3600.00	-	-

data set	Ours (linear splitting)					Z3 (linear splitting)				
	failed	all cases		Z3 succeeded		failed	all cases		Z3 succeeded	
	of 100	mean	median	mean	median	of 100	mean	median	mean	median
shuttle	0	4.65	4.44	4.50	4.29	12	720.82	242.48	328.20	174.32
sylva_agnostic	2	321.59	102.80	-	-	100	3600.00	3600.00	-	-
wall-robot-nav	0	7.53	4.51	2.64	2.66	74	2669.90	3600.00	22.69	10.88
banknote-auth	0	43.02	16.05	3.54	2.66	94	3416.31	3600.00	538.42	5.01
helena	1	129.13	76.42	56.62	62.92	52	2358.17	3600.00	1012.86	1042.01
MagicTelescope	0	29.64	10.50	1.13	1.08	94	3387.87	3600.00	64.27	7.78
satimage	0	146.32	48.24	4.57	4.59	97	3501.48	3600.00	315.99	57.04
texture	3	462.93	196.27	-	-	100	3600.00	3600.00	-	-
JapaneseVowels	0	50.28	25.43	9.39	7.80	96	3463.40	3600.00	185.03	177.94
hill-valley	0	35.76	14.74	-	-	100	3597.93	3600.00	-	-

Table 3.4: Time in seconds and and number of failed cases for the MAP experiments.

Minimum Adversarial Perturbation

Minimum Adversarial Perturbation (MAP) is the problem of finding the minimal perturbation of a data point $\boldsymbol{x}$ to change the model's prediction

$$\arg\min_{\boldsymbol{x}'} \|\boldsymbol{x} - \boldsymbol{x}'\|_p \text{ s.t. } \arg\max f(\boldsymbol{x}) \neq \arg\max f(\boldsymbol{x}')$$

where p is usually 1, 2, or ∞. Here we use $p = \infty$, but our algorithm can verify any of these because they are all convex. We choose MAP for benchmarking our algorithm because it is commonly studied and generally applicable, and because it eliminates the arbitrary choice of radius for the strictly easier problem of local adversarial robustness, that is, determining whether a perturbation of limited magnitude can change the prediction.

Minimum adversarial perturbation is found by verification of local adversarial robustness combined with binary search over the radius. Here we search until a tolerance of 10^{-3} is reached. For each radius, local robustness is checked for each other label; that is, if $\arg\max f(\boldsymbol{x}) = i$, then to check local robustness at $\boldsymbol{x}$ with radius r, we verify $(\boldsymbol{e}_j - \boldsymbol{e}_i)^\top f(\boldsymbol{x}') < 0$ for $\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty \leq r$ for every $j \neq i$, where $\boldsymbol{e}_i$ is the indicator with 1 at position i and 0 elsewhere, and likewise for $\boldsymbol{e}_j$.

We benchmark against two alternatives. First, we use a variant of our algorithm that replaces the choice of domain splitting with the strategy from ReluVal [181]. In particular, this bisects along the feature with the highest smear value, which is based

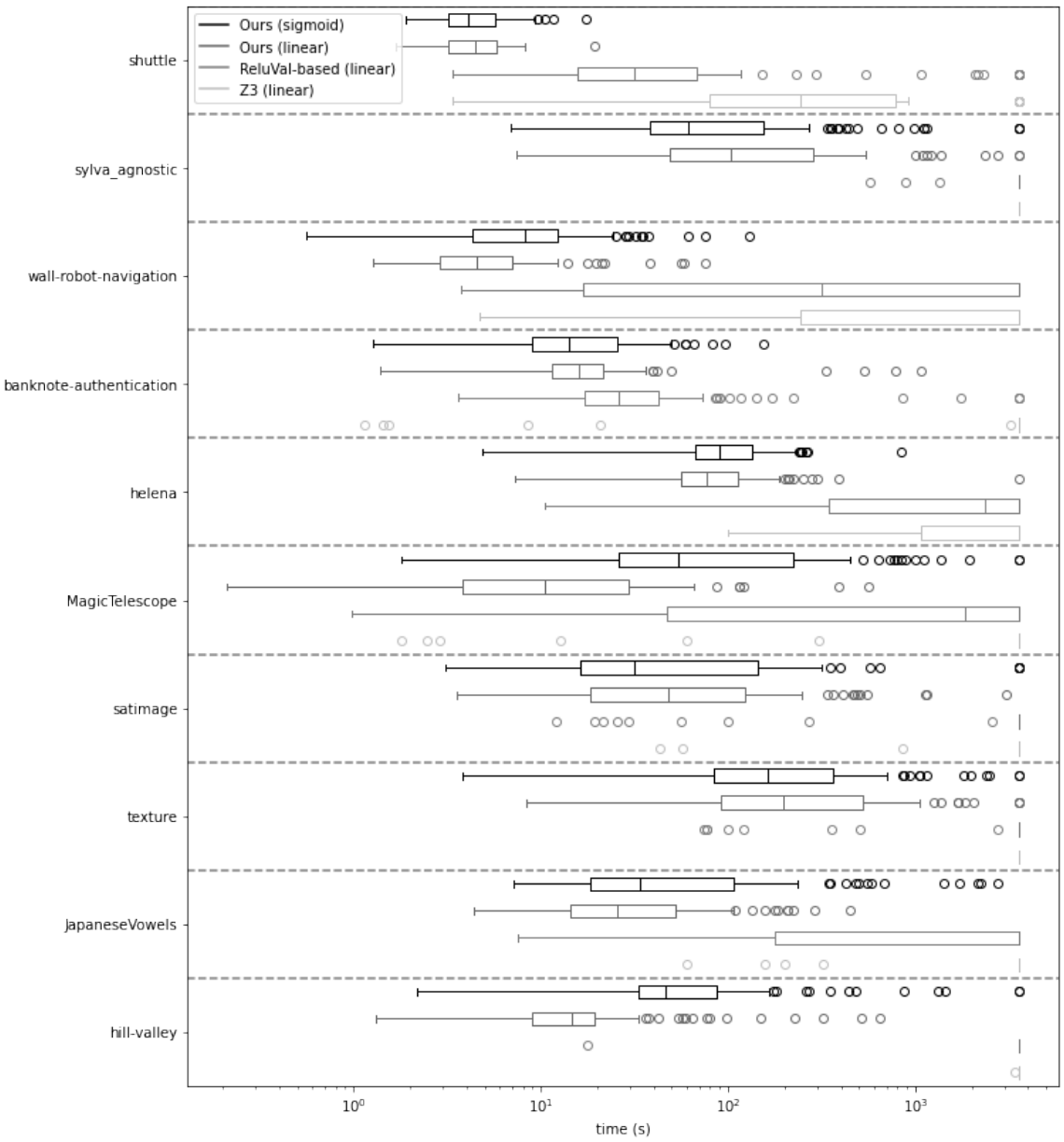


Figure 3.14: Distributions of time in seconds to find minimum adversarial perturbations. Timeouts are shown as the time limit, 3600 seconds.

on an upper bound of the magnitude of the gradient. As a result, it is limited to hyperrectangle domains, and for simplicity we implement the gradient bounding only for linear splitting functions, similarly to how [181] considers only ReLU activations. Second, we compare against the SMT solver Z3, a tool that is generally limited to linear arithmetic. Therefore, we apply it only to models with linear splitting functions, but even in this case, it can sometimes fail due to multiplications of these functions.

The results are shown in Table 3.4, which includes the number of failed cases as well as the mean and median time to find MAP for each data set and corresponding FDTs. All failed cases are due to timeouts except for one case where Z3 failed before the timeout on hill-valley. For comparison with Z3, the statistics of the subpopulation of samples where Z3 succeeds are also given for the FDTs with linear splitting; the "all cases" columns consider timeouts to have a value of 3600 seconds (the time limit). Figure 3.14 shows the distributions of the times. Based on the results, the time taken by our method appears to depend on some combination of the number of features (FDT inputs), number of labels (FDT outputs), and the size of the FDT. As is often the case in machine learning model verification, the majority of cases take a reasonable amount of time to complete, but a few cases take notably longer, with there being at least one timeout on many of the data sets. The distributions of times appear to be approximately symmetric on the logarithmic scale. In most cases, the times to verify FDTs with sigmoid splitting and with linear splitting are similar.

Across all data sets, the variant of our algorithm using the splitting approach from ReluVal [181] takes substantially longer than our proposed method, and Z3 takes substantially longer than that. Because of the limitations of SMT solvers, Z3 sometimes fails even on the simplest FDTs and data sets, and it cannot verify FDTs with sigmoid splitting at all. Moreover, the rightmost columns of Table 3.4 show that, even when we compare only cases where Z3 does succeed, our method is consistently faster.

Global Adversarial Robustness

Global adversarial robustness is a generalization of the local robustness concept; a model is globally robust if similar inputs always produce similar outputs. Specifically, given δ and ϵ, f is globally robust if and only if, for all $\boldsymbol{x}$, $\boldsymbol{x}'$ in some input domain, $\|\boldsymbol{x} - \boldsymbol{x}'\|_p \leq \delta$ implies $\|f(\boldsymbol{x}) - f(\boldsymbol{x}')\|_p \leq \epsilon$. Note that, unlike local robustness, this considers the raw predicted scores rather than the label prediction; otherwise global robustness over the entire model domain would not be possible because there would always be violations near the decision boundary. For experiments, we consider the entire domain of the model and again use $p = \infty$. Since considering two inputs effectively doubles the dimension of the search space, and since it considers every possible input to the model, this is a far more difficult problem than local robustness. It is limited to very small neural networks [101] and has only recently been verified for realistically sized tree ensembles [74]. The way of encoding and verifying global

shuttle

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	1858	50	10	5
10^{-2}	12	170	174	13
10^{-1}	4	4	6	10
10^{0}	5	7	14	24

MagicTelescope

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	3055	22
10^{-2}	-	-	-	26
10^{-1}	-	-	-	2340
10^{0}	135	163	290	539

sylva_agnostic

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	909	56
10^{-2}	-	-	1492	45
10^{-1}	852	202	-	-
10^{0}	193	253	419	765

satimage

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	-	350
10^{-2}	-	-	-	550
10^{-1}	570	2905	-	-
10^{0}	155	180	264	391

wall-robot-navigation

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	-	585
10^{-2}	-	-	-	-
10^{-1}	-	-	-	-
10^{0}	410	410	427	457

texture

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	-	1029
10^{-2}	-	-	-	1379
10^{-1}	-	-	-	-
10^{0}	302	366	454	553

banknote-authentication

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	1023	202	57
10^{-2}	-	1253	229	57
10^{-1}	-	-	1718	156
10^{0}	12	12	12	40

JapaneseVowels

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	-	1730
10^{-2}	-	-	-	-
10^{-1}	182	213	274	357
10^{0}	181	217	273	348

helena

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	85	68	68	68
10^{-2}	71	67	67	70
10^{-1}	69	67	66	66
10^{0}	69	66	67	67

hill-valley

δ, ϵ	0.2	0.4	0.6	0.8
10^{-3}	-	-	-	-
10^{-2}	-	-	-	-
10^{-1}	-	-	-	-
10^{0}	1161	1609	2162	3046

Table 3.5: Time in seconds for global adversarial robustness tests. White cells indicate a result of robust, lightly shaded cells not robust, and darkly shaded cells timeout.

robustness in our system is explained in the Additional Properties subsection of 3.5.2.

Results are shown in Table 3.5. Here, since we do not compare to Z3, we test our method only on FDTs with sigmoid splitting. As global robustness is a more difficult problem, there are many cases where the 3600 second timeout is reached, especially for small δ and ϵ and on the more complex models. Unsurprisingly, it also seems that there is a positive association between the time required to verify and the closeness to the boundary between robust and not robust in (δ, ϵ)-space. This concretely manifests in Table 3.5 in that the dark gray is always on the border between white and light gray. It is also unsurprising that, without a training algorithm

that has global robustness as an objective, of which none exist to our knowledge, a strong degree of global adversarial robustness (robustness with high δ and low ϵ) is not seen in complex models. Regardless, verification tools such as ours are still useful to identify and isolate violations of global adversarial robustness.

Notably, the results suggest that, for a given model, the difficulty of verifying global robustness is not closely related to the difficulty of verifying local robustness. For example, while local robustness was fast to verify for wall-robot-navigation, many tests for global robustness timed out; this may be because complex regions far from the data are considered by global, but not local robustness. For helena, local robustness was slow to verify, but global robustness was relatively fast; here the very large number of labels, combined with the relatively low accuracy, may indicate that the output range for any given class's score is generally small, meaning that the values of ϵ used here are large by comparison, making global robustness verification easier. This brings up the concern that global robustness can be achieved without affecting accuracy by simply shrinking the model's output range, that is, making its predictions less confident, which is usually not desirable. This suggests that perhaps a more useful definition of global robustness can be devised.

3.5.5 Discussion

While the past decade has been fruitful in the development of verification methodologies for neural networks and tree ensembles, this is, to the best of our knowledge, the algorithm for the verification of FDTs. We show that this is a complex problem resulting both from combinatorial explosion and the ability of FDTs to directly represent non-convex functions, unlike, for example, ReLU networks and tree ensembles, where the complexity of verification results from combinatorial challenges alone. Thus, despite the similarities to both of the aforementioned model classes, FDTs with their unique structure present additional challenges for verification, and therefore have motivated an approach that, while it fits into the paradigm of abstraction-refinement, differs in its formulation and theory from previous approaches. We are optimistic that our framework may lay the groundwork for verification of other kinds of models with similar challenges.

The main limitation, as in all machine learning verification methodologies, is in scalability with the complexity of the model. Larger input size necessitates more constraints and slows the convex optimization, larger output size causes properties like robustness to require more individual verification calls, and trees with more nodes take longer to traverse and potentially more iterations to verify. For various reasons, it is difficult to compare the scalability of our approach with the verification of other model types, such as neural networks and tree ensembles. One difficulty is that all such problems are NP-Complete, so all solutions are inherently not scalable in the worst case, but most achieve reasonable speed in practice by using smart heuristics, and because models trained on real data are usually more structured than the worst case.

This means comparisons are necessarily empirical, which is complicated by the wide variety of constantly evolving verification methodologies as well as differences in the models, data, implementations, hardware, choice of benchmarks, and more. There is no universal standard for benchmarks; local robustness problems are standard, but the choice of norm and radius are not. This is one of the reasons we chose to benchmark using minimum adversarial perturbation: it eliminates the choice of radius and is strictly more difficult than checking robustness at a given radius. Similarly, there is a set of safety properties in an aircraft collision avoidance setting first used by [100] that have become standard benchmarks for verification of neural networks, but only the neural networks themselves and not the training data are publicly available, so this benchmark cannot be used for tree ensemble or FDT verification. Additionally, there is always a tradeoff of model complexity (which is related to accuracy) vs. verifiability; for example, neural network verification scales poorly with the size of the network, with many algorithms limiting architecture choices, tree ensemble verification scales poorly with the number of trees, and our approach to FDT verification scales poorly with the dimensionality of the problem, which depends on the number of features and number and level of interdependence of tree nodes. Design choices come into play; for instance, we initialized our FDTs from fitted crisp trees for simplicity, but since oblique trees can be smaller than conventional trees, this initialization results in a larger FDT than necessary for good performance. Some models simply require less complexity than others to represent different kinds of data. Because of all this, it is difficult to perform a comprehensive, fair, and meaningful cross-model comparison of the scalability of verification. For these reasons, a general analysis of the scalability of verifying different model classes and the choice of which is best for a given application is left to future work.

Other than scalability, a notable limitation of our algorithm is the kinds of properties it can analyze. First, as described in the Additonal Properties subsection of 3.5.2, we cannot represent certain properties using FDDs. We are not aware of any commonly studied machine learning safety properties that cannot be verified using our system, but such cases could potentially arise among domain-specific safety properties. We are also limited to analysis on domains on which some solver can efficiently optimize a linear function[4], though more complex domains can be verified by covering them with convex domains. Additionally, performing optimization on small domains resulting from successive splitting can cause problems for solvers that are sensitive to numerical stability.

We also make a few points in favor of FDTs and our algorithm. We are able to find minimum adversarial perturbations in reasonable time on a wide range of data and model complexities, and we are among the first to show limited verification of global adversarial robustness on realistic models. While many verification methodologies rely on strict model design assumptions, such as ReLU activations in neural network

[4]See section "Choosing a solver" at https://www.cvxpy.org/tutorial/advanced/index.html for examples.

verification, we require no such assumptions. Our domain splitting and queueing approach is naturally well-suited to highly parallel computation. The necessity of solving convex optimization problems makes massive parallelization through dedicated hardware such as GPUs difficult, but such optimizations may still be feasible; in particular, if the domain boundaries and node splits are pairwise parallel or orthogonal, then convex optimization is not needed at all and GPU usage is straightforward. We support non-linear convex domains, enabling constraints using functions such as the L2 norm. The refinement process can be stopped at any point and, even if the property is not determined, information can be gleaned from the current bounds for approximate but extremely fast solutions. It also offers utility by finding regions of counterexamples rather than a single point, and it can be run longer to expand that region; the splitting partitions the original domain into regions that are fully safe, fully unsafe, or a mix of safe and unsafe, and by continuing to split the "mix" regions, it can, given enough time, produce arbitrarily precise descriptions of the true safe and unsafe regions. And finally, FDTs themselves offer benefits. They are trainable end-to-end with gradients, their architecture can be designed automatically, they are compact compared to tree ensembles, and their hierarchical partitioning approach is relatively interpretable compared to neural networks.

Finally, we note that, while the formulation of FDTs we verify in this section is quite general, it unfortunately does not include KDDTs due to their unique interdependence of splitting functions, and an adaptation of this algorithm for KDDTs remains elusive. While the inherent certifiable adversarial robustness of KDDTs discussed in Section 3.4 covers one very popular use case of verification, verification algorithms can be applied to a much wider class of safety properties, so we hope that this work may be an important step toward verification of not only KDDTs, but for smoothed models in general.

Chapter 4

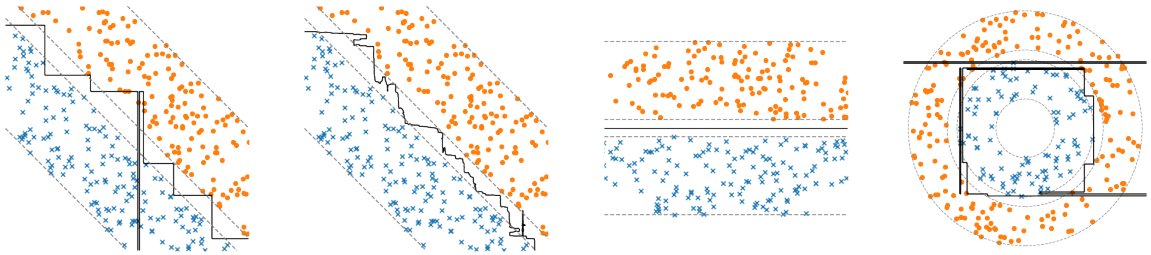
Interpretability

One of the foremost trust-related aspects of a model is its ability to be understood by users, particularly those without expertise in machine learning. For example, predictors that assist medical diagnosis are more useful and more likely to be appropriately trusted if the reasoning behind their predictions is easily understood by medical personnel. Beyond trust, interpretable methods can aid in diagnosis of issues with the model and data and sometimes even lead to insights about the domain of application.

We argue that we should strive for the ideal of intrinsic interpretability, that is, the transparency of the model itself, as opposed to post-hoc explainability that only produces an interpretable approximation of the model through a separate process. While this may be more difficult to achieve, we have an ethical imperative as developers of machine learning models and methods to be as transparent as possible. The goal of interpretable methods is not to gain the blind trust of users through an impression of understanding; it is to elucidate the model in all its merits and flaws, earn trust where it is deserved and suspicion where it is warranted, and illuminate the path toward models more deserving of trust. After all, as the famous saying goes, all models are wrong, but some are useful.

Interpretability is highly subjective and application-dependent by nature, and there is no universal standard of what makes a system highly interpretable. However, there are some generally agreed-upon rules of thumb; for example, smaller models are considered more interpretable than structurally similar but larger ones, sparse parametric expressions are considered more interpretable than dense ones, and rule-based logical representations such as decision trees are considered more interpretable than complex arithmetic "black box" models such as neural networks.

Decision trees in particular are often used as the quintessential example of intrinsically interpretable ML; they can be interpreted holistically (also called *global interpretation*) as a hierarchical partitioning of the input based on simple decision rules, and individual predictions can be interpreted (also called called *local interpretation*) as a series of such rules on a path from the tree's root to a leaf. Conventionally, these decision rules consist of a simple feature value and threshold, but many other



(a) A DT is complex and generalizes poorly.

(b) A RF is very complex and generalizes better, but not perfectly.

(c) After rotation, a DT is simple and generalizes perfectly.

(d) Other data requires a different transformation.

Figure 4.1: A toy example to motivate learning feature transformations. Points are sampled uniformly in the dashed bands.

decision rules are possible. For example, rules of the form $\boldsymbol{a}^\top \boldsymbol{x} < b$ for some $\boldsymbol{a}$, b are called oblique splits, and the trees using them Oblique Decision Trees (ODTs).

In practice, however, there are major barriers to the practical application of decision trees as interpretable models. First, as discussed throughout Chapter 3, conventional decision trees fall short in various aspects of robustness, most importantly in their generalization to yet-unseen data; while this can be somewhat alleviated by pruning, the most effective and popular solution is to use large ensembles with strategies such as bagging as in Random Forests [26], random thresholds as in ExtraTrees [68], or boosting as in XGBoost [35]. In addition, decision trees often grow quite deep and use many decision rules for any given prediction, even for extremely simple data. Together, these make both global and local interpretation much more difficult.

While we have established KDDTs as an effective means to improve the robustness of single-tree models, they generally do not make it possible to meaningfully reduce the size of trees, and they sometimes even necessitate larger trees. We posit that the tree size barrier results from greedy construction and simple decision rules, of which the key limitation is the latter; even if we replace greedy construction with a perfect tree learner, simple distributions can nonetheless require an arbitrarily large axis-aligned tree to fit. However, after a transformation of that distribution, even a greedy algorithm can produce a small tree, as shown in Section 4.3. Figure 4.1 gives a motivating example of a case where trees do poorly as interpretable learners on simple data, but succeed after a transformation. This sensitivity to the feature representation is also related to other limitations of tree-based models, such as their dependence on features hand-crafted using expert knowledge to achieve good performance on some problems, and their overall weakness on other problems, such as image classification.

This motivates the idea of learning transformations of the input features, or equivalently, more expressive decision rules such as oblique splits. This can reduce the size of trees, and by choosing the type of transformation and its regularization during

learning, once can control the tradeoff of complexity of the rules and complexity of the tree structure. It can also increase the performance of single-tree models and make them performant in modalities where trees are not typically well-suited, such as time series and image classification, by essentially transforming them into tabular data. These modalities are ubiquitous in critical domains such as clinical decision support where interpretability is highly motivated.

However, the problem of learning more complex splits is not easy; axis-aligned decision rule learning, as in the popular CART algorithm [25], is only practical because the search is restricted to a finite set of candidate rules that is efficiently exhaustively searchable. Finding even optimal oblique splits, which are among the simplest decision rules, is NP-Hard [85], and even oblique splits may not be considered interpretable unless they are sparse, that is, having few nonzero coefficients.

Nonetheless, methods have been developed to learn such trees, either by heuristically choosing splits in a tree construction process, or by constructing an axis-aligned tree and then optimizing the parameters of the splits (see Section 4.1.1 for examples). However, the former suffer from greedy, suboptimal choice of split, and the latter suffer from keeping the original structure of the tree. These are also usually restricted to a certain kind of split, such as oblique splits.

Here we propose a novel, flexible framework that is the first to alternate optimization of feature transformation parameters with complete regrowth of the tree, which is made possible by the efficiency, stability, and inherent differentiability of KDDTs. Refitting the tree completely throughout training reshapes the tree and moves thresholds as needed; it also maintains the property that each splitting rule is maximally informative given the available features and data. Together these result in smaller, more interpretable trees for a given level of performance. Moreover, this framework simplifies the application of decision trees to modalities such as images and time series, where transformation into a tabular format is required for the effective application of tree-based models.

This section covers work originally published in [75], as well as additional methods and demonstrations for image and time series data.

4.1 Related Work

4.1.1 Decision Trees as Interpretable Models

Decision trees have a long history of being chosen for their interpretability. In this section, we cover various work developing the use of trees toward interpretable modeling with an emphasis on those most similar to this work.

There is extensive work using decision trees extracted from more complex models either to entirely replace the model, or as an interpretable proxy. Examples include [40, 152, 175, 11, 133, 148, 82, 21].

Other work attempts to improve the interpretability of decision trees themselves by introducing new decision rules. These rules are more complex than the axis-aligned threshold rules, but compensate by improving performance of single-tree models while reducing their size. For example, [20] uses a branch-and-bound algorithm to search for bivariate splits. [54, 55, 73] use parallel coordinates, which incorporate information from multiple data features while being easy to visualize. [47] uses bilevel optimization to find nonlinear splits for growing trees.

The most common augmented decision rule is linear splits, that is, splitting using arbitrary hyperplanes rather than axis-aligned hyperplanes. These are often called oblique splits, and the trees that use them Oblique Decision Trees (ODTs). Algorithms for growing ODTs are first proposed along with CART [25]. [130] later discusses the difficulty of finding optimal linear splits due to local minima and proposes a heuristic using randomization and deterministic hill climbing.

Another approach is to grow an axis-aligned tree, fix the structure, then treat it as a parametric model in order to learn coefficients for linear splits. This can be made possible using Fuzzy Decision Trees (FDTs), wherein a decision is a weighted combination of both subtrees rather than wholly one or the other, to make the tree differentiable. In the case of ODTs, a splitting function such as sigmoid is used to map a linear combination of feature values to the weight of each subtree. [163] is the first to train globally optimized trees in this way. Later, [105] proposes ensembles of such trees, both as standalone models and as the final layer of a neural architecture, [121] presents various improvements such as balance and sparsity regularizations and dynamic learning rate adjustments, and [155] applies them to reinforcement learning.

Tree Alternating Optimization (TAO) is a recent and successful ODT learning algorithm that initializes similarly, but unlike FDTs, uses a linear discriminator at each node, such as logistic regression or linear SVM, then alternates optimization over each depth level of the tree. It can be used with a sparsity penalty to produce sparse ODTs, that is, where many or most of the coefficients are 0, making linear splits more interpretable. While it is shown that this alternating optimization never causes the loss to increase, there are no convergence or approximation guarantees. It was initially proposed for classification [28], then extended in various ways, including regression ensembles [201]; boosted ensembles [65]; clustering [64]; semi-supervised learning [202]; interpretable image classification [158]; interpretable natural language processing [87]; and as a tool for understanding parts of neural networks [86].

Our approach is different from any existing work in that we consider a broader scope of feature types, but more importantly, in that we continually and completely refit the decision tree throughout the learning process. This allows dynamic restructuring of the tree as the input features change, but requires an efficient from-scratch learning algorithm for differentiable decision trees; and KDDTs are ideal for this purpose. In addition to efficiency and differentiability, the kernel representation has the benefit of avoiding redundant splitting that may occur in other FDT formalisms because repeating similar splits can make the division less "fuzzy" and therefore reduce

loss by inflating the tree size. Our approach is also unique in that we allow the re-use of the same transformed feature throughout the tree, reducing the number of separate concepts needed to understand the model. Moreover, our approach uniquely weights the regularization of each feature by its usage within the tree on the training data, resulting in a more balanced enforcement of sparsity throughout the tree.

4.1.2 Interpretable Time Series Classification

For an overview of methods for time series classification, we refer the reader to [185, 9, 95].

Time series shapelets [196] are the most prominent approach for interpretable featurization of time series. Countless works have proposed or made use of variations of shapelets, so we will cover only a selection of highly-cited and relevant works. Shapelets are subsequences of time series; given a shapelet and a time series input, a feature value is computed as the distance between the shapelet and the best-matching subsequence of the input time series. [196] first used shapelets to construct a decision tree in the traditional top-down greedy growth method by, at each split, exhaustively searching over all possible combinations of shapelets mined from the training data and distance thresholds values. In large data sets with long time series, there may be a very large number of shapelets and thresholds to search over, so several subsequent works improve the efficiency of the search process. For instance, [142] apply a heuristic based on symbolic representation and random projection to improve scalability without impacting performance. Others, starting with [92], learn discriminative shapelets independently of the model, calling it a shapelet transform, then fit various models to achieve better performance than a decision tree. Shapelets can also be directly learned as parametric components of a differentiable model, such as logistic regression [80] or neural network [126], using gradient-based optimization of the loss, eliminating the need for exhaustive search and improving predictive performance. We also learn parametric shapelets (and other interpretable feature types) using gradient-based optimization, but as components of differentiable decision trees in order to improve performance without giving up the interpretability and practical benefits of single-tree models. To our knowledge, we are the first to do so.

Multivariate shapelets, designed for multivariate or multi-channel time series, pose greater challenges than univariate shapelets [111]. [69] first apply multivariate shapelets in trees using a distance threshold for each channel. [29] train one shapelet tree on each channel, then combine them into a voting ensemble. [22] more broadly consider notions of independent (separate features for each time series channel), multidimensional dependent (with aligned time), and multidimensional independent (with independent time) shapelets for general shapelet transforms, finding that multidimensional dependent performs best on benchmarks. [126] use parameterized multidimensional dependent shapelets as a neural network layer. We apply the same style of shapelet as input to a differentiable decision tree, but with additional

weighting parameters in the distance calculation that allow shapelets to be sparse and ignore irrelevant channels, time spans, etc.

4.1.3 Interpretable Image Classification

Image classification is dominated by neural networks, which have low inherent interpretability. A large amount of research has been dedicated to explaining the predictions of neural networks. There are countless surveys; for recent examples, see [200, 57, 151, 147].

A popular paradigm for interpretable image classification is image prototypes, a template-matching unit used in neural networks for image classification inspired by traditional approaches such as "bag-of-visual-words". As a sliding-window best-match method, they can be understood as a two-dimensional analog of time series shapelets. However, unlike with shapelets, the image is not compared with a template directly; instead, both are processed by a convolutional neural network, which is itself not interpretable, before the comparison is made in the resulting latent space. Image prototypes are originally proposed by [34], who incorporate them as part of a neural architecture called prototypical part network (ProtoPNet). The prototypes are identical to patches of images in the training set, but unlike traditional methods, they select the patches as part of the network. Several subsequent works make adjustments to the architecture and prototype selection [178, 146, 145, 192, 131]. One relevant example is ProtoTree [132], which incorporates image prototypes into a fixed-structure soft decision tree trained by gradient-based optimization.

4.2 Methods

4.2.1 Alternating Optimization of Trees and Features

Given training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, we aim to learn parameters $\theta \in \mathbb{R}^d$ of feature transformation $g_\theta : \mathbb{R}^p \to \mathbb{R}^{p'}$ and a decision tree $f : \mathbb{R}^{p'} \to \mathbb{R}^q$ to minimize empirical risk for the composed predictive model $f \circ g_\theta$. The tree is typically subject to constraints such as a max depth, minimum sample weight per node, or cost-complexity pruning.

Viewed within this framework, the prevailing approach in prior work is equivalent to the following process: (1) initialize θ such that g_θ is identity; (2) with θ fixed, optimize f using a greedy tree growth strategy such as CART; (3) with f fixed, optimize θ using gradient-based optimization. Some methods use a growing and/or pruning heuristic during (3) so that the structure of the tree may change.

Instead, we propose an approach inspired by alternating optimization, a common strategy to pragmatically solve difficult optimization problems, whereby we alternate fitting a differentiable tree as in (2) with one or more gradient-based updates to θ as in (3). In this case, alternating allows us to use different optimization strategies for

the feature transform parameters θ and the non-parametric decision tree f. This is summarized in Algorithm 3.

Algorithm 3 Learn a decision tree with transformed features.

Input: Training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, feature transformation architecture g_θ with parameters $\theta \in \Theta$, loss function ℓ, penalty function $p : \Theta \to \mathbb{R}_{\geq 0}$.
Output: Learned feature transformer g_θ and tree f.

```
Initialize θ                                          ▷ See Section 4.2.3 for examples
while stopping conditions are not met do              ▷ E.g. number of epochs
    Fit a KDDT f to minimize empirical risk (1/n) Σ_i ℓ(f(g_θ(x_i)), y_i)
    for some number of iterations do
        Update θ using (1/n) Σ_i ∇ℓ(f(g_θ(x_i)), y_i) + ∇p(θ)    ▷ E.g. minibatch SGD
```

We use KDDTs as the differentiable tree for this learning procedure. They are well-suited because they are fit as inherently differentiable models, rather than fitting a tree, then making it differentiable like many prior methods. Moreover, their efficient fitting is crucial since this process involves frequent repeated fitting. At the end, we can use the smoothed or unsmoothed KDDT for inference. The latter has better interpretability when predictions would otherwise take multiple paths, but can result in a small drop in performance.

Trees are fitted using an information gain based on an impurity measure, so the same impurity should be used as the loss function for feature learning. For classification, we use the Gini impurity; as shown by Theorem 2.2, this is equivalent to mean squared error loss. Likewise, if using entropy impurity, it is equivalent to cross-entropy loss, as shown by Theorem 2.3.

4.2.2 Kernel choice

In this context, the kernel is mainly used as a smoother to make the tree differentiable, so the choice of shape is not crucial. A good choice is the box kernel, which is efficient for fitting and enables sparse tree node membership without truncation. However, the *size* of the kernel relative to the scale of the input is important; we adopt the term *bandwidth* from kernel density estimation to describe kernel scaling. If the bandwidth is too small, a KDDT is very close to a conventional decision tree, and the loss gradient is either zero or highly unstable because it results from very few data near split boundaries; if the bandwidth is too large, however, splits are very soft, the tree grows large, the discriminative power is weak, and computation is slow due to dense membership of data in leaves. To control the bandwidth, we use one of two strategies: (1) design the feature transform so that the output range is limited, for example, to $[0, 1]$, and choose an appropriate kernel bandwidth; or (2) use a regularization on the feature transform parameters to automatically scale the feature outputs appropriately relative to the kernel. We generally find the latter approach to be more effective and

describe the use of regularizations in detail in Section 4.2.4. With this approach we also recommend scaling the parameters of the feature transform after initialization so that the standard deviation of the output on the training data is 1, resulting in a consistent initial bandwidth.

4.2.3 Parameterized Feature Transforms

In principle, any differentiable parameterized class of feature transforms can be used, ranging from simple linear transforms as in ODTs to expressive classes of functions such as MLPs that have high discriminative power on their own. Generally, more expressive feature transforms result in smaller trees, but are probably less interpretable, depending on the application. The goal then is to choose expressive feature classes that remain interpretable in the desired context when conjoined into a rule list, as in the decision path of a tree.

In this work, we focus primarily feature transformations that are fully transparent and propose a few. These can be composed, or their outputs concatenated so different kinds of transforms can be used together in the tree.

Before describing the primitives, we note a general consideration for features used with decision trees: conventional decision trees consider only the order of inputs and not the distance between values. That is, if one modifies the feature values in the data, but does not change the *order* of the training samples when sorted according to each feature, the resulting tree is functionally the same. However, as mentioned in the previous section, for fuzzy decision trees such as KDDTs, the this is not necessarily the case, and monotonic transformations of input do matter, but only on a local scale, so it may be preferable to avoid this kind of unnecessary complexity. Tree learning, whether crisp or fuzzy, is also completely shift invariant, so there is no reason to, for example, include a bias term on a linear transformation.

Identity. It may be beneficial to include the original, unmodified features along with transformed features so that the simplest possible rules are considered during construction of the tree. This also, in some sense, safeguards against situations where transformed features may actually be less informative than the original features.

Element-wise transformation. As noted above, element-wise monotonic transformations on their own are not very important in trees, but they can be useful if composed with other transforms. For example, the composition of element-wise square with linear transformation allows for conic decision rules. Some element-wise mappings worth considering have no parameters, such as $x \mapsto \exp(x)$ and $x \mapsto \log(x)$; others have parameters to be learned, such as $x \mapsto x^\alpha$. A notable option is the element-wise two-parameter Box-Cox transformation

$$x \mapsto \begin{cases} \frac{(x+\alpha)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x+\alpha) & \text{if } \lambda = 0 \end{cases}$$

which is used to alter distribution shape, typically to make values more normally distributed.

Linear transformation. The linear transformation is $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ with weights $\boldsymbol{A}$ and bias $\boldsymbol{b}$. The bias is used only if composing with another primitive due to the shift invariance of tree learning. Using only a linear transformation results in an ODT. These can be initialized randomly, as identity, or using a linear dimension reduction strategy such as principal component analysis.

Distance to prototype. Another feature type maps a point to its distance to each of k prototypes $\boldsymbol{p}_i$; in this sense, a decision rule $d(\boldsymbol{x}, \boldsymbol{p}_i; \boldsymbol{S}_i) < t$ can be interpreted simply as "is $\boldsymbol{x}$ sufficiently similar to $\boldsymbol{p}_i$". We use Mahalanobis distance $d^2(\boldsymbol{x}, \boldsymbol{p}_i; \boldsymbol{S}_i) = (\boldsymbol{x}-\boldsymbol{p}_i)^T \boldsymbol{S}_i^{-1} (\boldsymbol{x}-\boldsymbol{p}_i)$, where the learnable parameters are prototypes $\boldsymbol{p}_i$ and inverse covariance matrices $\boldsymbol{S}_i^{-1}$. The covariance matrix can be reparameterized in various ways to restrict the measurement of distance[1], for example, by making it diagonal, or by using the same for all prototypes. It should also be parameterized such that it is positive semidefinite; otherwise hyperbolic decision rules may be learned and the distance interpretation no longer applies. $\boldsymbol{S}_i^{-1}$ can be regularized for sparsity so that the distance is based on few features. If desired, the prototypes $\boldsymbol{p}_i$ can also be regularized to be similar to the training data; otherwise, they may not be similar to real instances, and in the most extreme case, they may become very distant, essentially collapsing into linear rules. The parameters can be initialized randomly, by selecting samples from the training data, or by using a mixture modeling or clustering algorithm.

Fuzzy cluster membership. Clustering partitions data into k clusters with high internal similarity; usually each cluster i is defined by a center $\boldsymbol{c}_i$, and each point belongs to the cluster with the closest center. Conventional clustering is not differentiable, so we instead use fuzzy clustering, a variant that assigns a degree of membership to each cluster, with the membership values summing to 1. Given distance functions d_i, $i \in [k]$, the membership in cluster i is

$$w_i(\boldsymbol{x}) = \frac{1}{\sum_{j=1}^{k} \left(\frac{d(\boldsymbol{x}, \boldsymbol{c}_i; \boldsymbol{S}_i)}{d(\boldsymbol{x}, \boldsymbol{c}_j; \boldsymbol{S}_j)} \right)^{\frac{2}{m-1}}}$$

with $m \in (1, \infty)$ a hyperparameter determining the "softness" of the cluster assignment, usually just set to 2, and d the Mahalanobis distance as defined previously. In this way, soft clustering can be viewed as a transformation of the "distance to prototype" features so that the resulting transformed features are interdependent and sum to 1; a decision rule $w_i(\boldsymbol{x}) > t$ is interpreted as "$\boldsymbol{x}$ is sufficiently closer to $\boldsymbol{c}_i$ than the other centers". The parameters can be initialized randomly or by using a fuzzy clustering algorithm such as fuzzy c-means [50, 15] with all $\boldsymbol{S}_i$ initialized as identity.

[1] For examples of different covariance types, see https://scikit-learn.org/stable/modules/mixture.html.

Sliding-window features. For modalities with ordered data, such as time series and images, and particularly when the input size is variable or there is some shift or misalignment between instances, one may use a sliding-window version of a feature transformation followed by some aggregation. For instance, a sliding window with a linear transformation is a convolution; a sliding window with a distance-to-prototype applied to time series is a time series shapelet, and this is easily generalized to two dimensions to create an image shapelet. For shapelets, we typically use distance with identity or diagonal covariance. Identity covariance is useful for simple interpretation. Diagonal covariance is useful because it increases expressiveness without severely affecting interpretability. It allows the weight of features to go to zero in the computation of distance, enabling the automatic adaptation of the size of the shapelet. It also can and frequently does result in sparsely computed distance when learned with a sparsity-promoting regularization. One may also regularize the prototypes $\boldsymbol{p}_i$ for sparseness and/or smoothness along the ordered axes (time axis for time series, or x and y axes for images) to improve interpretability. To aggregate the result of the convolution into a single value, one may for example take the average or the minimum or maximum; the minimum is standard for shapelets since it indicates a "best match" to the template. In addition to other options, shapelets may also be initialized by model-agnostic algorithms for learning a shapelet transform, such as the one proposed by [92].

4.2.4 Regularization

For each transformed feature $i \in [p']$, the parameters $\theta_i \subseteq \theta$ used to compute the ith transformed feature may be subject to a regularization weighted by the total usage of feature i, that is, the sum of total training sample weights over nodes that split using feature i. This weighting serves to balance the effects of the loss and regularization; a feature used in the decisions for relatively few data will have a relatively smaller loss gradient, and so should have a proportionally smaller regularization, and vice versa.

One reason to apply regularization is to improve the interpretability of features. The L1 regularization increases the sparsity of parameters, which can reduce the complexity of interpretation. Other regularizations might improve interpretability in an application-specific way, for instance, the smoothing of prototypes or shapelets for time series and images. Weighting by feature usage also ensures that often-used features are the most interpretable, whereas the opposite will tend to be the case with unweighted regularization.

Another reason to apply regularization is to automatically regulate kernel bandwidth. By using regularization that shrinks the output range of the features, such as a L1 or L2 regularization on a linear transformation, we effectively increase the bandwidth; meanwhile, minimizing the loss function shrinks the effective bandwidth because a smaller bandwidth allows fewer data to have split decision paths, resulting in purer leaves. This makes the actual choice of kernel bandwidth relative to regular-

data n, p (one-hot), q	LR	MLP	DT	RF	ET	XGB	ours: linear		ours: proto	
							fuzzy	crisp	fuzzy	crisp
iris [60]	**0.960**	0.953	0.947	0.947	0.953	0.947	**0.960**	**0.960**	0.947	0.927
150, 4 (4), 3	-	-	6.4	7.2e2	2.1e3	4.3e2	6.1	7.6	6.5	2.9
heart-disease [97]	**0.822**	0.792	0.707	0.802	0.795	0.792	0.812	0.812	0.793	0.779
303, 13 (20), 2	-	-	13.9	4.8e3	1.1e4	7.9e2	21.6	19.4	5.0	3.9
dry-bean [104]	0.925	**0.934**	0.912	0.923	0.921	0.928	0.920	0.913	0.915	0.901
13611, 16 (16), 7	-	-	99.8	6.7e4	2.0e5	1.3e4	1.1e2	45.8	1.6e2	94.2
wine [2]	0.983	**0.989**	0.904	0.977	**0.989**	0.955	0.983	0.983	0.961	0.961
178, 13 (13), 3	-	-	8.5	9.4e2	3.3e3	2.4e2	2.0	2.0	2.3	2.3
car [19]	0.926	0.992	0.977	0.964	0.971	**0.994**	0.991	0.992	0.980	0.979
1728, 6 (21), 4	-	-	95.3	2.3e4	3.1e4	4.5e3	29.0	29.0	59.5	55.9
wdbc [187]	0.974	**0.975**	0.935	0.965	0.970	0.968	0.972	0.972	0.961	0.961
569, 30 (30), 2	-	-	13.0	1.9e3	6.0e3	2.7e2	1.3	1.3	6.4	5.0
sonar [153]	0.755	0.879	0.735	0.826	**0.880**	0.855	0.818	0.799	0.798	0.817
208, 60 (60), 2	-	-	14.1	2.0e3	5.6e3	3.0e2	5.7	3.9	10.3	10.3
pendigits [5]	0.952	**0.994**	0.964	0.993	**0.994**	0.991	0.981	0.976	0.950	0.931
10992, 16 (16), 10	-	-	3.2e2	3.8e4	9.8e4	8.5e3	2.6e2	2.4e2	3.1e2	3.1e2
ionosphere [154]	0.875	0.917	0.892	0.934	**0.943**	**0.943**	0.932	0.920	0.920	0.909
351, 34 (34), 2	-	-	15.5	2.2e3	5.9e3	3.4e2	3.9	5.5	10.0	6.3

Table 4.1: 10-fold cross-validation accuracy and average number of splits for tree-based models. Best in bold.

ization strength so that one can always use the same kernel bandwidth and just tune the regularization strength. To some extent, it also automatically adapts effective bandwidth individually for each feature, and overall improves performance compared to using a fixed effective bandwidth.

4.3 Demonstration and Evaluation

Here we benchmark the proposed learning algorithm and demonstrate it and the resulting models' interpretation on several data sets. Additional experiment details are in Appendix C.6 and comprehensive results are in Appendix D.2.

4.3.1 Benchmarks on Tabular Data

We compare various configurations of our algorithm against popular tree-based baselines including decision trees, random forests, and ExtraTrees. We report 10-fold cross validation accuracy and average number of splits in the model.

The data sets are selected from among the the most viewed tabular classification data sets on the UCI machine learning repository [49] at the time of writing.

Categorical attributes are one-hot encoded, and the data is normalized to mean 0 and standard deviation 1; this is good for training and makes interpretation unitless. For our models, we show results for linear features and distance-to-prototype features with diagonal inverse covariance. Each is regularized with L1 coefficient $\lambda_1 = .01$ to promote sparsity. Our models and the conventional decision trees have cost-complexity pruning α selected by cross-validation. Other hyperparameters are fixed and described in Appendix C.6.1.

Results are shown in Table 4.1. Results for additional experimental configurations with additional metrics, including average decision path length, feature sparsity, and inference time are shown in Appendix D.2.1. On every data set, at least one of our models matches or comes close to the best baseline accuracy while being much smaller. Also note that, since we choose the pruning parameter α by cross-validation, there are many cases where a smaller tree than the one reported performs similarly. Especially for data sets where the ensembles greatly outperform the basic decision tree, the reduction in size for a performant model by using our method is huge. This carries the additional benefit that inference is much faster for our models. Models with linear features are often the best for a given dataset, while prototype features lag slightly behind, suggesting that prototype features are best used to supplement linear features rather than on their own. We also note that these benchmarks all use the same hyperparameters; this shows that good performance does not require great tuning effort, but such effort will probably result in a better model for most applications.

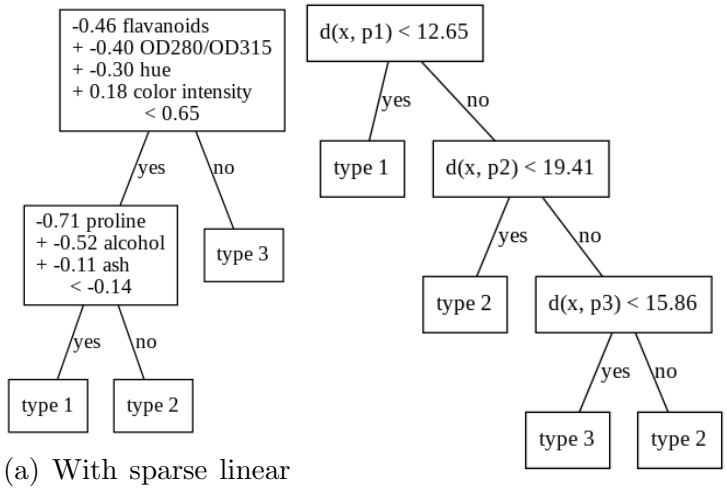
We also separately show results on MNIST and fashion-MNIST; these experiments are described in Section 4.3.3 and the results are shown in Table 4.2.

4.3.2 Interpreting a Wine Classifier

We show two examples of interpretation for trees trained on the wine data set. Both are trained with $\alpha = 0.01$ as in the benchmarks and achieve 97.2% accuracy on a 20% test split.

A tree with linear features is shown in Figure 4.2a. It has a stronger sparsity regularizer $\lambda_1 = 0.1$ compared to the benchmarks. Wines with low flavanoids, protein concentration, and hue are classified as type 3. There is also a smaller relationship between type 3 and high color intensity. Of the remaining wines, those with low proline and alcohol are type 2, and the rest are type 1. Similarly, there is also a smaller relationship between ash content and type 1 that may be important if the decision is not clear based on proline and alcohol.

A tree with prototype features is shown in Figure 4.2b. Here we use simple Euclidean distance for decision rules. Each prototype defines a certain wine profile; for instance, prototype 1 is high in alcohol and proline and close to average in other attributes. Wines similar to prototype 1 are type 1; wines similar to prototype 2 are type 2; wines similar to prototype 3 are type 3; the rest are type 2.



(a) With sparse linear features.

feature	p1	p2	p3
alcohol	2.03	-1.66	-0.15
malic acid	0.85	-0.35	1.15
ash	0.06	-1.00	-0.62
alcalinity of ash	-1.16	0.11	0.22
magnesium	0.95	-2.06	-0.08
total phenols	1.27	0.70	-1.46
flavanoids	0.04	2.69	-0.54
nonflav. phenols	-0.86	-0.33	0.29
proanthocynanins	0.74	-0.24	0.57
color intensity	0.70	-1.95	2.51
hue	0.43	1.10	-1.05
OD280/OD315	0.51	0.50	-1.31
proline	1.91	-1.55	-1.06

(b) With distance-to-prototype features.

Figure 4.2: Interpretable trees for wine classification.

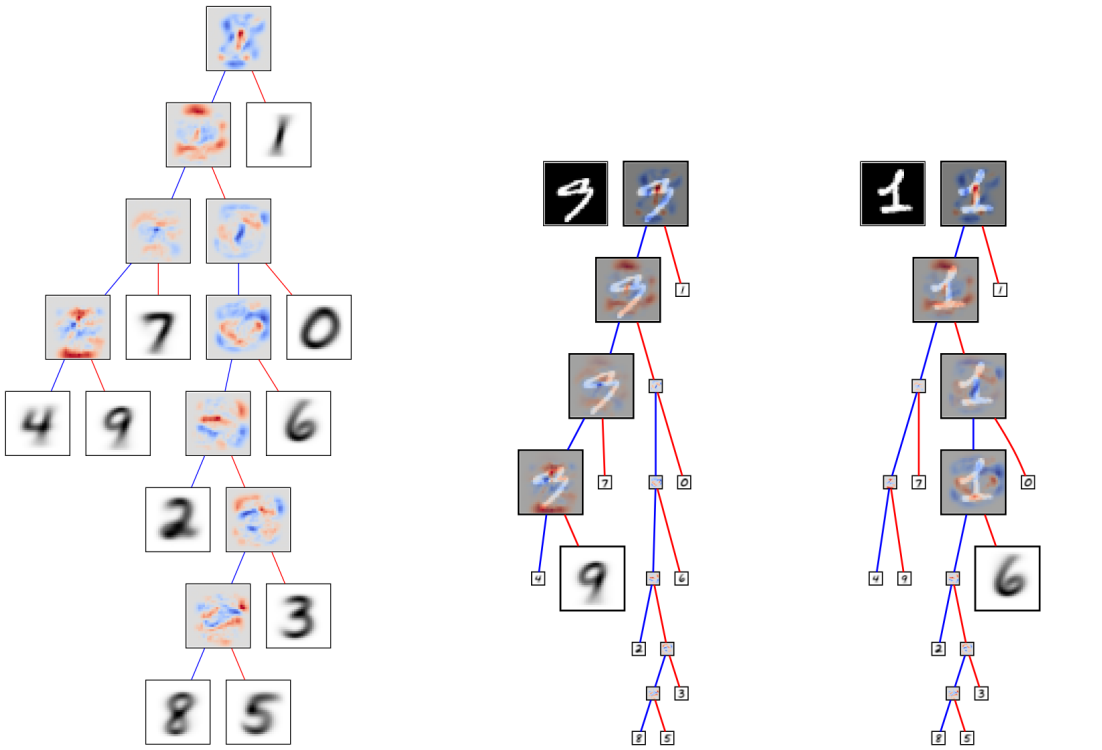
	MNIST				Fashion-MNIST			
α	fuzzy acc.	crisp acc.	splits	path len.	fuzzy acc.	crisp acc.	splits	path len.
10^{-2}	.9226	.9219	9	4.33	.8059	.8024	8	3.97
10^{-3}	.9500	.9419	36	5.91	.8439	.8364	20	4.69
10^{-4}	.9664	.9610	162	7.68	.8669	.8586	98	6.24
10^{-5}	.9708	.9602	1659	10.05	.8675	.8472	2209	11.55
RF	N/A	.9697	500524	1700	N/A	.8767	496819	1822

Table 4.2: Test accuracy and tree size for MNIST trees with linear features.

4.3.3 Interpreting MNIST Classifiers

Next we fit models to MNIST and Fashion-MNIST to demonstrate performance and interpretability on simple image classification. The performance and size of models for various cost-complexity pruning α are shown in Table 4.2, and the smallest MNIST model is shown in Figure 4.3. Larger models, as well as Fashion-MNIST models, are shown in Appendix D.2.2. Here we use linear features. Each internal node shows the feature's weight image, with negative values in blue and positive in red. Each leaf shows the average of the training data belonging to it.

We use some extra constraints to improve interpretability. First, in addition to a L1 regularization to promote sparsity, we use a smoothness regularization which penalizes the average squared difference of each weight with its neighbors, not including diagonal neighbors. The idea is that a weight image with smooth shapes is easier to understand than one with just a sparse handful of pixels, while also being more expressive. We notice that this smoothing can also reduce overfitting. We constrain the tree's threshold values to be zero to ease local interpretation. In this way, we can interpret a decision rule by overlaying the digit onto the weight image and asking



(a) The entire model.

(b) Interpretation of a correct prediction.

(c) Interpretation of an incorrect prediction.

Figure 4.3: An interpretable MNIST classifier with 92.19% test accuracy.

simply "Do the pen strokes better match the blue, or the red?" For example, in the root node in Figure 4.3a, we see that digits with ink primarily in a small near-vertical stroke in the center are classified as 1, whereas those with more ink in the closely surrounding space are other digits. Complete predictions are shown Figures 4.3b and 4.3c. In the case of the incorrect prediction, we can easily see that the error occurs in the root node, where the large serifs on the handwritten 1 overlap with too much blue in the weight image and result in the decision that this digit is not a 1. It is reasonable to assume, then, that the model will always predict incorrectly on similar large-serif 1 images, and that such digits might be underrepresented in the data. This demonstrates how interpretability can be used as a diagnostic tool.

For reference, Table 4.2 also shows performance and size for standard decision trees and random forests. As the size of our tree grows large, performance matches the random forest on MNIST and comes very close on Fashion-MNIST. Moreover, though our trees may seem large, they are still hundreds of times smaller than a comparable random forest, and the average decision path length, that is, the average number of nodes involved in the interpretation of a prediction, is thousands of times smaller. To separate 10 classes, our largest models require only 10.05 or 11.55 decision

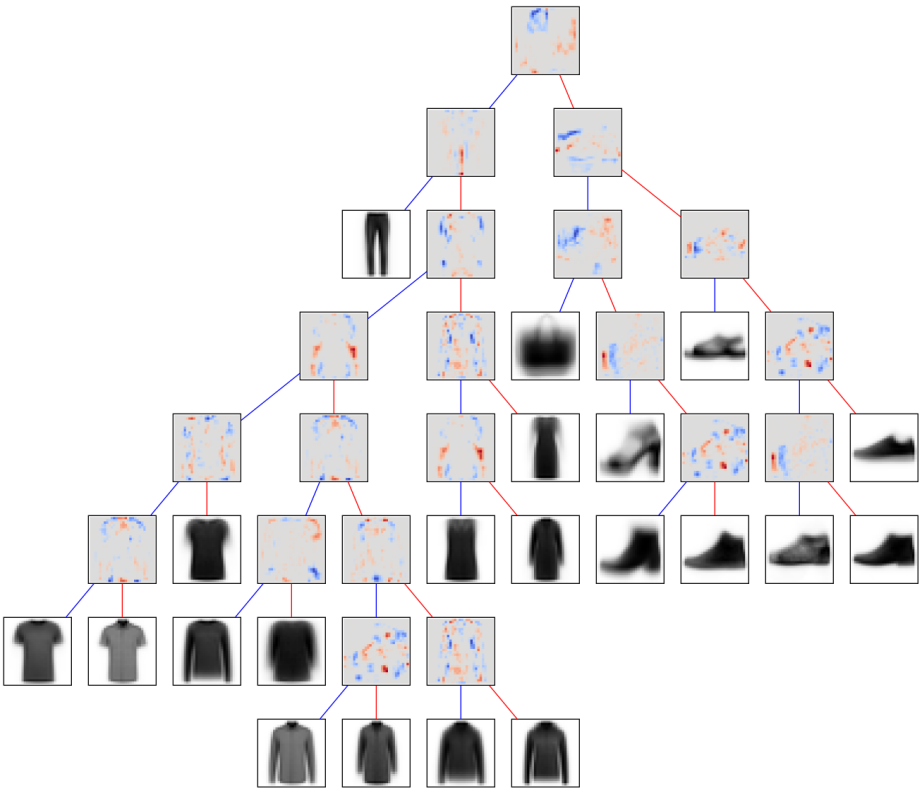


Figure 4.4: An interpretable Fashion-MNIST classifier. The tree automatically learns a hierarchy of fashion items.

rules on average for MNIST and Fashion-MNIST, respectively.

Testing on MNIST allows us to make some comparison of our models to TAO [28], which seems to be the best existing method for learning trees with sparse linear features. Ultimately, while their smallest model achieves 89.81% test accuracy with 16 splits, our smallest achieves 92.19% with just 9 splits. Likewise, their overall best-performing model achieves 94.31%, whereas ours achieves 96.10% without fuzzy splitting or 97.08% with it. This shows that our proposed approach can achieve state-of-the-art status for learning sparse oblique decision trees while also having greater flexibility to choose different features and adapt them to application-specific interpretability needs.

4.3.4 Learned Class Hierarchy in Fashion-MNIST

An additional property of our models is that, differently from other ODT learning algorithms such as TAO, we continually refit the tree using a generalization of CART, which greedily chooses the most informative splits first. Each subtree's class labels are thus as pure as possible, and classes that are easy to separate are separated first. By contrast, in a tree which is trained globally without this greedy splitting, any

particular split may not actually result in improvement in label purity, as long as it is informative to splits further down the tree. We argue that this characteristic of our trees is useful for interpretability.

A compelling example is shown in Figure 4.4, which shows the tree trained on Fashion-MNIST with $\alpha = 10^{-3}$. The tree forms an intuitive hierarchy of visual similarity of fashion items. Tops, bottoms, shoes, and bags are separated early. Within shoes, there is a separation of high vs. low shoes, then open vs. closed shoes. Within tops, there is separation of long vs. short tops; within short tops, there is separation of long vs. short sleeves; then, within long sleeves, there is separation of tops with a collar vs. tops without a collar. This allows us to understand decision rules and interpret predictions just by examining the populations at each leaf, *without even knowing the transformed features*. A similar structure of visually cohesive subtrees is observed in our MNIST trees, as in Figure 4.3a, but it is less obvious and intuitive since we do not tend to cognitively group digits by visual similarity like we might with fashion items. We do not observe this property in other ODTs; for example, in [158], a TAO tree fitted to Fashion-MNIST does not exhibit this visual hierarchy.

4.3.5 Interpreting Time Series Shapelets

Heartbeat Classification

We demonstrate classification with time series shapelets on the ECG5000 data set from the UCR archive [43]. This consists of a 20 hour ECG of a patient with severe congestive heart failure segmented into single heartbeats and interpolated into a fixed length. 500 training and 4500 test samples are randomly selected and annotated as either normal or one of several irregularities. The task is to classify a heartbeat according to this annotation.

Since the heartbeats are segmented to have the same shift and scale, shapelets without sliding window are an appropriate feature choice. The resulting tree is shown in Figure 4.5. Here the red and blue heatmaps show the distribution of training data at each node, that is, dark red shows where many waveforms are present, white where a moderate number are present, and dark blue where none are present. The shapelet at each node is shown in green. The opacity corresponds to the weight at each time step, and since we regularize the shapelets for sparsity, the green areas are small, indicating that only a small portion of the waveform is used in each decision. Data sufficiently similar to the shapelet go left down the tree, and others go right. Within each internal node is also shown the distribution of transformed feature values (distances to the shapelet) colored by class label, as well as the tree's decision threshold and the transition region of the fuzzy split. Within each leaf node is also shown the weight of data for each label; the leaf predicts the label with the highest weight. Based on this representation and the small tree size, it is very clear how decisions are made. For instance, we can see from the shapelets of the root node and its left child that the model labels waveforms with a prominent U-shaped curve on the right side as

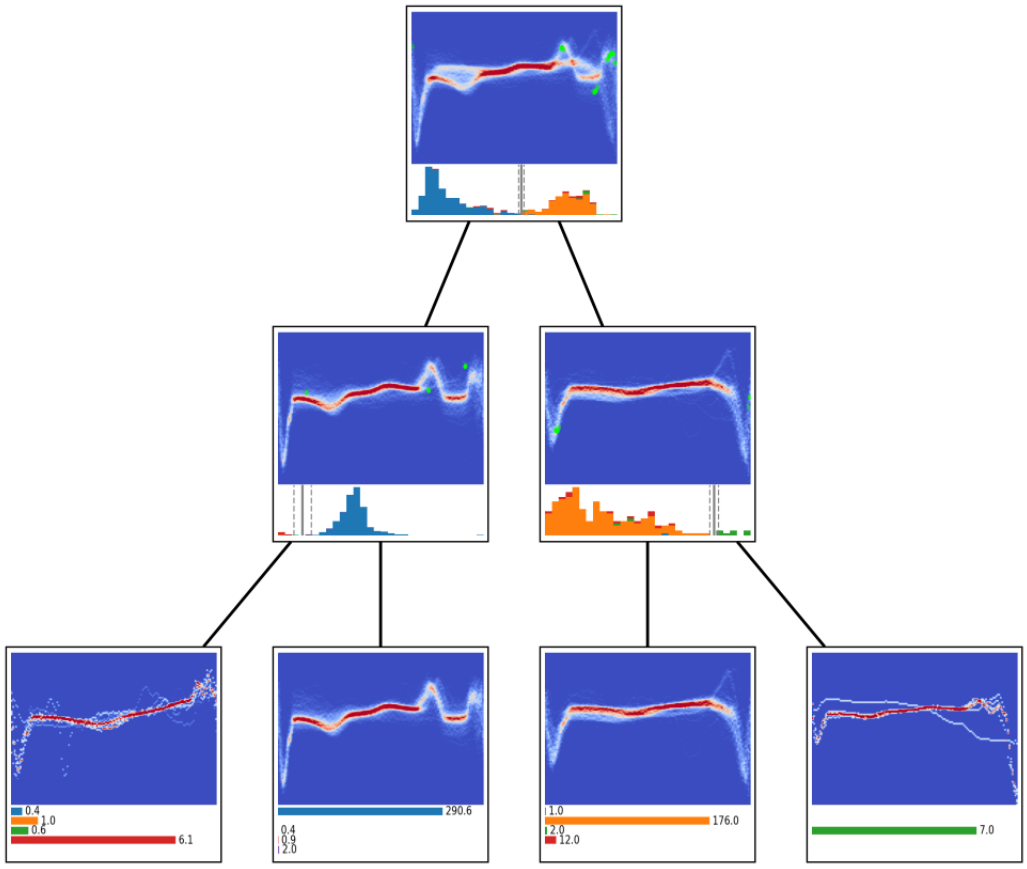


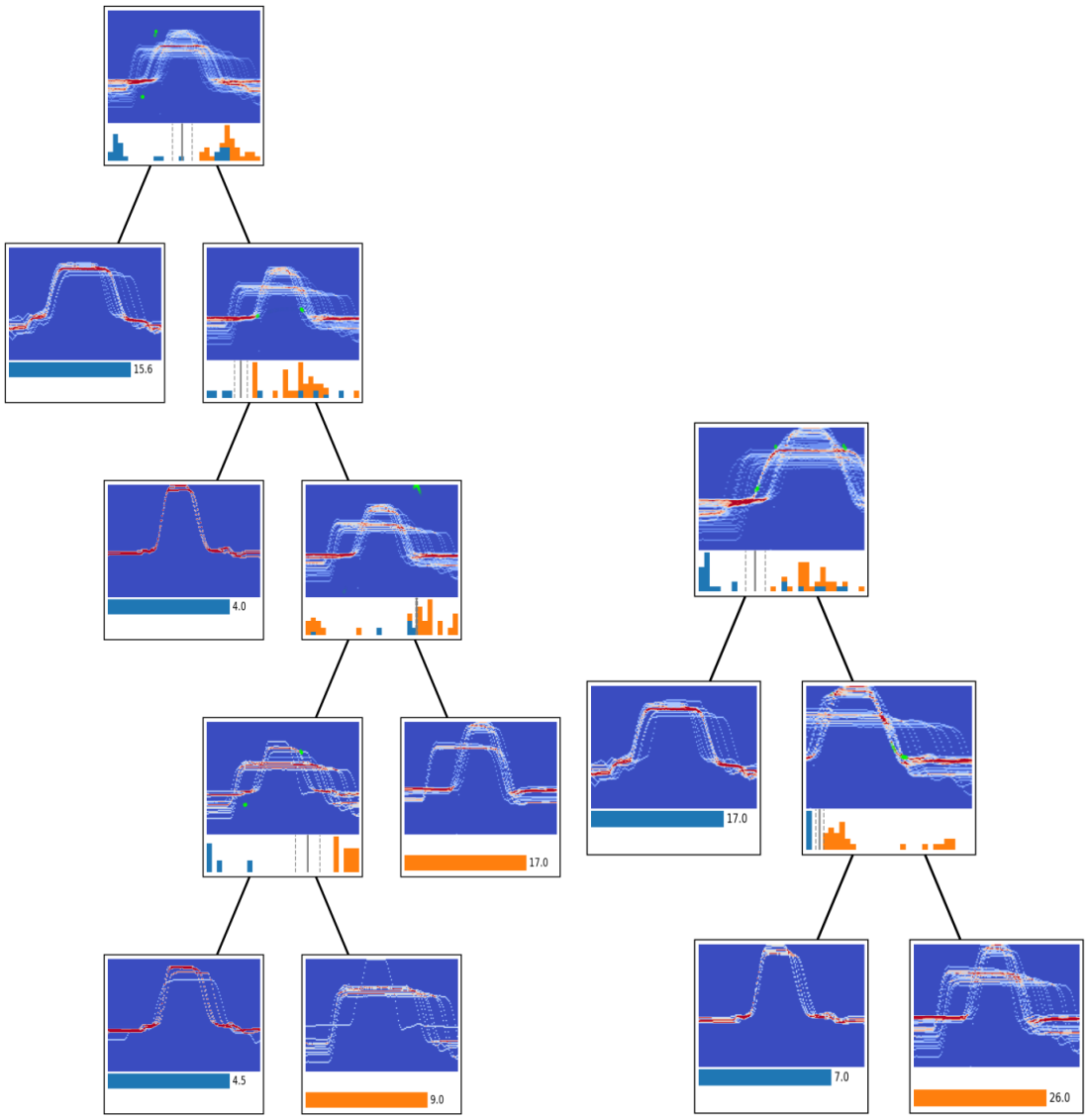
Figure 4.5: A shapelet-based tree for heartbeat classification.

class 1. Though we can see from the leaves that class 1 also exhibits a small dip in the waveform in the middle-left area, we can tell from the shapelets that this is less important to the model's decision.

This model has 97.2% training and 92.8% test accuracy; slightly better test accuracy can be achieved with a change of hyperparameters, but the resulting model is larger and less interpretable. For context, in a benchmark of deep learning for time series classification by [95], the best model on ECG5000 is "Encoder" with an average test accuracy of 94.1%. Meanwhile, our tree of just three interpretable splits matches the performance of MLP and CNN at 92.8% and 92.9% test accuracy, respectively, and outperforms some deep architectures such as t-LeNet and TWIESN.

Gun Detection

Next, we demonstrate classification with time series shapelets on the GunPoint data set from the UCR archive [43]. In this data set, an actor is filmed in profile either pointing a finger or raising a replica gun and the x-coordinate of the hand is recorded as a time series. The task is to determine whether the actor is pointing a finger or



(a) Without sliding window: 86.7%. (b) With sliding window: 96.0%.

Figure 4.6: Shapelet-based trees on the GunPoint data set and their test accuracy.

raising a gun.

Figure 4.6 shows trees trained with shapelet-based features, both with and without sliding window. See the previous subsection on heartbeat classification for a description of this kind of visualization In the sliding window case, the window size is 100, where the time series are of length 150. We can see from the distribution of time series in the root node of Figure 4.6a, which shows the whole training set, that there is some seemingly random time shift in the data, making the resulting tree complex and prone to overfitting. However, with the sliding window, as shown in Figure 4.6b,

where each time series is shown aligned to its best match with the shapelet, the sliding window is able to effectively align the data, resulting in a simpler tree with much better generalization. For other data, one might also consider scale invariance as in dynamic time warping, as well as shift and/or scale invariance of the value, that is, the y-axis of the time series. For context, [95] report an average test accuracy of 100% for FCN, but the majority of the remaining deep architectures perform below our interpretable model's score of 96.0%.

4.4 Discussion

We have proposed a new system for learning small, interpretable decision trees and shown examples under several notions of interpretability, including sparse linear combinations, similarity to prototypes, visual matching, and class-based hierarchies. Compared to ensembles and deep models, our single-tree models are interpretable because they are small, use sparse features, have a sparsely activated hierarchical representation based on logical rules, and work by simply partitioning the data. We use size and sparsity as rules of thumb for interpretability and design our demonstrations accordingly, but ultimately, interpretability is highly subjective, and this is not the best for every user and application. For instance, in some cases, high-level conceptual interpretability, such as the "this looks like that" interpretation of image prototypes [34], may be preferred over the detailed low-level interpretations we present in this work. Our methods are highly flexible to customize feature types to meet interpretability needs, especially as more diverse feature types are implemented, and towards this capability, ongoing work is developing a platform for interactive visualization and design of these models. Since there is no universally applicable quantification of interpretability, an ideal evaluation of interpretability will include user studies.

One tradeoff is that the models can take relatively long to train due to repeated fitting of fuzzy trees, though this can be alleviated by controlling the number of features output by the feature transformation; for instance, if using a linear transformation with MNIST as in our demonstrations, using the transformation to map the 784 features of MNIST down to just 10 to 20 transformed features, which is sufficient for a small tree, allows training to complete in mere minutes. Another tradeoff is that, while our results show that good models can be obtained without extensive hyperparameter tuning, the absolute best model for a given scenario certainly may require careful design of the feature transformation architecture and tuning of several hyperparameters.

A promising future direction for this work is to incorporate more feature types that may achieve good performance in more applications. One example is long, variable length time series where a "best match" feature may be insufficient, or time series where frequency information, which is not easily captured by the feature types covered here, is important. One might also consider possibilities for interpretable featurization

of text so that this work can be extended to text classification.

Another potential application of new feature types is broader image classification. The methods demonstrated here with linear and prototype features with optional sliding window are well-suited to data such as MNIST that is inherently two-dimensional and highly structured. However, more natural image data capturing three-dimensional objects with occlusions, variable lighting, etc. is more challenging. We find that, while our methods using the features proposed in this work can significantly improve the performance of trees on this kind of data, it still falls short of modern performance expectations, and the models can be difficult to interpret. One way to approach this would be to use CNN-based image prototypes as described in Section 4.1.3, which would result in rules based on visual similarity to a template, though the determination of similarity is made by a black-box. While this moves away from the ideal of full transparency, it may nonetheless improving interpretability compared to the current state-of-the-art.

Chapter 5

Pragmatic Advancements

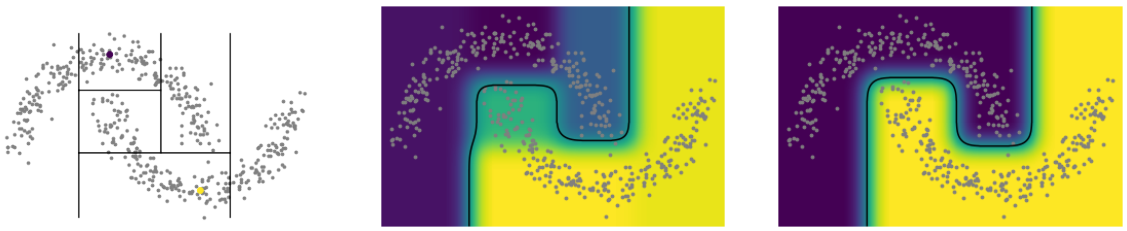
Much of current ML research commonly focuses strictly on parametric models, and so many topics advanced by recent research are not applicable to non-parametric models such as trees. Tree-based models in particular still achieve state-of-the-art performance on tabular problems [81] and are widely used due to their practical advantages, so gaps in their applicability are a crucial weakness in ML application.

In this chapter, we apply the uncertain interpretation paradigm towards closing some of these gaps. First, we propose an intrinsic tree-based method for Semi-supervised Learning (SSL), that is, learning from both labeled and unlabeled data, where there are few existing methods for decision trees. Next, we introduce a new model-agnostic method of Federated Learning (FL), a field dominated by parameter aggregation and regularization methods, by regularizing models for agreement in function space. Since loss functions are generally convex in function space, it has convergence guarantees with mild assumptions; when combined with our uncertain interpretation formalism and a quadratic loss function, we have fast convergence close to the consensus optimum. This kind of function space regularization can naturally also be used for other applications involving measuring and minimizing model disagreement. As an example, we demonstrate the merging of tree ensembles into a single tree.

5.1 Semi-supervised Learning

In applications of supervised machine learning, there is often copious data available, but labeling the data accurately is expensive. This motivates Semi-supervised Learning (SSL), which combines strategies from supervised and unsupervised learning to learn a predictive model from labeled and unlabeled data. It does this by leveraging some underlying assumption that informs the use of unlabeled data, such as clusterability, smoothness, or separability of the data. Effective SSL can greatly reduce the amount of labeled data needed to achieve a particular level of performance.

Most SSL methods are designed for parametric models. As a result, despite the



(a) Fit a tree to minimize semi-supervised impurity.

(b) Assign smoothed leaf values.

(c) Or assign robust leaf values.

Figure 5.1: An overview of our SSL algorithm. Gray data are unlabeled.

popularity and utility of tree-based models, SSL methods for trees are largely limited to model-agnostic wrapper methods such as self-training [173], for which tree-based models are not even well-suited [174]. This lack of options also limits the choice of assumption that drives the SSL process.

The main function of the few SSL algorithms specialized for decision trees [118, 119, 109] is choosing splits in a semi-supervised way; they do not propagate labels across regions of dense unlabeled data, limiting expressiveness. To fill this gap, we leverage KDDTs and their uncertain interpretation of input to introduce natural and efficient SSL algorithms based on a choice of two grounded assumptions:

- Smoothness assumption: inputs with similar feature values are likely to have similar labels.
- Robustness assumption: there exists a large-margin (on average) boundary between classes.

We first grow a tree using a semi-supervised splitting criterion that leverages fuzzy membership; unlike prior methods, it allows leaves to contain no labeled data, increasing expressiveness since trees may grow large even with very few labeled data. We then assign leaf values according to one of the above assumptions by constructing a similarity graph over the leaves and using a graph algorithm. As usual, the final model can be fuzzy or a conventional crisp tree. Figure 5.1 motivates this kind of approach and highlights the two leaf assignment strategies. Previous tree-based SSL methods do not propagate labels across leaves, so they cannot learn a good boundary in this example.

Ultimately, semi-supervised learning algorithms perform largely based on how well their underlying assumption(s) describe the data; by introducing two new assumptions for tree-based models—assumptions which are grounded, reasonable, and used in SSL for other models—we improve performance on data where other tree-based methods fall short, as shown by our experiment results in Section 5.1.3.

5.1.1 Related Work: Semi-supervised Learning

Over many years, numerous SSL algorithms with various underlying assumptions have been proposed for a wide range of models. We refer the reader to [174] for a comprehensive overview. We highlight two taxonomic distinctions: first, a method is *inductive* if it produces a predictive model, or *transductive* if it only assigns labels or pseudo-labels to the unlabeled training data; second, an inductive method is a *wrapper method* if it is agnostic to the predictive model used, or *intrinsic* if it is specialized for a particular class of models. The algorithm presented in this work is thus an intrinsically semi-supervised method for induction of decision trees.

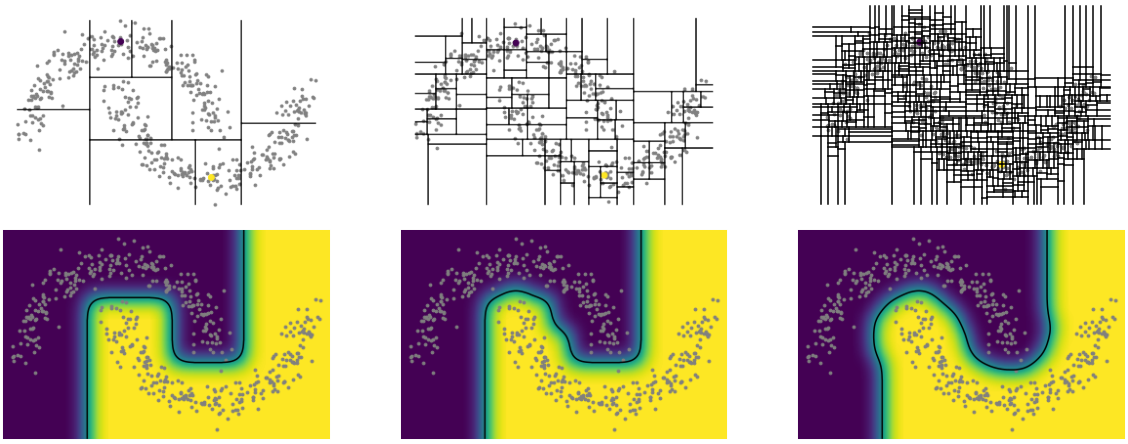
Wrapper methods were once the only option for semi-supervised learning of decision trees and remain a popular approach. Foremost are variants of self-training [173], an iterative process whereby the model is fitted to labeled data, then the most confidently predicted among the unlabeled data are labeled and added to the training pool for the next iteration. Despite its widespread use with trees, tree-based models are actually not well-suited for self-training due to poor calibration and overconfident predictions [174]. As a result, works such as [112, 108, 44, 167, 120] propose strategies to improve self-training of tree-based models.

A few intrinsic methods have been proposed for decision trees. By assigning pseudo-labels to unlabeled data using posterior probability computed from kernel density with reduced dimension on the labeled data, [118] and [119] choose better oblique splits using unlabeled data. Similarly, by using impurity of the *features* in addition to impurity of the *labels* as a splitting criterion for tree construction, [109] propose Semi-supervised Learning Predictive Clustering Trees (SSL-PCTs) that choose better conventional axis-aligned splits using unlabeled data. Our method of tree growth is similar in that it uses a semi-supervised splitting criterion, but our unsupervised impurity is based on fuzzy membership stead of feature variance and has linear rather than quadratic computational complexity in the number of features.

Graph-based SSL is a family of transductive methods that constructs a similarity graph over the data and uses a graph algorithm to assign pseudo-labels. We mention two that are relevant to this work. First, label propagation [204] solves a linear system such that each pseudo-label is a combination of its neighbors' labels or pseudo-labels, weighted by similarity. Second, graph min-cut approaches [17, 18] use min-cut algorithms to separate the data into classes such that the inter-class similarity is minimized.

5.1.2 Semi-supervised Learning with KDDTs

Our SSL algorithm has two phases. First, the tree structure is grown using a semi-supervised variation of KDDT fitting. Next, a similarity graph over the leaves is constructed and used to assign leaf values. This assignment, differently from previous work, propagates labels across leaves, so all leaves can be assigned a value even if only a few contain labeled data. The tree growth and assignment approaches are described



(a) 10 leaves (0.05 seconds). (b) 100 leaves (0.4 seconds). (c) 1000 leaves (4 seconds).

Figure 5.2: Trees of varying size with robust leaf assignment and their training time.

in the following sections. A visual summary is provided in Figure 5.1.

Semi-supervised Tree Growth

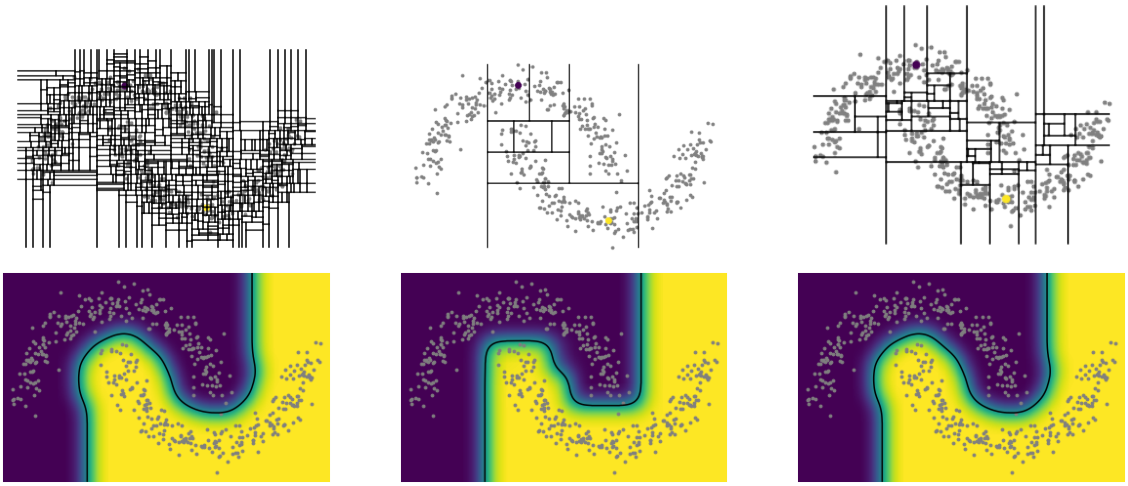
We grow trees in a semi-supervised manner using a simple adaptation of supervised splitting criteria; compute the semi-supervised impurity by assigning each unlabeled point its own unique label. This will cause the tree to separate unlabeled data points, but for crisp trees, it is otherwise uninformative, that is, it does not influence the optimal split. However, for fuzzy decision trees, it prioritizes splits which are the *least* fuzzy, that is, most unlabeled data belongs entirely to one side or the other, resulting in well-separated subgroups. For KDDTs, this means prioritizing splits far away from most data, with the resulting leaves containing data that is more difficult to separate, and thus more similar in feature values. This is similar to SSL-PCTs [109], but whereas the computation of feature variance in the construction of SSL-PCTs makes the cost of split search quadratic in the number of features, ours remains linear since each unlabeled data point contributes independently to the impurity. See Figure 5.1a for a toy example of tree fitting.

Recall that we write $\mu_i(\boldsymbol{x})$ the membership of $\boldsymbol{x}$ at node i and w_i the total weight of training data as defined in Equations 2.6 and 2.10. The semi-supervised Gini impurity at node i can be written

$$\text{Gini}(i) = 1 - \frac{1}{w_i^2}\left(\left\|\sum_{j\in D_L}\mu_i(\boldsymbol{x}_j)\boldsymbol{y}_j\right\|_2^2 + \sum_{j\in D_U}\mu_i(\boldsymbol{x}_j)^2\right) \tag{5.1}$$

for labeled data indices D_L and unlabeled data indices D_U.

By writing the impurity in terms of the selected decision threshold and using a Taylor expansion of the unsupervised term, a straightforward extension of the KDDT



(a) The 1000-leaf model from Figure 5.2c.

(b) Pruned to 10 leaves by leaf impurity.

(c) Merged same-label leaves down to 75 leaves.

Figure 5.3: Two methods for pruning a tree based on the assigned leaf values.

threshold search makes it possible to compute this in constant time for each candidate threshold, preserving the original asymptotic cost of threshold search.

As always, KDDTs can grow extremely large unless a growth stopping condition is set, and this is especially true with the semi-supervised criterion. An example of size variation on a toy data set with 500 samples is given in Figure 5.2. Generally a good choice to limit tree growth is a Cost-Complexity Pruning (CCP) parameter α. However, with our the semi-supervised criterion, the tree size can be very sensitive in a small range of α values that depends on, among other things, the number of unlabeled data; therefore, as a more clear and consistent hyperparameter, we instead set a maximum number of leaves and grow the tree in order of highest gain until reaching that number of leaves, or until no split results in any gain. This constructs the same tree that could be attained by choosing some particular CCP-α value, but allows one to choose the size directly. Other growth stopping conditions, such as maximum depth or minimum sample weight in a leaf, are also options.

If needed, a fitted tree with its leaf values assigned can be post-pruned according to the assigned leaf values to obtain a much smaller model for prediction, as demonstrated in Figure 5.3. One option is to prune according to supervised impurity, such as conventional Gini impurity, calculated using the assigned leaf values, as shown in Figure 5.3b. Compared to the equally sized tree in Figure 5.2a, which is grown to that size, the tree in Figure 5.3b, which is grown larger and then pruned down, fits the data better. Another option is to merge contiguous leaves with the same plurality label, as shown in Figure 5.3c. If either the tree is crisp or the leaf values are one-hot, as in our robust leaf value assignment method, this will never change the predicted class; compare, for example, Figures 5.3a and 5.3c. Otherwise, it may alter

the prediction in rare cases, but the model remains largely the same. Both strategies can produce better final models for a given size at the cost of more time spent fitting.

Smooth leaf value assignment

As a result of the smoothing induced by the uncertain input interpretation defining KDDTs, a natural SSL approach emerges by simply writing the predictions for unlabeled data and using the resulting system to solve for leaf values. Recalling Equations 2.8 and 2.11, a KDDT f makes prediction

$$f(\boldsymbol{x}) = \sum_{i \in \text{leaves}} \mu_i(\boldsymbol{x}) \boldsymbol{v}_i = \sum_{i \in \text{leaves}} \mu_i(\boldsymbol{x}) \frac{1}{w_i} \sum_{j \in D} \mu_i(\boldsymbol{x}_j) \boldsymbol{y}_j$$

for training data $D = D_L \cup D_U$.

For $j \in D_U$, $\boldsymbol{y}_j$ is unknown; instead substitute the predicted $f(\boldsymbol{x}_j)$. Then we have

$$\boldsymbol{v}_i = \frac{1}{w_i} \sum_{j \in D_L} \mu_i(\boldsymbol{x}_j) \boldsymbol{y}_j + \sum_{i' \in \text{leaves}} \left(\frac{1}{w_i} \sum_{j \in D_U} \mu_i(\boldsymbol{x}_j) \mu_{i'}(\boldsymbol{x}_j) \right) \boldsymbol{v}_{i'}$$

which, by stacking over each i, forms a linear system

$$\boldsymbol{V} = \boldsymbol{V}' + \boldsymbol{A}\boldsymbol{V} \tag{5.2}$$

which we use to solve for the unknown matrix $\boldsymbol{V}$ of leaf values. Here each row is the corresponding leaf's value $\boldsymbol{V}_{i,:} = \boldsymbol{v}_i$, $\boldsymbol{V}'$ is similarly the known matrix of leaf value components from labeled data only $\boldsymbol{V}'_{i,:} = \frac{1}{w_i} \sum_{j \in D_L} \mu_i(\boldsymbol{x}_j) \boldsymbol{y}_j$, and $A_{i,i'} = \frac{1}{w_i} \sum_{j \in D_U} \mu_i(\boldsymbol{x}_j) \mu_{i'}(\boldsymbol{x}_j)$ indicates the weight of shared unlabeled samples for each pair of leaves i, i'. $\boldsymbol{A}$ can be interpreted as a similarity graph over the leaves.

Equivalence to label propagation

Given a kernel $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_{\geq 0}$, label propagation [204] assigns pseudo-labels $\boldsymbol{Y}_U$ by solving the linear system $\boldsymbol{Y}_U = \boldsymbol{K}_{UU} \boldsymbol{Y}_U + \boldsymbol{K}_{UL} \boldsymbol{Y}_L$ where

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{LL} & \boldsymbol{K}_{LU} \\ \boldsymbol{K}_{UL} & \boldsymbol{K}_{UU} \end{bmatrix}$$

with $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) / \sum_\ell k(\boldsymbol{x}_i, \boldsymbol{x}_\ell)$ is the row-normalized kernel matrix split into blocks corresponding to labeled data D_L and unlabeled data D_U.

A semi-supervised KDDT f with smoothed leaf value assignment makes predictions on its training data matching the labels assigned by label propagation with kernel

$$k_f(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i \in \text{leaves}} \frac{\mu_i(\boldsymbol{x}) \mu_i(\boldsymbol{x}')}{w_i}.$$

This is easily verified by observing that Equation 5.2, which is used to solve for leaf values, is equivalent to the system

$$f(\boldsymbol{x}_j) = \sum_{i \in \text{leaves}} \mu_i(\boldsymbol{x}_j)\boldsymbol{v}_i = \sum_{\ell \in D_L} k_f(\boldsymbol{x}_j, \boldsymbol{x}_\ell)\boldsymbol{y}_\ell + \sum_{\ell \in D_U} k_f(\boldsymbol{x}_j, \boldsymbol{x}_\ell) f(\boldsymbol{x}_\ell)$$

for $i \in D_U$. Noting that $\sum_{\ell \in D} k_f(\boldsymbol{x}_j, \boldsymbol{x}_\ell) = 1$ for all $j \in D$, the above is equivalently $\boldsymbol{Y}_U = \boldsymbol{K}_{UU}\boldsymbol{Y}_U + \boldsymbol{K}_{UL}\boldsymbol{Y}_L$ where $\boldsymbol{Y}_U = f(\boldsymbol{X})$ and $\boldsymbol{K}$ is defined by k_f.

As a result, we can view this method as label propagation on a partition of the domain rather than the data themselves. A finer partition (larger tree) yields a more expressive model, but is potentially more vulnerable to the curse of dimensionality of kernel-based methods.

Robust leaf value assignment

Another leaf value assignment strategy uses graph min-cut to maximize a notion of adversarial robustness, that is, distance from the decision boundary, thereby placing the boundary in low-density regions. This is based on the innate adversarial robustness of KDDTs via the theory of randomized smoothing as explored in-depth in Section 3.4. In this sense, the predicted value of a KDDT is directly linked to the robustness of the prediction. For example, for Gaussian smoothing, the robust radius at $\boldsymbol{x}$ is lower bounded by $r = \sigma\Phi^{-1}(f_{\max}(\boldsymbol{x}))$, where Φ^{-1} is the inverse Gaussian CDF, σ is the standard deviation of the smoother, and $f_{\max}(\boldsymbol{x})$ is the highest predicted class probability at $\boldsymbol{x}$ [38]. By maximizing $\sum_{j \in D} f_{\max}(\boldsymbol{x}_j)$, we maximize a robustness objective $\sum_{j \in D} \Phi(r_j/\sigma)$, where each r_j lower bounds the robust radius for sample j. A similar notion exists for box kernels and L1 robustness; see Section 3.4 for details. An objective like this makes sense because, regardless of practical concerns, we cannot maximize the robust radii $\sum_j r_j$ directly since it is not upper-bounded. Thus we define a semi-supervised objective $\sum_{j \in D_L} \boldsymbol{y}_j^\top f(\boldsymbol{x}_j) + \sum_{j \in D_U} f_{\max}(\boldsymbol{x}_j)$ for the model to be *robustly correct* on labeled data and *robust* on unlabeled data.

This is still difficult to optimize, so we relax the objective to instead maximize $\sum_{j \in D_L} \boldsymbol{y}_j^\top f(\boldsymbol{x}_j) + \sum_{j \in D_U} f(\boldsymbol{x}_j)^\top f(\boldsymbol{x}_j)$, which lower bounds the original objective and is equal at the extremes $f_{\max}(\boldsymbol{x}) = 1$ and $f_{\max}(\boldsymbol{x}) = f_j(\boldsymbol{x})\ \forall j \in [q]$. This objective is equivalently written as $\text{Tr}(\boldsymbol{V'}^T \text{diag}(\boldsymbol{w})\boldsymbol{V} + \boldsymbol{V}^\top \boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{V})$ with $\boldsymbol{V}$ and $\boldsymbol{V'}$ as in Equation 5.2, $\text{diag}(\boldsymbol{w})$ a matrix with vector $\boldsymbol{w} = (w_i)_{i \in \text{leaves}}$ on the diagonal and 0 elsewhere, and $\boldsymbol{M}$ the membership matrix $\boldsymbol{M}_{j,i} = \mu_i(\boldsymbol{x}_j)$. Since $\boldsymbol{M}^T\boldsymbol{M}$ is positive semidefinite, the objective is convex. Moreover, each row of $\boldsymbol{V}$ must sum to 1 to represent a valid probability distribution over the class labels. This constitutes the maximization of a convex function subject to linear constraints, so at least one maximizer must exist at a corner point; that is, there is some maximizing $\boldsymbol{V}$ with only one nonzero element, which has value 1, in each row.

Let $\boldsymbol{A} = \boldsymbol{M}^T\boldsymbol{M}$. For each leaf i, let c_i denote the index such that $V_{i,c_\mathbb{L}} = 1$, that is, c_i is the index of the class predicted at leaf i. Rewrite maximization of the

objective as minimization of the following loss.

$$\begin{aligned}\ell(\boldsymbol{V}) =& |D_L| + |D_U| - \operatorname{Tr}(\boldsymbol{V'}^\top \operatorname{diag}(\boldsymbol{w})\boldsymbol{V} + \boldsymbol{V}^\top \boldsymbol{A}\boldsymbol{V}) \\ =& \sum_i w_i \sum_c V'_{i,c} + \sum_{i,i'} A_{i,i'} - \sum_i w_i V'_{i,c_i} - \sum_{i,i'} \mathbf{1}\{c_i = c_{i'}\} A_{i,i'} \\ =& \sum_{i,c} \mathbf{1}\{c \neq c_i\} w_i V'_{i,c} + \sum_{i,i'} \mathbf{1}\{c_i \neq c_{i'}\} A_{i,i'}.\end{aligned}$$

We can construct a graph such that this loss is the value of a k-terminal cut, also called a multiway cut. This is a generalization of the min-cut problem where there may be many terminals, whereas standard min-cut has only two—a source and a sink. Given nodes $N = \{s_1, \dots, s_k, n_1, n_2, \dots\}$, a k-terminal cut is a partition of the nodes into k sets $C_1, \dots, C_k$ such that $s_1 \in C_1$, $s_2 \in C_2$, etc. The value of the cut is defined as the total weight of edges removed to separate the nodes into sets: $\sum_{C \neq C'} \sum_{n \in C} \sum_{n' \in C'} w(n, n')$.

The graph is constructed as follows. For each class c, define node s_c, and for each leaf i, define node n_i. Set edge weights $w(s_c, n_i) = w_i V'_{i,c}$ and $w(n_i, n'_i) = A_{i,i'}$. Then, given a cut $C_1, \dots, C_k$, for each i, set $V_{i,c_i} = 1$ such that $n_i \in C_{c_i}$. Then the value of the cut is exactly $\ell(\boldsymbol{V})$, and so the minimum such cut provides leaf values $\boldsymbol{V}$ that minimize the loss.

For $k = 2$, the problem is simply called *min cut* or *max flow*, and there are many algorithms to solve it with various complexity. The problem of finding the minimum k-terminal cut for $k \geq 3$ is NP-Hard; however, a simple heuristic achieves an approximation of $2 - 2/k$ by solving the standard minimum cut problem in a one-vs-rest fashion k times [42]. We use this heuristic.

In practice, a model fitted in this way may sacrifice correctness on labeled data for greater robustness on unlabeled data; in the worst case, if there are relatively few labeled samples and the unlabeled data is not easily separable, it is possible that the most robust model always predicts the global majority label. In this case, simply increase the weight of the labeled samples relative to the unlabeled samples.

Model selection

There are two important hyperparameters that must be selected: the tree size and the kernel bandwidth. One may wish to select these automatically to maximize performance. We find that, despite the limited number of labeled data in the semi-supervised setting, cross-validation of a supervised metric is the best approach for hyperparameter selection. Rather than accuracy, we select using Mean Absolute Error (MAE) since it is less coarse when the number of labeled data is small. For classification, MAE is $\frac{1}{n}\sum_i 1 - \boldsymbol{y}_i^\top f(\boldsymbol{x}_i)$. To tune tree size, one can simply grow the tree to the largest size and prune it back down to get smaller sizes, so the tree need only be grown once. The tree must be re-fit from scratch, however, for different bandwidths. Since the tree growth phase dominates the run time, for experiments,

we grow the tree just once on all data, then perform k-fold cross validation over the labeled part of the data by redoing the leaf assignment phase only. This is cost effective, but if cost is not a concern, it is of course better to completely re-fit the model during cross validation.

Ensembles

and while KDDTs can generalize better than standard decision trees, they also benefit from ensemble approaches; see Section 2.3.6. For some ensemble methods, the extension to SSL is straightforward. Random forests [26], perhaps the most popular tree ensemble, use bagging, where each model is trained on a bootstrap sample of the data, and at each split, only a random subsample of features is considered. We follow the precedent of [109] and adapt this to semi-supervised learning by bootstrap sampling from the union of the labeled and unlabeled data, then train the semi-supervised trees on each bootstrap sample with feature subsampling. Another popular tree ensemble algorithm, ExtraTrees [68], only changes the tree fitting process by selecting thresholds at random, and can be used for SSL without modification. Boosted tree ensembles, however, including algorithms such as AdaBoost [63] and XGBoost [35], use a supervised loss function, so the adaptation to SSL is not so straightforward. We leave this topic to future work.

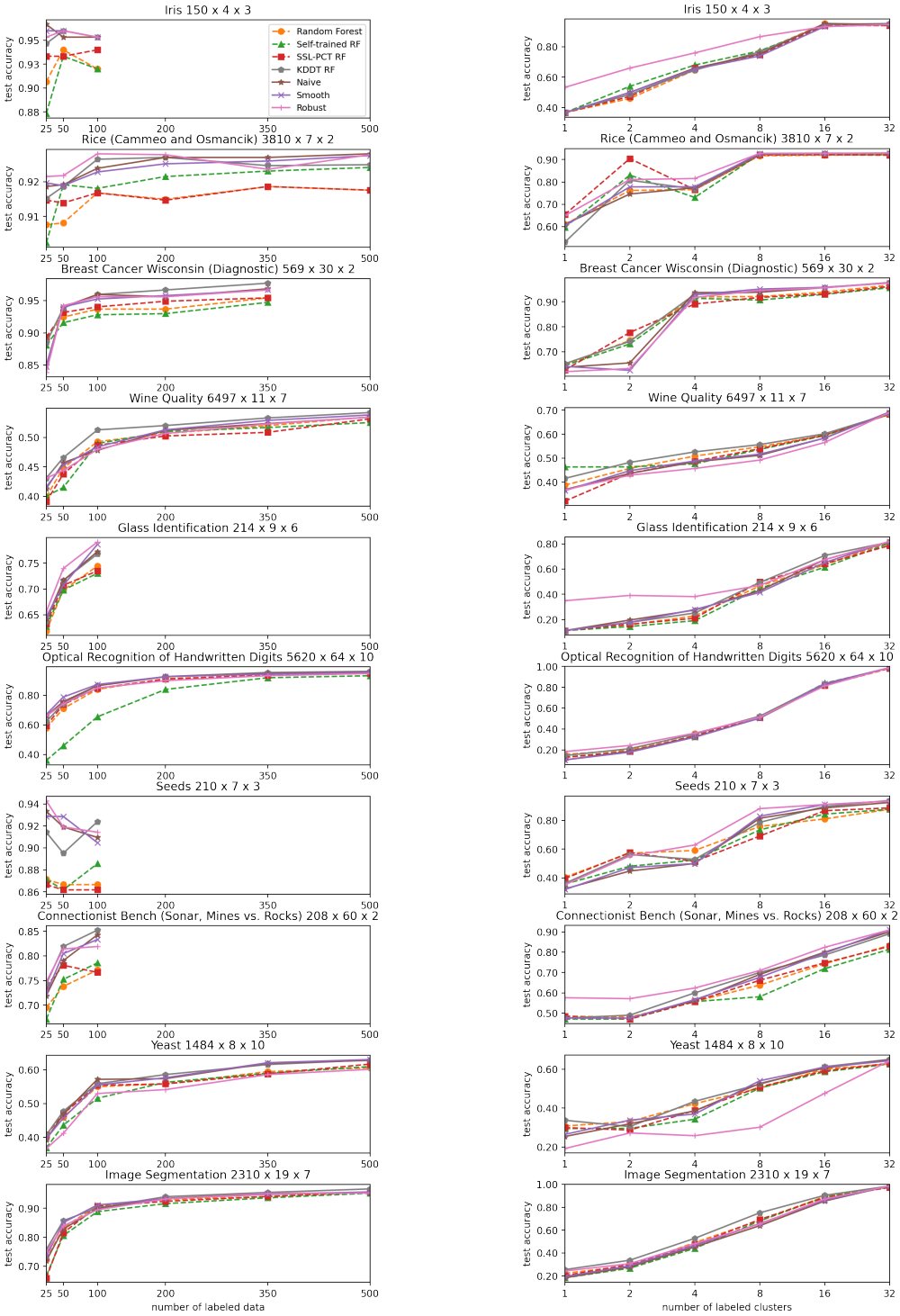
Computational cost

The complexity of tree growth with the semi-supervised criterion is the same as for standard KDDT fitting; see Section 2.3.3 for details. In practice, we observe that tree growth dominates the total run time.

For leaf value assignment, computing a dense leaf similarity graph is quadratic in the number of leaves; however, when using a bounded or truncated kernel, the sparse membership of data in leaves makes the leaf graph itself also sparse, so the computation should be done with sparse matrices. Next, the complexity of computing leaf values is the same as the related graph-based methods with sparse graphs, namely label propagation and min-cut SSL, but the size of the graph is the number of leaves instead of the number of data. Therefore, for fixed tree size, our methods' total cost scales linearly with the number of data, better than purely graph-based SSL. Using sparse representations, we observe that leaf value assignment is a relatively small portion of the total run time.

5.1.3 Experiments

We benchmark our methods in random forests against baselines including supervised random forests, self-trained random forests, and SSL-PCT random forests [109]. Our methods include the two proposed leaf assignment strategies as well as fully supervised KDDT random forests and a "naive" strategy where we use our semi-supervised tree



(a) Random selection of labeled data. (b) Cluster-based selection of labeled data.

Figure 5.4: Comparison of performance of SSL methods implemented as random forests. Ours are solid lines, and baselines are dashed.

growth, but typical supervised leaf value assignment; this method necessarily constrains tree growth such that each leaf contains at least one labeled sample. For data, we select the most viewed data sets from the UCI Machine Learning Repository [49] at the time of writing, with filters to classification tasks and numerical features, skipping any that have missing values or actually contain categorical features. For each, we repeat the following 5 times with different random seeds: randomly split the data into 80% train, 20% test, then sample various subsets of the training data to use as labeled, with the rest used as unlabeled. To sample labeled data, we use two strategies. The first, as in [109], selects a fixed number of data uniformly at random to be labeled, so that the labeled and unlabeled data are from the same distribution. The second clusters the data into 32 clusters using k-means, then randomly selects a fixed number of clusters to be labeled. This strategy has the labeled data coming from a different distribution than unlabeled data and tests a SSL method's ability to propagate labels across regions of unlabeled data. Hyperparameter and data preprocessing details are in Appendix C.7. Figure 5.4 shows the results.

For uniformly random selection of labeled data (Figure 5.4a), the KDDT-based methods outperform the crisp tree methods; however, our semi-supervised methods are very similar to supervised KDDTs, sometimes performing slightly better and sometimes slightly worse. Our robust method has the best average rank and the highest rate of achieving rank 1 when the number of labeled data is 25 or 50. For the other numbers of labeled data, the supervised KDDT forest is the best at both. The natural conclusion is that KDDTs are simply very good at generalizing with low training sample count, to the point that none of the tested SSL methods offer consistent or substantial improvement.

However, for cluster-based selection of labeled data (Figure 5.4b), there is a greater need for semi-supervised modeling, and accordingly, there are many cases where the semi-supervised methods show a clear advantage over supervised methods. In particular, our proposed method with robust leaf value assignment stands out, with a clear advantage over other methods on multiple cluster counts for several data sets, including Iris, Glass Identification, Optical Recognition of Handwritten Digits, Seeds, and Connectionist Bench. This may be because, of our two methods, the robust leaf value assignment is better able to propagate labels far across regions of unlabeled data, whereas the smoothed method tends to have the confidence of predictions fade over long distance from labeled data. The latter trait may be desirable in some applications. The robust method has the best average rank when the number of clusters is 1, 2, or 4, and the highest rate of achieving rank 1 when the number of clusters is 1, 2, 4, or 8. For the other cluster counts, the supervised KDDT random forest achieves the respective bests, except for the 32 cluster case where all data is labeled and our semi-supervised methods are equivalent to standard supervised KDDT fitting, so the differences in rank of KDDT-based methods are random.

There are also cases where a semi-supervised method performs much worse than supervised methods; see, for example, self-training on Optical Recognition of Hand-

written Digits with uniformly sampled labeled data, or our robust method on Yeast with cluster-based label data selection. Semi-supervised methods are based on an assumption about the distribution of data and labels, and if the assumptions do not adeqeuately describe the data, it is possible that the method reduces performance compared to supervised methods, so it is expected to observe this on some data sets.

5.1.4 Discussion

We proposed a novel intrinsic method for semi-supervised learning of decision trees. It efficiently constructs tree structures using a semi-supervised impurity based on fuzzy splitting. Then, it uses graph-based semi-supervised concepts to propagating label information across similar leaves; however, unlike graph-based approaches, it scales well with the size of the data and produces a predictive model that has all the practical benefits of decision trees. It also implements different assumptions from previous tree-based SSL methods and, as a result, performs well on many data sets where those methods fall short. Our experiments, in particular, show superior performance from the robust leaf assignment strategy, and additionally demonstrate that just using supervised KDDTs is competitive with dedicated SSL methods in many cases.

The method is limited in that it implements particular kinds of assumptions for SSL, which will not usefully describe all data sets. It also requires some hyperparameter tuning for good performance. Future work may investigate the proposed approach for regression tasks and consider its adaptation to other tree-based models, such as boosted ensembles.

5.2 Decentralized Federated Learning in Function Space

Federated Learning (FL) trains models across a network of devices or silos, called clients, that provide data. Examples include smartphones, edge servers, IoT devices, or institutional data centers. Unlike traditional centralized approaches, where data is collected and processed on a central server, FL assumes that the data must remain on the clients without being transmitted or shared. This addresses a wide range of real-world limitations that would otherwise prevent the application of machine learning or render it difficult, such as prohibitions on data sharing, privacy concerns, or data storage or transfer limitations.

Broadly speaking, FL methods are either centralized or decentralized. Centralized methods make use of a central server that does not contribute any data, but does aggregate information from clients, often in the form of model updates, and coordinates the learning process. For example, the most prominent FL algorithm, federated averaging (FedAvg) [125], averages the weights of client models on the central server, then

sends the average model back to clients to continue training. Centralized approaches are simple, relatively easy to implement and analyze, and widely used. However, a central server can be detrimental in some situations. It is a bottleneck for communication and computation; as a single point of failure, it is a vulnerability in the system; and it may be impractical or infeasible in cases involving unreliable networks, field communications, or particular privacy concerns.

In contrast, Decentralized Federated Learning (DFL) methods such as decentralized federated averaging (DFedAvg) [164] do not depend on a central server and rely instead on peer-to-peer communication. Such approaches can alleviate the bottleneck, improve robustness to system failures, and enhance practicality in resource-limited settings. Comparing and optimizing various frameworks for decentralized FL, each with unique features and tradeoffs such as privacy, fairness, rates of convergence, and robustness, is an ongoing area of research [14].

In this section, we adopt the functional view of models applied throughout this thesis to lay the foundations for a new approach to DFL that views the optimization and enforcement of consensus of distributed models in function space. In particular, our approach is an iterative process in which clients exchange models with their neighbors and then learn a model on their data with a Function Space Regularization (FSR), which penalizes disagreement between its model and its neighbors' models. This update can be analyzed as a proximal gradient method, a well-studied algorithm for convex optimization. The regularization requires computing inner products and norms of functions, which is expensive in general. However, where it can be efficiently applied, it unlocks a few advantages and useful capabilities not available when optimizing from a strictly parametric perspective:

- Support for non-parametric models. Non-parametric models often offer advantages such as less burden of architecture design and hyperparameter choice, efficient optimization with no special hardware requirements, small memory footprint, or reliable performance on certain problems. Existing FL methods generally aggregate models or enforce agreement by some operation on their parameters, and are therefore incompatible with non-parametric models.

- Broadly applicable theoretical convergence. In parametric machine learning, the objective function is often non-convex in the parameters, making convergence analysis challenging. However, virtually all learning objectives are convex in function space, guaranteeing convergence of the federated learning iteration so long as the local learning problems are solved near-optimally.

- Robustness to client heterogeneity. A key limitation of FL algorithms is their vulnerability to client heterogeneity [198, 186, 14], particularly to differences in their local data distributions. The negative effect of client heterogeneity on the convergence of FedAvg is well-studied both theoretically and empirically [115, 183, 114, 99]. Our function space approach is relatively robust to client heterogeneity, performing even in the most adverse cases possible.

- Low communication cost. Another key limitation of FL algorithms is the potentially huge communication cost [198, 186, 14]. Convergence may require many iterations, and it is common practice to use very large models due to their favorable convergence properties for local learning. Thanks to the convexity of the federated component of the learning process, our function space method can learn a performant model in relatively few iterations. In addition, where they are applicable, non-parametric models can provide more options to minimize communication requirements at each iteration; for example, decision trees have adaptive size and a very small minimal representation.

5.2.1 Related Work: Decentralized Federated Learning

For a recent review of DFL, see [14]. They provide an in-depth comparison between centralized and decentralized approaches, and also designate some methods as semi-decentralized. Among others, they highlight client data heterogeneity and efficiency of computation, storage, and communication as open challenges.

The prominent FedAvg algorithm has a decentralized variant called Distributed Federated Averaging with Momentum (DFedAvgM) [164]. Each client performs a fixed number of gradient updates with momentum, then broadcasts its model. Clients then average the model parameters they receive from their neighbors and begin a new round of gradient updates.

Similarly to ours, some prior methods are based on a regularization that penalizes disagreement between neighboring models. While ours penalizes distance in function space, prior work penalizes distance in parameter space. [176] propose a method of asynchronous model propagating and updates with convergence in expectation to the optimal solution for convex quadratic objectives. For more general convex objectives, they propose a similar method based on the Alternating Direction Method of Multipliers (ADMM) [23], a popular method for distributed convex optimization. This approach additionally communicates a set of dual variables with the same structure as the model parameters. [4] propose the Distributed Jacobi Asynchronous Method (DJAM), which similarly has clients asynchronously update and communicate models, but is based on a block-coordinate descent paradigm. They show convergence with probability 1 for strongly convex objectives and empirically demonstrate similar convergence rates to the ADMM variant of [176] without the need to tune a hyperparameter. These methods are presented as learning personalized models since the regularization encourages similarity of neighbor models, but does not require exact consensus, with the tradeoff of agreement and personalization determined by a hyperparameter. Our approach can be used similarly, but we generally seek an optimal consensus model.

Among such penalty-based methods, with a similar optimization algorithm to ours, [12] uses the Distributed Proximal Gradient Method (DPGM) for the synchronous distributed optimization of the sum of strongly convex smooth functions

and possibly nonsmooth convex functions of scalar input. They frame the penalized objective as a relaxation of the consensus-constrained objective and show convergence to a neighborhood of the constrained optimum, including for an inexact variant where the updates are assumed to be subject to error with limited expected magnitude.

If our method is used in parameter space instead of function space, it is similar to a synchronous variant of DJAM, or a special case of a multidimensional DPGM. We make similar assumptions with similar convergence guarantees; however, ours is applicable to a much wider range of realistic learning problems since most learning objectives are convex in function space, even if they are non-convex in parameter space. We also provide an approach for increasing the penalty to achieve convergence to the constrained solution itself.

Recently [91] survey tree-based methods in DFL. These methods rely on boosting, and sometimes other ensemble methods. Our framework is model-agnostic, and it is particularly pragmatic for single-tree models, which can maximize interpretability and minimize communication cost.

5.2.2 Proximal Gradient Method in Function Space

We first formalize the notion of learning viewed in function space, then describe the proposed framework for decentralized federated learning, its convergence properties, and its application to a few model classes.

A hypothesis $h : \mathcal{X} \to \mathcal{Y}$ is a function mapping input values to a prediction. The goal of a learning algorithm is to select an optimal hypothesis h^* from a hypothesis space $\mathcal{H}$ that minimizes an objective function or "risk" $R : \mathcal{H} \to \mathbb{R}$. The term "risk" is used to describe an expected loss, usually estimated empirically using the training data. Here for convenience we use the term "risk" and the symbol R to encompass the entire learning objective, including both the empirical risk and other components such as regularization.

We assume that the hypothesis space $\mathcal{H}$ is a (real, separable) Hilbert space equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ and associated norm $\|h\|_{\mathcal{H}}^2 = \langle h, h \rangle_{\mathcal{H}}$. A useful example is the space of L^2-integrable functions from $\mathcal{X} \subseteq \mathbb{R}^p$ to $\mathcal{Y} \subseteq \mathbb{R}^q$, where the commonly associated inner product is

$$\langle h, g \rangle_{\mathcal{H}} = \int_{\mathcal{X}} \langle h(x), g(x) \rangle_{\mathcal{Y}} \, dx \tag{5.3}$$

and $\langle \cdot, \cdot \rangle_{\mathcal{Y}}$ is the usual vector inner product on $\mathbb{R}^q$. This is by no means the only possible Hilbert space or inner product, but it is simple and practical for learning applications, so it is the main focus of the empirical evaluation in this work. When h is a model, this inner product may be difficult to compute exactly or even approximate efficiently; we discuss this challenge in Section 5.2.4.

Though most model classes do not exactly form a Hilbert space, many are reasonable to analyze as such. For instance, neural networks and decision trees have

a "universal approximator" property; they can approximate any function in L^2 with arbitrary precision, given enough size. Other classes, such as linear models, are themselves a subspace of L^2.

In a decentralized federated learning setting, each client $i \in [n]$ has a sample of training data that define its local risk R_i, which it can use to learn a local model h_i. We focus primarily on consensus learning, where the goal is to select a consensus model h^* that minimizes the aggregate risk

$$\bar{R}[h] = \sum_i R_i[h]. \tag{5.4}$$

It is assumed that the clients may not communicate data, so each risk R_i may only be evaluated at client i, and clients communicate by exchanging models. Thus we write an equivalent optimization problem in terms of local models with an agreement constraint

$$\begin{aligned} H^* = \underset{\boldsymbol{h} \in \mathcal{H}^n}{\arg\min} & \sum_{i \in [n]} R_i[h_i] \\ \text{s.t. } & h_i = h_j \ \forall i, j \end{aligned} \tag{5.5}$$

where $H^* \subseteq \mathcal{H}^n$ is the set of optimal consensus models and $\boldsymbol{h}^* = (h^*, \ldots, h^*) \in H^*$.

Let $L \in \mathbb{R}^{n \times n}$ be a symmetric Laplacian of the communication graph with $L_{i,j} < 0$ if i and j are neighbors that may directly communicate and $L_{i,i} = -\sum_{j \neq i} L_{i,j}$. To make optimization possible in a distributed network, we relax the agreement constraint in (5.5) to a disagreement penalty $\frac{1}{2}\lambda \sum_{i \neq j} -L_{i,j} \|h_i - h_j\|^2$ for some penalty coefficient $\lambda > 0$. The relaxed optimization problem can then be written

$$\tilde{H} = \underset{\mathbf{h} \in \mathcal{H}^n}{\arg\min} \sum_{i \in [n]} R_i[h_i] + \frac{1}{2}\lambda \langle \mathbf{h}, \mathbf{L}\mathbf{h} \rangle_{\mathcal{H}^n} \tag{5.6}$$

where $\mathbf{L}$ is a positive operator $(\mathbf{L}\mathbf{h})_i = \sum_j L_{ij} h_j$ on $\mathcal{H}^n$, which is itself a Hilbert space with inner product $\langle \mathbf{h}, \mathbf{g} \rangle_{\mathcal{H}^n} = \sum_i \langle h_i, g_i \rangle_{\mathcal{H}}$, and $\tilde{H}$ is the solution set.

To solve this, we use an iterative process initialized by each client minimizing its local risk: $h_i^{(0)} = \arg\min_h R_i[h]$. Then the clients exchange models with their neighbors on the network and the proximal gradient method, a convex optimization algorithm to minimize the sum of a smooth, differentiable function and a possibly nonsmooth function, yields the separable iterative update

$$h_i^{k+1} = \underset{h_i \in \mathcal{H}}{\arg\min} \ R_i[h_i] + \frac{1}{2\gamma} \|h_i - h_i^k\|^2 + \lambda \sum_{j \in [n]} L_{i,j} \langle h_i, h_j^k \rangle_{\mathcal{H}} \tag{5.7}$$

for proximal gradient parameter γ, which is analogous to a learning rate. This depends only on information available to client i at iteration t and is solved using a local learning algorithm augmented with a function space regularization.

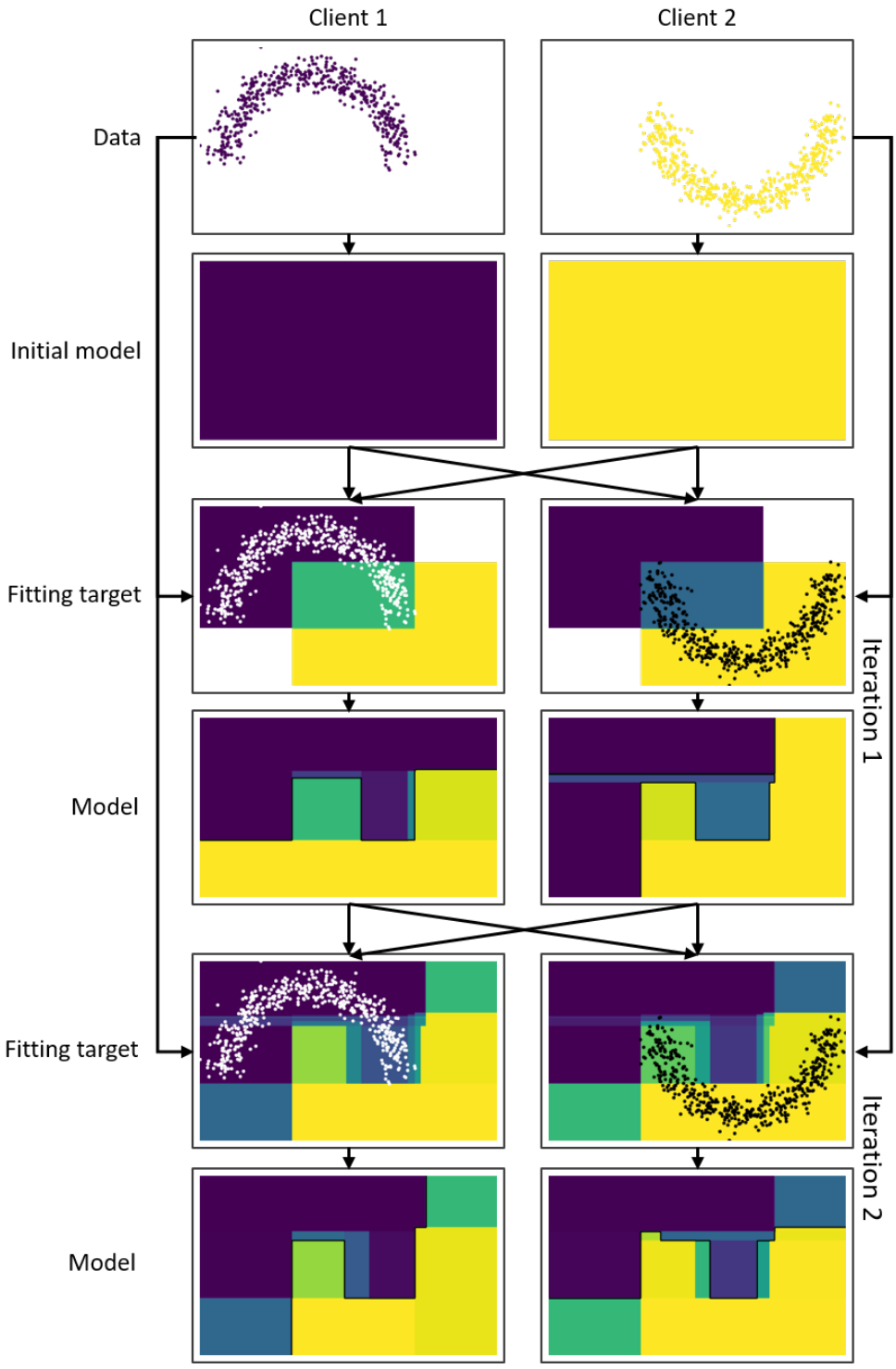


Figure 5.5: Visualization of the first two iterations of our algorithm applied to learn trees (without smoothed error) on the two moons data set, split into clients by class.

An example of decision trees learned on toy data in this way is shown in Figure 5.5. Here we use the inner product (5.3), which penalizes disagreement uniformly over the bounding box. The data is split into two clients by class, which is considered a very difficult case in FL due to high client heterogeneity. As a result, each client's initial model, which is just fit to its own data, is a single-leaf tree that always makes the same prediction. Then the first iteration begins: the clients exchange their initial models and bounding boxes and fit again. Essentially, whenever a client fits a model, it is targeting the average of its own previous model and its neighbors' previous models, except where its own local data is present. The influence of local data causes changes to the model that are then propagated to its neighbors in the next iteration, allowing them to learn from other clients' data without actually seeing it directly. In this simple case, just one iteration achieves perfect global accuracy at both clients. As the iterations proceed, the two clients' models become more similar.

5.2.3 Convergence Analysis

We analyze the convergence of iteration (5.7) and the proximity of its solution set $\tilde{H}$ to the consensus solution set H^*. We then discuss some practical considerations related to the theoretical convergence. See Appendix B for omitted proofs.

Notation.

- On a vector in Hilbert space such as $\mathcal{H}$ or $\mathcal{H}^n$, $\|\cdot\|$ denotes the norm induced by the corresponding inner product. On a matrix or linear operator, $\|\cdot\|$ denotes its spectral norm.
- Given v and nonempty set S, $d(v, S) = \inf_{s\in S}\|v - s\|$ is the shortest distance from v to S. For sets S and T, $d(S, T) = \inf_{s\in S, t\in T}\|s-t\|$ is the shortest distance between them.
- $\mathbf{E}$ is the operator such that $(\mathbf{Eh})_i = \frac{1}{n}\sum_j h_j$ for all i, projecting $\mathbf{h} \in \mathcal{H}^n$ to consensus.
- Given a linear operator $\mathbf{L}$, $\sigma(\mathbf{L})$ denotes the spectrum of $\mathbf{L}$. We say $\mathbf{L}$ has spectral gap when there exists $\nu > 0$ s.t. $\sigma(\mathbf{L}) \cap (0, \nu) = \emptyset$.
- Given a linear operator $\mathbf{L}$, we say $\mathbf{L}$ is positive if $\mathbf{L}$ is positive semi-definite and self-adjoint.

Assumption 5.1. The client risks R_i are convex and have a minimum on $\mathcal{H}$.

While convexity is a strong assumption in parameter space, virtually all commonly used loss functions are convex in function space, making this broadly applicable.

Assumption 5.2. The communication graph represented by L is connected.

The smallest eigenvalue of a graph Laplacian is always zero. Let ν be the second-smallest eigenvalue of L; ν is known as the algebraic connectivity of the communication graph and Assumption 5.2 implies that $\nu > 0$. Moreover, it is straightforward to show that $\sigma(\mathbf{L})$ consists of the eigenvalues of L, so $\mathbf{L}$ has spectral gap and $\|\mathbf{L}\| = \|L\|$.

Assumption 5.3. The proximal gradient parameter γ satisfies $0 < \gamma < \frac{1}{\lambda\|L\|}$.

Then the proximal gradient method converges weakly to a solution of the relaxed optimization problem (5.6) [39, Theorem 3.4 (i)]. However, under stronger assumptions, we can show fast convergence to a neighborhood of the solution set of the original constrained optimization problem (5.5).

Assumption 5.4. For each i, R_i is convex quadratic; in particular, there exist positive operators $A_i : \mathcal{H} \to \mathcal{H}$, $a_i \in \mathcal{H}$, and $\alpha_i \in \mathbb{R}$ such that $R_i[h] = \frac{1}{2}\langle h, A_i h\rangle + \langle a_i, h\rangle + \alpha_i$. Moreover, A_i have spectral gap at least $\mu > 0$ and commute with each other.

Assumption 5.4 implies that R is quadratic, in particular, $R[\mathbf{h}] = \frac{1}{2}\langle \mathbf{h}, \mathbf{A}\mathbf{h}\rangle + \langle \mathbf{a}, \mathbf{h}\rangle + \alpha$ with (self-adjoint) $\mathbf{A}$ satisfying $(\mathbf{A}\mathbf{h})_i = A_i h_i$, $\mathbf{a}_i = a_i$, and $\alpha = \sum_i \alpha_i$. Thus R is differentiable and ∇R is Lipschitz continuous with constant $\|\mathbf{A}\|$, and $\sigma(\mathbf{A}) = \bigcup_i \sigma(A_i)$, so $\sigma(\mathbf{A}) \cap (0, \mu) = \emptyset$. The commutativity of A_i, A_j further implies $\sigma(\sum_i A_i) \cap (0, \mu) = \emptyset$. Lemma 5.1 shows that these gaps in the spectra establish a growth rate of the respective quadratic functions.

Lemma 5.1. *Let φ be a quadratic function $\varphi(h) = \frac{1}{2}\langle h, Ah\rangle + \langle a, h\rangle + \alpha$ on a Hilbert space $\mathcal{H}$ with a minimum value φ^*. If A is positive with spectral gap $\sigma(A) \cap (0, c) = \emptyset$, then*

$$\|\nabla\varphi[h]\| \geq c\, d(h, \arg\min \varphi) \tag{5.8}$$

$$\varphi[h] \geq \varphi^* + \frac{1}{2} c\, d^2(h, \arg\min \varphi) \tag{5.9}$$

for any $h \in \mathcal{H}$.

Next, Lemma 5.2 establishes the existence of spectral gap in the operator defining the quadratic objective of the relaxed optimization problem (5.6).

Lemma 5.2. *There exists some $c > 0$ such that $\sigma(\mathbf{A} + \lambda\mathbf{L}) \cap (0, c) = \emptyset$.*

With these, Theorem 5.1 establishes linear convergence of iteration (5.7).

Theorem 5.1. *The distance of the clients' local hypotheses to the relaxed solution set is bounded by*

$$d(\mathbf{h}^{k+1}, \tilde{H}) \leq \frac{1}{\sqrt{1 + \gamma c}}\, d(\mathbf{h}^k, \tilde{H}). \tag{5.10}$$

Proof. Let $\varphi[\mathbf{h}] = R[\mathbf{h}] + \frac{1}{2}\lambda\langle \mathbf{h}, \mathbf{L}\mathbf{h}\rangle$ denote the minimization objective of the relaxation (5.6) and $\varphi^* = \min_{\mathbf{h}\in\mathcal{H}^n} \varphi[\mathbf{h}]$. By [13, Lemma 2.3, Remark 2.1], we have the following bound for any $\tilde{\mathbf{h}} \in \tilde{H}$.

$$\begin{aligned}\varphi[\mathbf{h}^{k+1}] - \varphi^* &\leq -\frac{1}{2\gamma}\|\mathbf{h}^{k+1} - \mathbf{h}^k\|^2 - \frac{1}{\gamma}\langle \mathbf{h}^k - \tilde{\mathbf{h}}, \mathbf{h}^{k+1} - \mathbf{h}^k\rangle \\ &= \frac{1}{2\gamma}\left(\|\mathbf{h}^k - \tilde{\mathbf{h}}\|^2 - \|(\mathbf{h}^{k+1} - \mathbf{h}^k) + (\mathbf{h}^k - \tilde{\mathbf{h}})\|^2\right) \\ &= \frac{1}{2\gamma}\left(\|\mathbf{h}^k - \tilde{\mathbf{h}}\|^2 - \|\mathbf{h}^{k+1} - \tilde{\mathbf{h}}\|^2\right).\end{aligned}$$

Next Lemmas 5.1 and 5.2 imply that $\varphi[\mathbf{h}^{k+1}] \geq \varphi^* + \frac{1}{2}cd^2(\mathbf{h}^{k+1}, \tilde{H})$. Applying this to the above inequality,

$$\frac{1}{2}cd^2(\mathbf{h}^{k+1}, \tilde{H}) \leq \frac{1}{2\gamma}\left(\|\mathbf{h}^k - \tilde{\mathbf{h}}\|^2 - \|\mathbf{h}^{k+1} - \tilde{\mathbf{h}}\|^2\right).$$

Since $d^2(\mathbf{h}^k, \tilde{H}) = \inf_{\mathbf{h}\in\tilde{H}}\|\mathbf{h}^k - \mathbf{h}\|^2$, for any $\epsilon > 0$, there exists $\tilde{\mathbf{h}} \in \tilde{H}$ such that $\|\mathbf{h}^k - \tilde{\mathbf{h}}\|^2 \leq d^2(\mathbf{h}^k, \tilde{H}) + \epsilon$. Then

$$\frac{1}{2}cd^2(\mathbf{h}^{k+1}, \tilde{H}) \leq \frac{1}{2\gamma}\left(d^2(\mathbf{h}^k, \tilde{H}) + \epsilon - d^2(\mathbf{h}^{k+1}, \tilde{H})\right)$$

for any $\epsilon > 0$, and the claim follows. □

Now Theorem 5.2 shows that this solution is within $O(1/\lambda)$ of the consensus optimal solution set H^*.

Theorem 5.2. *For a given λ, for any $\tilde{\mathbf{h}} \in \tilde{H}$,*

$$d(\tilde{\mathbf{h}}, H^*) \leq \frac{\|\mathbf{A}\|}{\lambda\nu}\sqrt{\frac{\|\mathbf{A}\|}{\mu}}\left(1 + \frac{n\|\mathbf{A}\|}{\mu}\right) d(H^*, \arg\min R) \in O(1/\lambda). \tag{5.11}$$

Together, these theorems tell us that the iteration converges quickly to the relaxed optimum, and moreover, that as we increase the penalty coefficient λ, the relaxed optimum approaches the consensus optimum, that is, the solution to the original problem (5.5) of finding a globally optimal hypothesis.

Smoothed Squared Error

While quadratic loss functions are common, using a quadratic loss is not sufficient to satisfy Assumption 5.4 (quadratic risk) because typical pointwise empirical risk cannot be expressed as quadratic using the inner product (5.3). Fortunately, the smoothed risk (2.1) induced by the uncertain input interpretation introduced in this

thesis, combined with a quadratic loss, does satisfy Assumption 5.4 when the kernel k has a minimum nonzero value.

In particular, suppose each client i has data $\boldsymbol{x}_j \in \mathbb{R}^p$, $\boldsymbol{y}_j \in \mathbb{R}^q$, $j \in [N_i]$, and let $k : \mathbb{R}^p \times \mathbb{R}^p$ be a symmetric smoothing kernel that integrates to 1 over the domain. Then define risks as the smoothed sum squared error

$$\begin{aligned}
R_i[\mathbf{h}_i] &= \sum_j \mathbb{E}_{\mathbf{x}\sim k(\cdot,\boldsymbol{x}_j)} \|\mathbf{h}_i(\boldsymbol{x}_j) - \boldsymbol{y}_j\|^2 \\
&= \sum_j \int_z k(\boldsymbol{z}, \boldsymbol{x}_j) \|\mathbf{h}_i(\boldsymbol{z}) - \boldsymbol{y}_j\|^2 \, d\boldsymbol{z} \\
&= \sum_j \int_z k(\mathbf{z}, \boldsymbol{x}_j) (\mathbf{h}_i(\boldsymbol{z})^\top \mathbf{h}_i(\boldsymbol{z}) + \boldsymbol{y}_j^\top \boldsymbol{y}_j - 2\mathbf{h}_i(\boldsymbol{z})^\top \boldsymbol{y}_j) d\boldsymbol{z} \\
&= \frac{1}{2} \left\langle \mathbf{h}_i, 2 \sum_j k(\cdot, \boldsymbol{x}_j) \mathbf{h}_i \right\rangle + \left\langle -2 \sum_j k(\cdot, \boldsymbol{x}_j) \boldsymbol{y}_j, \mathbf{h}_i \right\rangle + \sum_j \boldsymbol{y}_j^\top \boldsymbol{y}_j
\end{aligned}$$

where the spectrum of the positive linear operator $\mathbf{h}_i \mapsto \left(\sum_j k(\cdot, \boldsymbol{x}_j)\right) \mathbf{h}_i(\cdot)$ is the essential range of $\sum_j k(\cdot, \boldsymbol{x}_j)$. Then $\mu = 2 \min_{j,\boldsymbol{z}|k(\boldsymbol{z},\boldsymbol{x}_j)>0} k(\boldsymbol{z}, \boldsymbol{x}_j)$ and $\|\mathbf{A}_i\|$ is bounded by $\|\mathbf{A}_i\| \leq 2 \sum_j \max_{\boldsymbol{z}} k(\boldsymbol{z}, \boldsymbol{x}_j)$; if we assume $k(\cdot, \boldsymbol{x}_j)$ is the same at each j, then μ and $\|\mathbf{A}\|$ are simply twice its minimum nonzero value and up to $2N$ times its maximum, respectively, suggesting the use of uniform kernels.

Here we have intentionally defined the risk as the sum, rather than the mean, of loss values at the training samples so that, when the risks are summed over the clients as in the objective (5.6), all samples are equally weighted. This also prevents μ from depending on the number of samples.

This error smoothing is often important to achieve good performance. Without it, it is possible that, as optimization proceeds, both local risk and disagreement approach zero, but average global accuracy, that is, the accuracy on the union of all training sets, averaged over client models, does not improve. We sometimes observe this in practice when not using error smoothing, especially for moderate to high dimensional data where the coverage of the data over the domain is poor. However, for simple cases with good coverage of the domain, such as the example in Figure 5.5, it can perform just fine without error smoothing.

With Bounded Errors

Suppose update (5.7) is carried out with additive error $\mathbf{e}^k \in \mathcal{H}^n$ as

$$h_i^{k+1} = e_i^k + \arg\min_{h_i \in \mathcal{H}} R_i[h_i] + \frac{1}{2\gamma} \|h_i - h_i^k\|^2 + \lambda \sum_{j \in [n]} L_{i,j} \langle h_i, h_j^k \rangle. \tag{5.12}$$

This error can account for imperfect learning algorithms, hypothesis spaces that are not Hilbert spaces but are a ε-cover of one, etc. Incorporating this into Equa-

tion (5.10) via the triangle inequality, we have

$$d(\mathbf{h}^{k+1}, \tilde{H}) \leq \frac{1}{\sqrt{1+\gamma c}} d(\mathbf{h}^k, \tilde{H}) + \|\mathbf{e}^k\| \tag{5.13}$$

and, if $\|\mathbf{e}^k\|$ is bounded by a constant for all k, then the accumulation of error is bounded by a convergent geometric series. This means we can expect convergence close to the optimum even if the learning algorithm cannot solve (5.7) perfectly, which is the case for both neural networks and trees of bounded size.

5.2.4 Local Learning in Function Space

We next discuss how the optimization problem in iteration (5.7) can be solved by incorporating function space regularization with learning algorithms. We give a model-agnostic strategy using a Monte Carlo method to approximately compute the inner product (5.3) as well as better model-specific methods for a non-exhaustive list of prominent model classes.

Model-agnostic Approximations

The inner product (5.3) and associated norm can be estimated by a Monte Carlo method arising naturally from the equalities

$$\langle h, g \rangle_{\mathcal{H}} = m(\mathcal{X}) \mathbb{E}_{x \sim \mathcal{U}_{\mathcal{X}}} [\langle h(x), g(x) \rangle_{\mathcal{Y}}] \tag{5.14}$$

$$\|h\|_{\mathcal{H}} = m(\mathcal{X}) \mathbb{E}_{x \sim \mathcal{U}_{\mathcal{X}}} [\|h(x)\|_{\mathcal{Y}}] \tag{5.15}$$

where $\mathcal{U}_{\mathcal{X}}$ is the uniform distribution on $\mathcal{X}$ and $m(\mathcal{X})$ is the Lebesgue measure, or p-volume, of $\mathcal{X} \subseteq \mathbb{R}^p$. This, of course, demands that $m(\mathcal{X})$ is finite, but this is of little practical consequence. A similar strategy can be employed for other inner products. If the risk is based on mean squared error, as is recommended by our convergence theory, then this strategy applied to (5.7) can be reduced to simply incorporating a number of appropriately-weighted random samples from $\mathcal{U}_{\mathcal{X}}$ into the training set with labels the outputs from other models.

While this is simple and general, it is the least desirable approach overall. The quality of the approximation depends on the number of samples, and as the dimension of $\mathcal{X}$ increases, so does the number of samples required to achieve reasonable coverage. Depending on the model, it may be expensive to compute the output of several models on very many data, and some learning algorithms may not handle very large training sets efficiently. This motivates future work to improve sample efficiency, either by modifying the inner product or using a more efficient sampling strategy.

Neural Networks

Neural networks are typically trained using a minibatch gradient-based optimization algorithm. This motivates a variation of the above model-agnostic concept where

new samples are taken at each batch. Recall that we recommend a smoothed squared error loss and defining local risk as the sum, rather than mean, of loss on local data. Then at client i, the normalized loss for a batch $(x_1, y_1), \ldots, (x_b, y_b)$ is

$$\begin{aligned} &\frac{1}{bq}\sum_{i=1}^{b}\|h(x_i+\epsilon_i)-y_i\|^2 \\ &+\frac{m(\mathcal{X})}{Nb'q}\sum_{i=1}^{b'}\left[\frac{1}{2\gamma}\|h(z_i)-h_i^k(z_i)\|^2+\lambda\sum_{j\neq i}-L_{i,j}\|h(z_i)-h_j^k(z_i)\|^2\right] \end{aligned} \tag{5.16}$$

where b is the batch size, b' is the penalty batch size, N is the local training set size, each ϵ_i is sampled from $k(\cdot, x_i)$, and each z_i is sampled from $\mathcal{U}_{\mathcal{X}}$. For classification, where $\mathcal{Y} = [0,1]^q$, this scaling places the first (risk) term in $[0,1]$ and the second (penalty) term in $[0, \frac{m(\mathcal{X})}{N}(\frac{1}{2\gamma}+\lambda L_{i,i})]$.

Decision Trees

Decision trees are an ideal model class for the application of this framework. Not only are they non-parametric, making them compatible with function space but not parametric FL methods, but the KDDT fitting algorithm provides an efficient fitting mechanism for both smoothed squared error and regularization to other models.

We assume the domain $\mathcal{X}$ is a hyperrectangle, for example, a bounding box of the data. Then a tree can be represented as a collection of hyperrectangles $\mathcal{R}_i \subseteq \mathcal{X}$ for leaf nodes i and associated values $\boldsymbol{v}_i \in \mathcal{Y}$. The leaves form a partition of $\mathcal{X}$, and the tree as a function is written

$$h(\boldsymbol{x}) = \sum_i \mathbf{1}\{\boldsymbol{x} \in \mathcal{R}_i\}\boldsymbol{v}_i \tag{5.17}$$

and, for a second tree g with nodes $\mathcal{S}_j$ and values $\boldsymbol{w}_j$, inner product (5.3) is

$$\langle h, g\rangle = \sum_i \sum_j m(\mathcal{R}_i \cap \mathcal{S}_j)\langle \boldsymbol{v}_i, \boldsymbol{w}_j\rangle \tag{5.18}$$

with m the Lebesgue measure, which is easy to compute for the intersection of hyperrectangles. By traversing one tree and tracking the intersecting subtrees of the other, one can avoid computing the zero-measure terms, which are the majority if h and g are similar trees; if h and g are identical, then the number of nonzero terms is just the number of leaves. Since our algorithm penalizes disagreement, the trees are usually similar in practice. However, in the worst case, every term may be nonzero, and the computational cost is proportional to the size of h times the size of g.

Now we have that trees are a collection of hyperrectangles with associated values, and we have a means of extracting those hyperrectangles on a domain. As described in Section 2.3 and Theorem 2.2, the KDDT fitting algorithm with Gini impurity can

fit a tree to minimize squared error uniformly over hyperrectangle inputs; this is the same as normal fitting with box kernels, except that the boxes each have different dimensions from the others. Thus the KDDT algorithm is straightforwardly applied to solve (5.7) with smoothed squared error.

If using unsmoothed squared error with regularization to other trees, fitting is more complicated. By a straightforward generalization of Theorem 1 of [77], during the search for an optimal split, for each training index i and feature index $f \in [p]$, both $z_f < x_{i,f}$ and $z_f \leq x_{i,f}$, where z is an input to the tree, are candidates for the optimal decision rule. This is achievable with an alteration of the KDDT fitting algorithm, but it introduces the strange phenomenon of leaves that have zero measure, which are completely absent in the computation of function inner products. Thus two trees may have zero disagreement in function space, but make completely different predictions on the data. It can be circumvented by setting a stopping condition for tree fitting that prevent zero-measure leaves, but simply using smoothed squared error is simpler, better motivated in the context of function space disagreement regularization, and likely to result in better performance anyway, as demonstrated by the superior performance of KDDTs over DTs in Section 2.3.6.

It should also be noted that the KDDT algorithm is only useful when the choice of inner product and smoothing are compatible with its assumptions as specified in Section 2.3.3. If not, we would need a more general fitting algorithm, or lacking that, revert to model-agnostic approximations.

Linear Models

In Section 2.4.2, we establish an equivalence of error smoothing, a kind of function space regularization, to parameter regularization for linear models. Since the difference of linear models is linear, this also implies that the function space regularization of the difference of linear models is equivalent to a regularization on the difference of parameters. Thus the proposed function space DFL framework, as we apply it in this work, is equivalent to a parameter space method for linear models. For this reason, we omit linear models from our empirical evaluation.

5.2.5 Experiments

To demonstrate the proposed algorithm, we benchmark our method applied to KDDTs and a small MLP on the 12 data sets summarized in Table 2.1. To compare against the most similar parametric method, we use as a baseline a synchronous variant of DJAM which we call Parameter Space Regularization (PSR). We initially included DFedAvgM as a baseline, but since it trains only for a handful of updates per communication iteration, whereas the other methods fully train a model, it was not able to achieve meaningful performance in the 20 iterations in our experiments, so we omit it from the results.

For each data set, we randomly split it into 50% train, 50% test, then split the training data into clients by using k-means to group the features into two clusters. This is a split with high client heterogeneity. We also split the data by class, that is, such that there is one client per class, and each client sees only one class. This is the most extreme case of client heterogeneity. To define the communication graph, we sample a random ring (2-regular) graph. Additional experiment details are in Appendix C.8.

For our FSR methods, we use smoothed squared error with a box kernel with radius δ, that is, $k(\boldsymbol{z}, \boldsymbol{x}) \propto \mathbf{1}\{\|\boldsymbol{z} - \boldsymbol{x}\|_\infty \leq \delta\}$. We select λ and δ by training models with a range of values for each and selecting the ones that result in the highest final average global training accuracy. As a result, this is more of a proof-of-concept than a demonstration of best-case practical utility, and we leave the problem of efficient hyperparameter selection in a distributed setting to future work.

The results for the cluster-based data split are shown in Figure 5.6. On most data sets, we see that the FSR-based methods learn faster than PSR, sometimes reaching their best accuracy in just one or two rounds of communication. They also often outperform PSR in final accuracy. The FSR KDDT and FSR MLP predictably perform differently, with each outperforming the other about half the time. In these experiments, the tree size is not tuned by cross-validation, and the other hyperparameters are selected by training accuracy, so the trees sometimes overfit more than the MLPs.

On a few data sets, the FSR-based methods fall short in performance. The worst cases are optdigits and pendigits, image data where the number of informative features is likely to be high; this is consistent with our expectations for the limitations of the method in its current form, but ongoing efforts show promise towards improving the resilience of the method to higher-dimensional data.

The results for the class-based data split are shown in Figure 5.6. This is the most extreme possible heterogeneous split and, unsurprisingly, the PSR method is unable to learn anything in these experiments. For FSR, the results vary, but it is clearly learning in all cases, which is a significant feat for this kind of data split, especially in so few iterations. In most cases, convergence is slower compared to the cluster-based split. This is due in part to the fact that, for data sets with more than two classes, the ring graph means that it takes longer for information to propagate across all clients. It is also due to to the inherent challenge of the class-based split itself. In simple, low-dimensional examples such as Iris, our methods perform very well, still reaching good performance in few iterations. In the more challenging cases, learning is slower and the accuracy is sometimes unstable across iterations.

It is interesting to note that, despite the simple Monte Carlo method used to apply our FSR method when training the MLP (we use 1000 uniformly random samples for regularization per minibatch, which is very reasonable), stable learning is possible even on moderately-dimensional data where sampling random noise would seem not to cover the domain well. In these cases, the (randomized) error smoothing is crucial: there are cases where, without error smoothing, the disagreement penalty

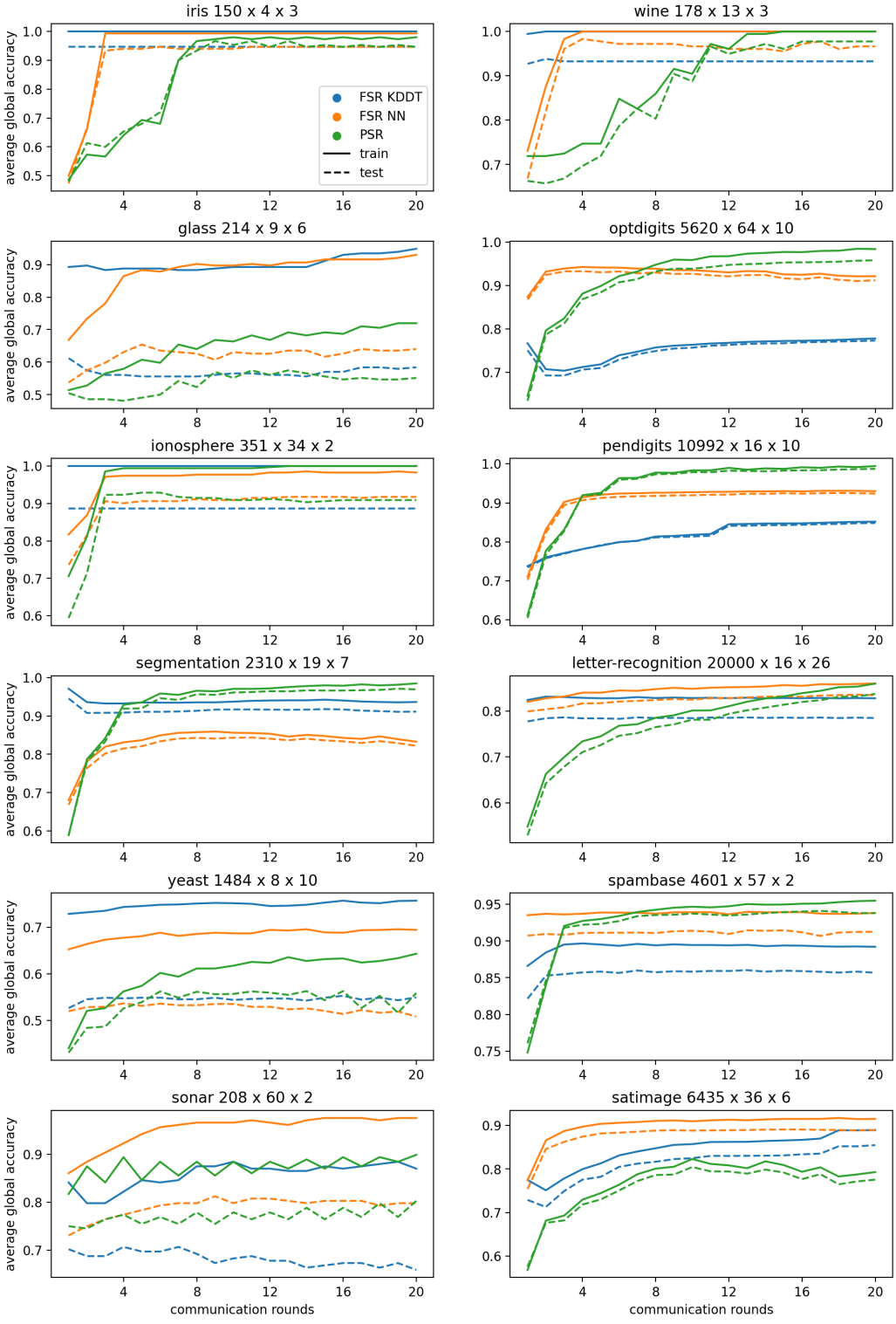


Figure 5.6: Results for FL experiments with data split by clustering.

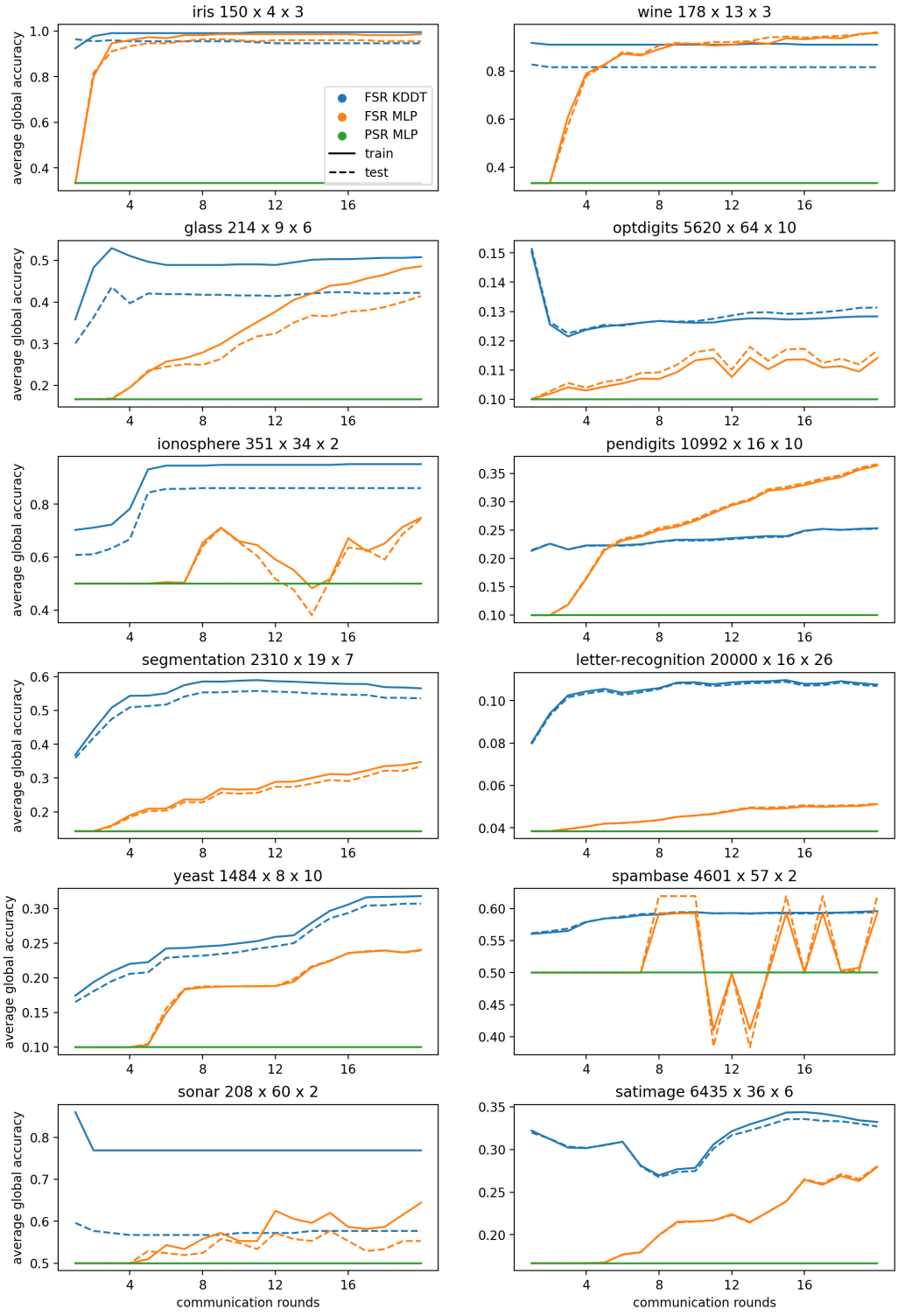


Figure 5.7: Results for FL experiments with data split by class.

does nothing to improve accuracy compared to local learning alone; however, with it, performance is very good.

5.2.6 Limitations and Future Directions

Limitations

The most important limitations of the proposed framework are as follows.

Hyperparameter selection. Our algorithm introduces hyperparameters in the form of the regularization coefficient λ and, if using smoothed error, the size and shape of the kernel. We discuss kernel choice in Section 2.5. As for λ, while the convergence theory suggests setting λ based on μ, μ may be very large; for instance, with box kernel with radius δ in p dimensions, μ is proportional to $(2\delta)^p$. Learning algorithms are not likely to work well with extremely strong penalties like this. Hyperparameter selection is further complicated by the distributed setting, where tuning efforts cost valuable iterations and bandwidth, and where computing global performance metrics without exchanging data may not be straightforward.

Computational cost. While our approach is promising for reducing the total communication cost and iterations needed to learn a model, the need to compute or approximate a function inner product can significantly increase the cost of the local learning problem at each iteration, depending on the model, inner product, connectivity of the network, and other factors. Moreover, our method, like DJAM, fully solves a local optimization problem at each iteration, and therefore has higher computational cost per iteration compared to methods like DFedAvgM that only perform a fixed number of updates at each iteration.

Curse of dimensionality. When using the basic function inner product (5.3), agreement is enforced uniformly over the domain, which might not be well-aligned with the risk for high-dimensional data, which is often assumed to lie on a low-dimensional manifold. The increasingly large penalty with the dimension as suggested by our convergence theory, as mentioned above, is one way this manifests. Moreover, the Monte Carlo approximation of the inner product, which we use with neural networks, suffers from poor sample complexity for high-dimensional domains. Further innovation is required for the proposed framework to competitively learn models for high-dimensional information.

Future Directions

This work lays the theoretical foundation and and gives an empirical proof of concept for the proposed method; there are many possible directions for future development and study.

Comprehensive application and benchmarking. This work proposes the foundation for a new paradigm of decentralized federated learning, and accordingly,

the evaluation is a proof-of-concept focusing on comparison against similar foundational methods from the parametric paradigm. It is left to future work to benchmark the method against more complex state-of-the-art methods with strategies that aim to address specific challenges, such as client heterogeneity and the necessity of asynchronous updates. Where possible, the strategies that augment parametric methods should also be incorporated into our function space approach. It is also left to future work to apply the framework to various other learning tasks such as regression, unsupervised and self-supervised learning, etc.

Asynchronous updates. Synchronous methods can only proceed as quickly as the slowest participant. In a dynamic or unreliable network with clients of varying computational power and availability, an asynchronous approach is preferable. Asynchronous methods are common in the literature, and several of the most related methods to ours are asynchronous as covered in Section 5.2.1, so we are optimistic that an asynchronous variant of our method can be developed and analyzed.

Increasing λ. Theorem 5.1 shows that, to a point, convergence is faster when λ is smaller; however, Theorem 5.2 shows that the relaxation is closer to the constrained solution when λ is larger. This motivates a strategy for increasing λ during optimization to achieve fast convergence to H^*. Such a strategy may also mitigate the burden of choosing λ. An ideal method would update λ frequently, even every iteration; to avoid the overhead of coordinating synchronous updates to λ across the network, one might, for example, allow each client to have its own local λ that is updated and exchanged with neighbors each iteration. An ideal method should also be sensitive to changes such as the introduction of new clients or data. While these adaptations would be practically ideal, convergence analysis may be more challenging.

Broader support for specific models. This work covers efficient deterministic learning with function space regularization for a couple specific model classes, but such methods surely exist for others, and further study into such methods would improve the efficiency and performance of those models when used in this framework compared to black-box approximations. Promising places to start would be tree ensembles, which can extend the analysis of this work, and unsupervised methods such as clustering and mixture modeling, which open the door to a new application of the algorithm and are promising candidates for a precise and efficient function space inner product.

Improved model-agnostic methods. The model-agnostic Monte Carlo method proposed in Section 5.2.4 for computing function space inner products and norms is simple to implement, but suffers from poor sample efficiency as the dimensionality of the domain grows large. Methods for reducing the number of samples required to adequately approximate the inner products and norms would reduce the computational burden to solve (5.7) and, ultimately, make better models achievable on higher-dimensional data in practice.

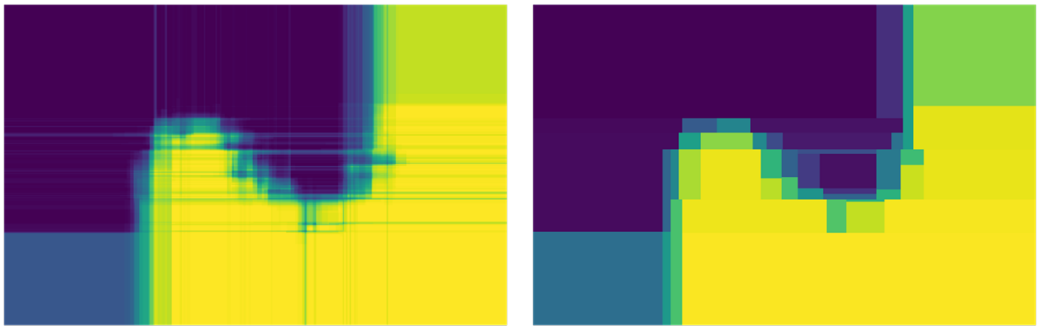
Improved inner products for high-dimensional input. This work uses a basic, common function inner product, but it is certainly not the only possible option.

Others may better align with the risk for different problems. Most critically, high-dimensional data is usually assumed to exist on some lower-dimensional manifold; thus an inner product that implicitly assumes an approximately uniform distribution is naive. A concrete example is images, where the sampling-based approximation of the basic inner product is ultimately enforcing agreement on random noise images, which are not representative of real images. One possibility is to use inner products weighted by density: given some probability density estimate $p : \mathcal{X} \to \mathbb{R}_{\geq 0}$, $\langle f, g \rangle = \int_{\mathcal{X}} p(x) \langle f(x), g(x) \rangle \, dx$ is a valid inner product that is better aligned with the global risk. For some pairs of density model and prediction model, such as diagonal-covariance Gaussian mixtures and trees, exact computation of the inner product is possible; otherwise, for a model-agnostic Monte Carlo approximation, it reduces to simply sampling from p instead of $\mathcal{U}_{\mathcal{X}}$, permitting, for instance, the use of a generative model for the regularization of image, audio, or text models. Another similar approach would be to use some shared dimension reduction or low-dimensional latent-space embedding to process the data, perhaps from a foundation model, then use function space regularization to learn the mapping from the embedding to the prediction. This has the potential to improve the performance and sample efficiency of function space algorithms on high-dimensional data of various modalities.

Robustness on dynamic and unstable networks. There are various ways that a network may be unstable and evolve over time, ranging from the introduction of new clients and data to shifting network connections to the varying availability of clients to participate to the permanent loss of some clients. For example, [14] highlight military and vehicular applications as a domain where these challenges are especially prevalent. Our method's inherent fast convergence and low communication cost are a good start, but it would be even stronger if combined with methods to improve robustness to specific challenges such as these. Many existing works address these challenges for other distributed optimization algorithms, and it is likely that many can be adapted for use with ours.

Networks with heterogeneous models. An interesting feature of the function space interpretation is that two models need not have the same architecture, or even belong to the same model class, to compute the inner product and learn with function space regularization. This unlocks the possibility of using different kinds of models at different clients, for example, to accommodate different computational resources. This would be most applicable for personalized models, as strictly enforcing consensus might limit all models to whatever is expressible by all of them, including the least powerful among them. An exploration of the use cases and practicality of this concept would be well-motivated.

Privacy. Privacy, that is, the protection of client data from leakage, is a primary motivation for using FL in many applications. Privacy depends partially on the model, partially on the distributed learning algorithm, and partially on the method of communication. It is left to future investigation to study the privacy implications of the proposed algorithm, as well as the models it introduces as candidates for federated



(a) Random forest (7870 nodes). (b) Merged tree (89 nodes).

Figure 5.8: A random forest trained on the two moons data set and a merged tree.

learning, such as decision trees.

Other applications of function space regularization. Outside decentralized federated learning, there are other areas of machine learning research where function space regularization of disagreement between models may be used in place of parameter regularization to achieve convexity, compatibility with non-parametric models, or other benefits. Examples include model distillation, ensemble merging, and continual learning.

5.3 Ensemble Merging

The hardware limitations, energy costs, and transparency needs of various applications motivate the topic of model compression [27], that is, the reduction of the size or complexity of a trained model with minimal reduction in performance. A common paradigm for model compression is knowledge distillation, where a small student model learns from a large teacher model, or ensemble of teacher models. See [37, 45] for surveys on model compression and [79] for knowledge distillation.

The vast majority of recent work focuses on the compression of large neural networks to reduce hardware and energy demands. However, the compression of tree ensembles by merging into a single tree, despite less recent focus, is particularly well-motivated due to the widespread application of tree ensembles. A good merging algorithm may achieve the best of both, that is, the reliable good performance of tree ensembles and the compactness, efficient inference, and interpretability of single-tree models.

Algorithms for solving local learning problems with function space regularization as discussed in Section 5.2.4 can also be applied directly as knowledge distillation algorithms by simply fitting with regularization to teacher model(s) without any local data. Since tree ensemble merging is especially well-motivated, and since the KDDT fitting algorithm easily generalizes to an efficient, deterministic algorithm for learning

trees with function space regularization to other trees, we study in this section the application of this algorithm for merging tree ensembles. A visual example is shown in Figure 5.8.

A number of prior works focus on the merging of tree ensembles. [107] recently surveys tree aggregation, including both the selection of a representative tree from a set and the merging of trees, noting the relevance of these topics to federated learning. They further subdivide methods of merging decision trees into those that 1. aggregate decision regions, 2. aggregate hierarchical structures, and 3. aggregate logical rules. Our approach is best aligned with category 1., as we essentially interpret trees as a collection of hyperrectangular regions and use the KDDT fitting algorithm, a generalization of CART, to fit to them. Recent work in this category includes the following. [56] interpret a tree ensemble as the set of hyperrectangles formed by the overlap of a leaf from each tree. They propose a procedure to construct a tree to partition these regions, thereby exactly representing the ensemble, then prune it to reduce size without affecting classification accuracy. [8] take a similar approach, but use a set of heuristic criteria to merge the regions to combat the exponential explosion of their quantity. [162] explicitly computes all regions, weights them importance based on one of several aggregations of the amount of data in the leaves of which the region is the intersection, merges contiguous same-class regions, and constructs a tree to represent the final list of regions.

Compared these methods, our approach is simple and easy to understand and implement as a natural extension of universally understood tree fitting methods. It is fully deterministic, requires no held out data, and enables intuitive control of the tradeoff of tree size vs. fidelity through typical tree growth stopping conditions. It never explicitly represents overlapping leaf regions like prior work, and since it stops tree construction early rather than post-pruning to the desired size, growing small trees is very efficient. On the opposite end of the spectrum, if allowed to fully grow, the merged tree exactly matches the ensemble. Like all region-based methods, however, it suffers from a curse of dimensionality where the number of overlapping regions grows at worst exponentially with the dimension of the input. While our method avoids the associated exponential increase in cost for a given tree size, it cannot avoid the increase in the size of tree required to approximate an ensemble with a useful level of fidelity.

5.3.1 Experiments

We demonstrate the application of this approach to ensemble merging on random forests trained on the 12 data sets described in Table 2.1. We start by training DTs and RFs of 100 trees, both with tree size selected by 5-fold cross-validation, then merge trees with size up to four times the size of the DT or the average size of trees in the RF, whichever is larger. Additional details in Appendix C.9.

The results are shown in Figure 5.9, which shows test accuracy vs. tree size aver-

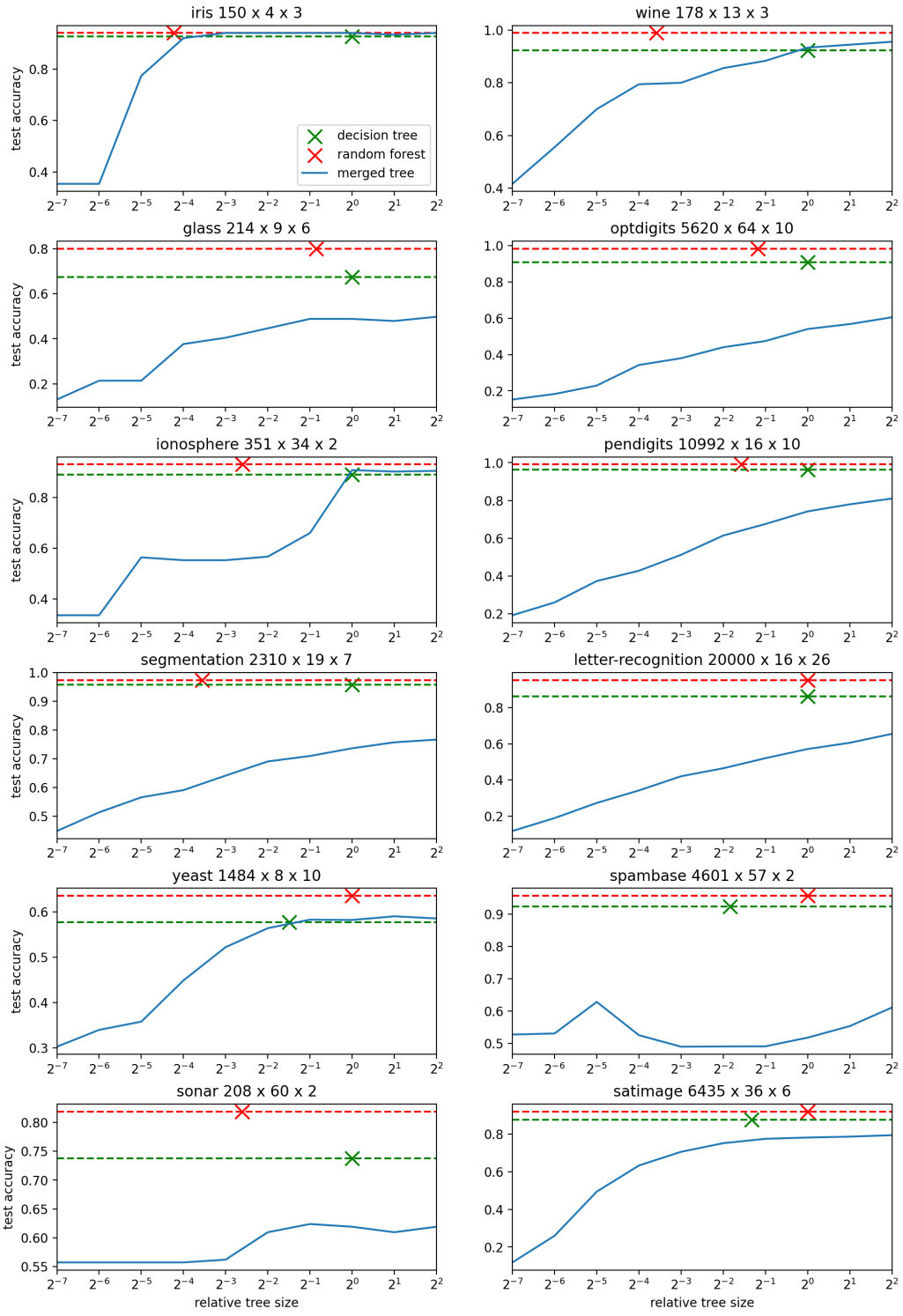


Figure 5.9: Results of ensemble merging experiments. For RFs, the average tree size is reported; the total model size is much larger.

aged over 5 trials. For random forests, we show the average tree size. As expected, for low-dimensional data, we are able to achieve a merged tree of reasonable size that outperforms a tree trained directly on the data. However, for even moderate-dimensional data, it is clear that a very large merged tree is often needed to approximate the ensemble well. This may still be useful if, for example, fast single-thread inference is needed, which is faster with a large tree than a similarly sized or even somewhat smaller ensemble.

Chapter 6

Conclusions

6.1 Contributions

In this thesis, towards the development of trustworthy methods in machine learning, we adopt of a functional view of models and introduce a formalism for uncertain interpretation of input that can be viewed as a kind of functional regularization. We develop methods for its efficient application to decision trees, a staple model class with inherent trust benefits that has fallen behind in the modern ML landscape. The resulting model class, a novel FDT called KDDTs, offer a variety of general and trust-related enhancements over conventional trees.

We test the KDDTs, as a form of regularized decision tree, against various notions of robustness. Compared to conventional decision trees, they have reduced sensitivity to small changes in data, better robustness to noisy features, better robustness to noisy labels, and an efficient mechanism for achieving and verifying adversarial robustness through the theory of randomized smoothing. We also propose the first verification algorithm for FDTs along with theoretical results and benchmarks showing its practicality and superiority to existing approaches.

We leverage the efficient fitting and differentiability of KDDTs to propose the first algorithm for gradient-based learning of feature transformations for trees that continually refits the tree throughout the process, retaining the adaptability, size minization, and information hierarchy of the classic CART algorithm while greatly reducing the size and increasing performance of single-tree models. When combined with domain-appropriate feature class selection and regularization, the results achieve an unprecedented balance of performance and interpretability. It also opens the door to more practical application of tree-based methods to modalities such as time series and images.

We enhance the utility of trees, which have fallen behind due to the focus of modern research on parametric models. We propose a method for Semi-supervised Learning (SSL), where there are few existing methods for decision trees, that leverages the unique capabilities of KDDTs. We introduce a new method of Federated Learning

(FL) in function space that is model-agnostic, theoretically and empirically fast to converge, and resilient to client heterogeneity. We furthermore introduce an ensemble merging method based on the FL algorithm that is efficient, simple, and effective for sufficiently low-dimensional data.

Tree-based models are everywhere, from retail to finance to sciences to healthcare. Anywhere they are applied, KDDTs can be swapped in with little effort, enhancing performance, utility, and trustworthy qualities. Our implementation of KDDTs and related methods covered in this thesis, including demonstration of its application, is publicly available at https://github.com/autonlab/kddt.

In particular, there are certain conditions where KDDTs shine most. Our benchmark results in Section 2.3.6, as well as the study of robustness in Chapter 3, suggest that KDDTs outperform alternatives the most when a single tree is preferable to an ensemble, the training set size is small, the features or labels are noisy, or there is possibility of adversarial perturbation. In particular, the results in Figures 3.4 and 3.5 suggest that, if the noise in the data is understood, then that knowledge may be leveraged toward kernel design to achieve better performance than by automatic kernel design. KDDTs are also uniquely useful when interpretability is a priority, both by improving single-tree performance and enabling ensemble merging, and also by making it possible to learn transformations of features that result in small, performant trees, as shown in Chapter 4. These also reduce featurization effort that would otherwise be required to effectively apply trees to time series and simple images, and future work may extend it further. Finally, the unique SSL and federated learning capabilities enabled by KDDTs are useful in applications with limited labeled data and restrictions on data sharing, respectively. In particular, the federated learning capability outperforms alternatives when the clients are highly heterogeneous, when fast convergence with low communication cost is a priority, or when tree-based models are preferred for their interpretability and utility.

6.2 Key Takeaways

We restate some of the themes and lessons learned from this work in hopes that they may inspire future endeavors towards the development of trustworthy AI.

A model that is simple and embraces uncertainty enables trust. We spoke at length in the introduction of this thesis about the connection between machine learning and science and how a good model, like a good scientific theory, is simple. In this work, we have demonstrated that, by combining simple models with an interpretation of data as uncertain, we can make strides towards trustworthy AI. These models are robust. They are transparent: their merits and flaws are evident, accelerating iterative improvement and calibrating trust; they are directly interpretable without the need for post-hoc approximate explanation; and they may even lead to insights in their domains of application. They are practical and high in utility. Moreover, models with these attributes are better inclined to tackle other complex and

computationally difficult problems in Trustworthy AI.

Both classical and modern perspectives are important. This work takes inspiration from modern perspectives and recent developments in areas such as neural network verification, randomized smoothing, nonconvex gradient-based optimization, and decentralized federated learning and applies them to decision trees, a flagship of classical machine learning. This blend of perspectives inspires methods that are more efficient, performant, or trustworthy than achievable by either perspective individually. We believe that opportunities like these are abundant, and the work is important; despite perceptions arising from the limitations of classical methods, they and the problems they are well-suited to address are abundant, and these problems are by no means completely solved. We consistently see ML practitioners and a subcommunity of researchers that value this work. However, for this kind of work to truly flourish, mainstream values in ML research must expand to embrace both sides of the field. With enough time and effort, we may see the gap close and achieve the best of both.

Models are not just parameters. To speak on one particular difference in perspective, much of the research in ML, especially recently, focuses exclusively on parametric models. In our opinion, this tips too far in the direction of exploitation of successful ideas at the cost of exploration of new ones. In this work, we have shown that core concepts like regularization, differentiation, and model similarity that are normally associated with model parameters need not be. In this case, we achieve these by adopting a functional perspective of models, introducing different benefits and drawbacks; for instance, the functional perspective on federated learning can benefit neural network learning, even though they are parametric. It of course also enables compatibility with non-parametric models like trees.

6.3 Limitations and Future Directions

Much of this work and its current limitations motivate future study. We highlight a number of promising topics, ordered according to the relevant content in this thesis.

Kernel Selection. As described in Section 2.5, the choice of kernel shape and size is crucial and nontrivial. Since its selection cannot be easily separated from the model, the best approach we have so far is to choose a simple, general kernel shape such as a box (uniform on a hypercube) or Gaussian kernel and choose a single bandwidth for all features by cross-validation with the learned model. Selection methods that achieve lower cost without a loss in performance would make KDDTs and other models with the the uncertain interpretation more practical to learn in resource-constrained settings. Moreover, cross-validation with the learning algorithm is generally not practical for selecting from a larger space of kernels, such as kernels with different bandwidth for each feature, or adaptive kernels with different size and/or shape at different inputs. If such kernel selection methods can be developed, it has the potential to reduce the cost of training and further improve the performance

of the associated models.

KDDTs as Interpretable Intermediate Layers. In Chapter 4, we leverage the differentiability of KDDTs so that they can function as the final layer of a multi-layer gradient-based learning pipeline. However, it is not currently possible to use them as an *intermediate* layer since they require labeled data for tree fitting; we can not update or fit them using loss gradients alone.

If this capability can be developed, it would open up a world of possibilities for the application of KDDTs as efficient, sparsely activated, interpretable components at various points in a larger pipeline. As a few speculative examples, KDDTs could be used as components of a neural architecture, as a fully tree-based layered architecture, or as interpretable routers for mixture-of-experts.

We note that many works already train trees in a fully gradient-based fashion, ranging from simple FDTs as in [163] to complex tree-shaped networks composed of transformers, routers, and solvers, each potentially a deep architecture, as in [168]. The key ingredients that distinguish our method are the formalism for defining fuzzy splitting and its efficient integration with the CART algorithm so that trees may be grown in a differentiable setting, allowing adaptive tree structure as learning proceeds. A good adaptation of our method should maintain these unique advantages.

More Expressive and Specialized Feature Transforms. In Section 4.2.3, we propose a number of feature transformation primitives for use with our tree feature learning pipeline studied in Chapter 4. We selected these for their simplicity, transparency, and general applicability. However, on complex and/or multimodal data, more expressive and specialized features may improve performance at the cost of some amount of transparency. Examples include the following.

- Template matching features with shift, scale, and/or rotation invariance.
- Image prototypes with a convolutional network for matching as in [34].
- Frequency-based features for time series.
- Featurization or other latent representation of text.

Though the interpretability of these varies, their use inside a tree-based partitioning model may still be more interpretable than their use with, say, a neural network. See the motivating discussion and Fashion-MNIST example in Section 4.3.4. Moreover, a great deal of research has gone into post-hoc explainability for various components, and the tree may serve to enhance the utility of such methods by breaking the explanation down into a series of simpler steps.

Decentralized Federated Learning. The DFL framework proposed in Section 5.2 is foundational and opens many avenues for future study. We discuss these at length in Section 5.2.6.

Appendices

Appendix A

Acronyms

ACR Average Certified Radius

ADMM Alternating Direction Method of Multipliers

AI Artificial Intelligence

CART Classification and Regression Trees

CCP Cost-Complexity Pruning

CDF Cumulative Distribution Function

DAG Directed Acyclic Graph

DFedAvgM Distributed Federated Averaging with Momentum

DFL Decentralized Federated Learning

DJAM Distributed Jacobi Asynchronous Method

DPGM Distributed Proximal Gradient Method

DRS (De-)Randomized Smoothing

DSE Decision Stump Ensemble

DT Decision Tree

ERM Empirical Risk Minimization

ET ExtraTrees

FDT Fuzzy Decision Tree

FDD Fuzzy Decision DAG

FSR Function Space Regularization

FL Federated Learning

KDDT Kernel Density Decision Tree

KDE Kernel Density Estimation

LR Linear (or Logistic) Regression

MAE Mean Absolute Error

MAP Minimum Adversarial Perturbation

ML Machine Learning

MLP Multi-Layer Perceptron

MSE Mean Squared Error

ODT Oblique Decision Tree

RF Random Forest

SMT Satisfiability Modulo Theories

SSL Semi-supervised Learning

SSL-PCT Semi-supervised Learning Predictive Clustering Tree

SVM Support Vector Machine

TAI Trustworthy AI

TAO Tree Alternating Optimization

XGB Extreme Gradient Boosting

Appendix B

Proofs

This section contains proofs omitted from the main text.

Theorem 2.2. *For KDDTs, Gini impurity is equivalent to Mean Squared Error (MSE) risk on the smoothed empirical distribution $\hat{p}_k$.*

Proof. Begin by writing the empirical MSE for a KDDT classifier with underlying tree h.

$$\begin{aligned} R[h] &= \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_k}[\|\mathbf{y} - h(\mathbf{x})\|_2^2] \\ &= \frac{1}{n}\sum_{i\in[n]}\int_{\mathbb{R}^p} k(\boldsymbol{z},\boldsymbol{x}_i)\|\boldsymbol{y}_i - h(\boldsymbol{z}_i)\|_2^2 \, d\boldsymbol{z} \\ &= \frac{1}{n}\sum_{i\in[n]}\int_{\mathbb{R}^p} k(\boldsymbol{z},\boldsymbol{x}_i)(\boldsymbol{y}_i^\top\boldsymbol{y}_i + h(\boldsymbol{z}_i)^\top h(\boldsymbol{z}_i) - 2\boldsymbol{y}_i^\top h(\boldsymbol{z}_i)) \, d\boldsymbol{z} \end{aligned}$$

Substitute the crisp tree prediction $h(\boldsymbol{x}) = \sum_{j\in\text{leaves}} \mathbf{1}\{\boldsymbol{x}\in R_j\}\boldsymbol{v}_j$ with R_j as in 2.7.

$$\begin{aligned} &= \frac{1}{n}\sum_{i\in[n]}\int_{\mathbb{R}^p} k(\boldsymbol{z},\boldsymbol{x}_i)\Bigg(\boldsymbol{y}_i^\top\boldsymbol{y}_i + \sum_{j,j'\in\text{leaves}} \mathbf{1}\{\boldsymbol{z}\in R_j\}\mathbf{1}\{\boldsymbol{z}\in R_{j'}\}\boldsymbol{v}_j^\top\boldsymbol{v}_{j'} \\ &\quad - 2\boldsymbol{y}_i^\top \sum_{j\in\text{leaves}} \mathbf{1}\{\boldsymbol{z}\in R_j\}\boldsymbol{v}_j\Bigg) \, d\boldsymbol{z} \end{aligned}$$

Leaves are non-overlapping, so $\mathbf{1}\{\boldsymbol{z}\in R_j\}\mathbf{1}\{\boldsymbol{z}\in R_{j'}\} = 0$ when $j \neq j'$.

$$\begin{aligned} &= \frac{1}{n}\sum_{i\in[n]}\int_{\mathbb{R}^p} k(\boldsymbol{z},\boldsymbol{x}_i)\Bigg(\boldsymbol{y}_i^\top\boldsymbol{y}_i + \sum_{j\in\text{leaves}} \mathbf{1}\{\boldsymbol{z}\in R_j\}\boldsymbol{v}_j^\top\boldsymbol{v}_j \\ &\quad - 2\boldsymbol{y}_i^\top \sum_{j\in\text{leaves}} \mathbf{1}\{\boldsymbol{z}\in R_j\}\boldsymbol{v}_j\Bigg) \, d\boldsymbol{z} \end{aligned}$$

$$= \frac{1}{n} \sum_{i \in [n]} \left(\boldsymbol{y}_i^\top \boldsymbol{y}_i + \sum_{j \in \text{leaves}} \boldsymbol{v}_j^\top \boldsymbol{v}_j \int_{R_j} k(\boldsymbol{z}, \boldsymbol{x}_i)\ d\boldsymbol{z} - 2 \sum_{j \in \text{leaves}} \boldsymbol{y}_i^\top \boldsymbol{v}_j \int_{R_j} k(\boldsymbol{z}, \boldsymbol{x}_i)\ d\boldsymbol{z} \right)$$

Substitute Equation 2.6.

$$= \frac{1}{n} \sum_{i \in [n]} \left(\boldsymbol{y}_i^\top \boldsymbol{y}_i + \sum_{j \in \text{leaves}} \mu_j(\boldsymbol{x}_i) \boldsymbol{v}_j^\top \boldsymbol{v}_j - 2 \sum_{j \in \text{leaves}} \mu_j(\boldsymbol{x}_i) \boldsymbol{y}_i^\top \boldsymbol{v}_j \right)$$

Since $\sum_{i \in [n]} \boldsymbol{y}_i^\top \boldsymbol{y}_i$ is constant, for the purpose of optimization, we can substitute it for another constant n (when $\boldsymbol{y}_i$ are indicators, they are actually equal).

$$= \frac{1}{n} \left(n + \sum_{i \in [n]} \sum_{j \in \text{leaves}} \mu_j(\boldsymbol{x}_i) \boldsymbol{v}_j^\top \boldsymbol{v}_j - 2 \sum_{i \in [n]} \sum_{j \in \text{leaves}} \mu_j(\boldsymbol{x}_i) \boldsymbol{y}_i^\top \boldsymbol{v}_j \right)$$
$$= \frac{1}{n} \left(n + \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \boldsymbol{v}_j^\top \boldsymbol{v}_j - 2 \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \boldsymbol{y}_i^\top \boldsymbol{v}_j \right)$$

Substitute using Equations 2.10 and 2.11.

$$= \frac{1}{n} \left(n + \sum_{j \in \text{leaves}} w_j \boldsymbol{v}_j^\top \boldsymbol{v}_j - 2 \sum_{j \in \text{leaves}} w_j \boldsymbol{v}_j^\top \boldsymbol{v}_j \right)$$
$$= \frac{1}{n} \left(n - \sum_{j \in \text{leaves}} w_j \boldsymbol{v}_j^\top \boldsymbol{v}_j \right)$$

We have $\sum_{j \in \text{leaves}} w_j = \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) = \sum_{i \in [n]} 1 = n$.

$$= \frac{1}{n} \sum_{j \in \text{leaves}} w_j (1 - \boldsymbol{v}_j^\top \boldsymbol{v}_j)$$

This is precisely the total Gini impurity as defined in Equations 2.9 and 2.12. □

Theorem 2.3. *For KDDTs, Entropy impurity is equivalent to cross-entropy risk on the smoothed empirical distribution $\hat{p}_k$.*

Proof. Begin by writing the empirical cross-entropy loss for a KDDT classifier with underlying tree h.

$$\begin{aligned} R[h] &= \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_k} [-\mathbf{y}^\top \log h(\mathbf{x})] \\ &= -\frac{1}{n} \sum_{i \in [n]} \int_{\mathbb{R}^p} k(\boldsymbol{z}, \boldsymbol{x}_i) \boldsymbol{y}_i^\top \log h(\boldsymbol{z}_i)\ d\boldsymbol{z} \end{aligned}$$

Substitute the crisp tree prediction $h(\boldsymbol{x}) = \sum_{j \in \text{leaves}} \mathbf{1}\{\boldsymbol{x} \in R_j\}\boldsymbol{v}_j$ with R_j as in 2.7.

$$= -\frac{1}{n} \sum_{i \in [n]} \int_{\mathbb{R}^p} k(\boldsymbol{z}, \boldsymbol{x}_i) \boldsymbol{y}_i^\top \log \sum_{j \in \text{leaves}} \mathbf{1}\{\boldsymbol{z} \in R_j\}\boldsymbol{v}_j \; d\boldsymbol{z}$$

Since leaves are a partition of $\mathbb{R}^p$, the sum over j has exactly one nonzero term, so we can take it out of the log.

$$= -\frac{1}{n} \sum_{i \in [n]} \int_{\mathbb{R}^p} \sum_{j \in \text{leaves}} \mathbf{1}\{\boldsymbol{z} \in R_j\} k(\boldsymbol{z}, \boldsymbol{x}_i) \boldsymbol{y}_i^\top \log \boldsymbol{v}_j \; d\boldsymbol{z}$$
$$= -\frac{1}{n} \sum_{j \in \text{leaves}} \sum_{i \in [n]} \boldsymbol{y}_i^\top \log \boldsymbol{v}_j \int_{R_j} k(\boldsymbol{z}, \boldsymbol{x}_i) \; d\boldsymbol{z}$$

Substitute Equation 2.6.

$$= -\frac{1}{n} \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \boldsymbol{y}_i^\top \log \boldsymbol{v}_j$$

Substitute using Equations 2.10 and 2.11.

$$= -\frac{1}{n} \sum_{j \in \text{leaves}} w_j v_j^\top \log \boldsymbol{v}_j$$

This is precisely the total Entropy impurity as defined in Equations 2.9 and 2.13. □

Theorem 3.1. *For any decision tree and symmetric, non-negative loss function, there exist loss-minimizing leaf values that are plurality indicators.*

Proof. The loss at a single leaf is as follows.

$$\sum_{i \in [n]} \mu(\boldsymbol{x}_i) \ell(\boldsymbol{v}, \boldsymbol{y}_i)$$

Let $w_j = \sum_{i \in [n]} \mu(\boldsymbol{x}_i) \mathbf{1}\{\boldsymbol{y}_i = \boldsymbol{e}_k\}$ denote the weight of class $j \in [q]$ at the leaf. Without loss of generality, assume $w_1 \geq w_j$ for $j > 1$, that is, label 1 is a plurality label.

$$= \sum_{j=1}^{q} w_j \ell(\boldsymbol{v}, \boldsymbol{e}_j)$$
$$= w_1 \ell(\boldsymbol{v}, \boldsymbol{e}_1) + \sum_{j=2}^{q} w_j \ell(\boldsymbol{v}, \boldsymbol{e}_j)$$

$$
\begin{aligned}
&= w_1 \ell(\boldsymbol{v}, \boldsymbol{e}_1) + \sum_{j=2}^{q} (w_j + (w_1 - w_1)) \ell(\boldsymbol{v}, \boldsymbol{e}_j) \\
&= w_1 \sum_{j=1}^{q} \ell(\boldsymbol{v}, \boldsymbol{e}_j) + \sum_{j=2}^{q} (w_j - w_1) \ell(\boldsymbol{v}, \boldsymbol{e}_j) \\
&= w_1 c + \sum_{j=2}^{q} (w_j - w_1) \ell(\boldsymbol{v}, \boldsymbol{e}_j) && (\ell \text{ is symmetric}) \\
&\geq w_1 c && (\ell \text{ is non-negative and } w_1 \geq w_j) \\
&= w_1 \sum_{j=1}^{q} \ell(\boldsymbol{e}_1, \boldsymbol{e}_j) && (\ell \text{ is symmetric}) \\
&\geq \sum_{j=1}^{q} w_j \ell(\boldsymbol{e}_1, \boldsymbol{e}_j) && (w_1 \geq w_j) \\
&= \sum_{i \in [n]} \mu(\boldsymbol{x}_i) \ell(\boldsymbol{e}_1, \boldsymbol{y}_i) && (\text{as above})
\end{aligned}
$$

This is the loss with $\boldsymbol{v} = \boldsymbol{e}_1$. Therefore this leaf value, the indicator of a plurality label, is a minimizer of the loss. □

Theorem 3.2. *The learned structure of a decision tree is invariant to forward loss correction.*

Proof. With forward correction loss $\ell_{\boldsymbol{T}}$, we have the following optimal leaf value $\boldsymbol{v}_j^{(\boldsymbol{T})}$ at leaf j.

$$
\begin{aligned}
\boldsymbol{v}_j^{(\boldsymbol{T})} &= \arg\min_{v} \sum_{i=1}^{n} \mu_j(\boldsymbol{x}_i) \ell_{\boldsymbol{T}}(\boldsymbol{v}, \boldsymbol{y}_i) \\
&= \arg\min_{v} \sum_{i=1}^{n} \mu_j(\boldsymbol{x}_i) \ell(\boldsymbol{T}\boldsymbol{v}, \boldsymbol{y}_i)
\end{aligned}
$$

A change of variables gives

$$
\begin{aligned}
\boldsymbol{T}\boldsymbol{v}_j^{(\boldsymbol{T})} &= \arg\min_{v} \sum_{i=1}^{n} \mu_j(\boldsymbol{x}_i) \ell_{\boldsymbol{T}}(\boldsymbol{v}, \boldsymbol{y}_i) \\
&= \boldsymbol{v}_j
\end{aligned}
$$

from which get corrected optimal leaf value $\boldsymbol{v}_j^{(\boldsymbol{T})} = \boldsymbol{T}^{-1}\boldsymbol{v}_j$. Plugging this in, the total corrected loss is

$$
\sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \ell_{\boldsymbol{T}}(\boldsymbol{v}_j^{(\boldsymbol{T})}, \boldsymbol{y}_i)
$$

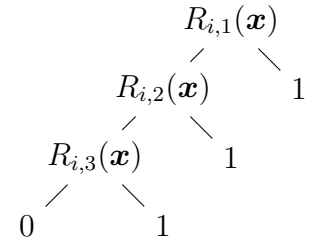
$$= \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \ell(\boldsymbol{T}\boldsymbol{T}^{-1}\boldsymbol{v}_j, \boldsymbol{y}_i)$$
$$= \sum_{j \in \text{leaves}} \sum_{i \in [n]} \mu_j(\boldsymbol{x}_i) \ell(\boldsymbol{v}_j, \boldsymbol{y}_i)$$

which is the same as the loss without forward correction. Therefore, the same splits are chosen, and the same tree structure is learned. □

Theorem 3.6. *Computing the smoothed prediction of a tree ensemble is NP-Hard.*

Proof. We show a polynomial reduction from 3-SAT. A 3-SAT formula F with m variables and n clauses has the form $C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Each clause C_i for $i \in [n]$ has the form $L_{i,1} \vee L_{i,2} \vee L_{i,3}$. Each literal $L_{i,j}$ for $i \in [n], j \in [3]$ is either a variable $X_{k_{i,j}}$ or its negation $\neg X_{k_{i,j}}$ for some $k_{i,j} \in [m]$. We call these positive and negative literals, respectively. An assignment maps each variable to true or false; an assignment satisfies F if the mapping causes F to evaluate to true. The 3-SAT problem is to determine whether a satisfying assignment exists.

For each C_i, construct a tree T_i with input $\boldsymbol{x}$



where each $R_{i,j}$ is a decision rule

$$R_{i,j}(\boldsymbol{x}) = \begin{cases} x_{k_{i,j}} > 0, & L_{i,j} = X_{k_{i,j}} \\ x_{k_{i,j}} \leq 0, & L_{i,j} = \neg X_{k_{i,j}} \end{cases}$$

and proceed right if the rule evaluates to true, otherwise left. Then we see that, if we let $X_i \iff x_i > 0$, then $R_{i,j}(\boldsymbol{x}) \iff L_{i,j}$, so $C_i \iff T_i(\boldsymbol{x}) = 1$.

Next, construct an ensemble E of $4n - 1$ trees, of which $2n$ are two each of the trees constructed above and the remaining $2n - 1$ are single-node stumps that always predict 0. Such an ensemble predicts 1 if and only if all $2n$ of the non-stump trees predict 1; therefore $E(\boldsymbol{x}) = 1 \iff \forall i \in [n]\ T_i(\boldsymbol{x}) = 1 \iff \forall i \in [n]\ C_i \iff F$.

Suppose F is unsatisfiable. Then $E(\boldsymbol{x}) = 0$ for all $\boldsymbol{x}$, and any smoothing of the predictions of E yields 0.

Suppose F is satisfiable with an assignment X. Then any corresponding $\boldsymbol{x}$ with $X_i \iff x_i > 0$ has $E(x) = 1$. Moreover, if we consider the domain $D = [-1, 1]^m$, then the set of such $\boldsymbol{x}$ on D has measure 1; therefore, if we smooth the predictions of E on $\mathcal{U}_D$, it yields a value strictly greater than 0.

Thus F is satisfiable if and only if smoothing E on $\mathcal{U}_D$ yields a value greater than zero. From this we conclude that smoothing the predictions of ensembles is NP-Hard. □

Lemma 3.1. *For any node i, $e_i = \sum_{j \in T_i} v(i, j)$.*

Proof. By induction. The base case occurs when i is a leaf. Leaves have a constant value and no children, so $\sum_{j \in T_i} v(i, j) = v(i, i) = 0 = e_i$.

For the inductive case, in which i is an internal node, we assume that the claim holds for the descendants of i, that is,

$$e_{\ell_i} = \sum_{j \in T_{\ell_i}} v(\ell_i, j)$$
$$e_{r_i} = \sum_{m \in T_{r_i}} v(r_i, j).$$

The following equality is evident from the illustration in Figure 3.11; it can also be derived from Equation 3.11.

$$\begin{aligned} 2e_i = &\left|l_i^{\min} - l_i^{\max}\right| + \left|u_i^{\min} - u_i^{\max}\right| \\ &+ (1 - \sigma_i^{\min}) e_{\ell_i} + \sigma_i^{\min} e_{r_i} \\ &+ (1 - \sigma_i^{\max}) e_{\ell_i} + \sigma_i^{\max} e_{r_i} \end{aligned}$$

Divide by two and substitute Equations 3.12 and 3.13.

$$e_i = v(i, i) + (1 - \bar{\sigma}_i) e_{\ell_i} + \bar{\sigma}_i e_{r_i}$$

Apply the inductive hypothesis.

$$= v(i, i) + (1 - \bar{\sigma}_i) \sum_{j \in T_{\ell_i}} v(\ell_i, j) + \bar{\sigma}_i \sum_{j \in T_{r_i}} v(r_i, j)$$

Substitute Equation 3.13.

$$\begin{aligned} &= v(i, i) + \sum_{j \in T_{\ell_i} \cup T_{r_i}} v(i, j) \\ &= \sum_{j \in T_i} v(i, j) \end{aligned}$$

The claim follows by induction. □

Theorem 3.8. *If splitting the domain affects only the chosen node, then choosing to split at the node that maximizes $v(i, \cdot)$, where i is the root, minimizes error summed over the resulting subdomains.*

Proof. Suppose the split occurs at node k. The $\bar{\sigma}$ for the new subdomains are

$$\bar{\sigma}_k^- = \frac{1}{2}(\sigma_k^{\min} + \bar{\sigma}_k)$$
$$\bar{\sigma}_k^+ = \frac{1}{2}(\bar{\sigma}_k + \sigma_k^{\max})$$

which gives

$$\bar{\sigma}_k^- + \bar{\sigma}_k^+ = 2\bar{\sigma}_k. \tag{B.1}$$

By the main assumption of this theorem, the $\sigma^{\min}$, $\sigma^{\max}$, and $\bar{\sigma}$ values are not affected at any other node.

Let v^- and v^+ denote the new v for the new subdomains. From Equations 3.11, 3.13, and B.1, one can derive

$$v^-(k,k) = v^+(k,k) = \frac{1}{2}v(k,k) \tag{B.2}$$

which is also made clear by examining Figure 3.11.

By Lemma 3.1, we have the following summed root error $e_i^- + e_i^+$ for the two resulting domains.

$$e_i^- + e_i^+ = \sum_{j \in T_i} v^-(i,j) + \sum_{j \in T_i} v^+(i,j)$$

We have $v^-(i,j) = v^+(i,j) = v(i,j)$ for $j \notin T_k$ since these do not depend on values from k.

$$= 2 \sum_{j \in T_i \setminus T_k} v(i,j) + \sum_{j \in T_k} [v^-(i,j) + v^+(i,j)]$$

By Equation 3.13, we can see that, for any $j \in T_k$, $v(i,j) = v(i,k)v(k,j)/v(k,k)$.

$$= 2 \sum_{j \in T_i \setminus T_k} v(i,j) + \sum_{j \in T_k} [v^-(i,k)v^-(k,j)/v^-(k,k) + v^+(i,k)v^+(k,j)/v^+(k,k)]$$

We have $v^-(i,k)/v^-(k,k) = v^+(i,k)/v^+(k,k) = v(i,k)/v(k,k)$ since $v(i,k)/v(k,k)$ does not depend on k.

$$= 2 \sum_{j \in T_i \setminus T_k} v(i,j) + \sum_{j \in T_k} [v^-(k,j) + v^+(k,j)]v(i,k)/v(k,k)$$
$$= 2 \sum_{j \in T_i \setminus T_k} v(i,j)$$
$$+ [v^-(k,k) + v^+(k,k)]v(i,k)/v(k,k)$$

$$+ \sum_{j \in T_{\ell_k}} [v^-(k,j) + v^+(k,j)] v(i,k)/v(k,k)$$
$$+ \sum_{j \in T_{r_k}} [v^-(k,j) + v^+(k,j)] v(i,k)/v(k,k)$$

To the second line, apply Equation B.2. To v^- and v^+ in the last two lines, apply the recursive definition of v from Equation 3.13.

$$= v(i,k) + 2 \sum_{j \in T_i \setminus T_k} v(i,j)$$
$$+ \sum_{j \in T_{\ell_k}} [(1-\bar{\sigma}_k^-) v^-(\ell_k,j) + (1-\bar{\sigma}_k^+) v^+(\ell_k,j)] v(i,k)/v(k,k)$$
$$+ \sum_{j \in T_{r_k}} [\bar{\sigma}_k^- v^-(r_k,j) + \bar{\sigma}_k^+ v^+(r_k,j)] v(i,k)/v(k,k)$$

We have $v^-(\ell_k,j) = v^+(\ell_k,j) = v(\ell_k,j)$ and $v^-(r_k,j) = v^+(r_k,j) = v(r_k,j)$ since they do not depend on k. Then, apply Equation B.1.

$$= v(i,k) + 2 \sum_{j \in T_i \setminus T_k} v(i,j)$$
$$+ 2 \sum_{j \in T_{\ell_k}} (1-\bar{\sigma}_k) v(\ell_k,j) v(i,k)/v(k,k)$$
$$+ 2 \sum_{j \in T_{r_k}} \bar{\sigma}_k v(r_k,j) v(i,k)/v(k,k)$$

Next, apply some of the same identities again in reverse order.

$$= v(i,k) + 2 \sum_{j \in T_i \setminus T_k} v(i,j)$$
$$+ 2 \sum_{j \in T_{\ell_k}} v(i,k) v(k,j)/v(k,k)$$
$$+ 2 \sum_{j \in T_{r_k}} v(i,k) v(k,j)/v(k,k)$$
$$= v(i,k) + 2 \sum_{j \in T_i \setminus T_k} v(i,j) + 2 \sum_{j \in T_{\ell_k}} v(i,j) + 2 \sum_{j \in T_{r_k}} v(i,j)$$
$$= 2 \sum_{j \in T_i} v(i,j) - v(i,k)$$

Finally, again apply Lemma 3.1.

$$= 2e_i - v(i,k)$$

Therefore the summed error is minimized when k is chosen to maximize $v(i,k)$. □

Theorem 3.9. *For any $\epsilon > 0$, an approximation with error at most ϵ is produced in finite steps.*

Proof. Let $\text{depth}(T_i)$ be the depth of the tree rooted at i, with a single-node tree having depth 1, and $\text{dist}(i,j)$ the distance from the root i to j. Define

$$\phi = \sum_{j \in T_i} [\text{depth}(T_i) - \text{dist}(i,j)] v(i,j).$$

Intuitively, ϕ measures the approximation error weighted according to its depth in the tree, where deeper is lower weight. We show that, in the worst case, when a split occurs, a portion of that error is moved deeper into the tree or removed, thus decreasing ϕ.

Let v' denote the function v for a subdomain after the split, and similarly for ϕ. We make the following three remarks.

1. For any $j \notin T_k$, $v'(i,j) = v(i,j)$ since it does not depend on k.
2. $v'(i,k) = \frac{1}{2} v(i,k)$, as shown in the proof of Theorem 3.8.
3. $\sum_{j \in T_i} v'(i,j) \leq \sum_{j \in T_i} v(i,j)$. If this does not hold, based on statement 1 and on Lemma 3.1, the reduction of splitting value range would result in an increase in the total approximation error, which is not possible based on the calculation of the bounds in Equation 3.11.

These show that $v(i,k)$ is decreased by half and $v(i,j)$ for deeper nodes j are increased by no more than the same amount. Since the increased nodes have a greater depth than k, they have lower weight in ϕ; thus $\phi' \leq \phi - \frac{1}{2} v(i,k)$.

Moreover, because k was chosen to maximize $v(i,k)$, we have

$$\phi \leq |T_i| \text{depth}(T_i) v(i,k),$$

where $|T_i|$ is the number of nodes in T_i, and therefore

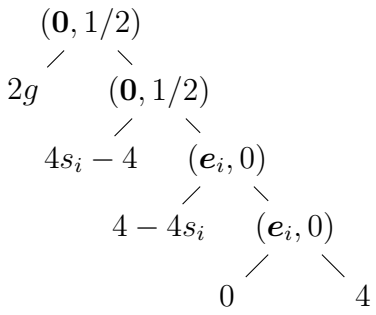
$$\phi' \leq \left(1 - \frac{1}{2|T_i|\text{depth}(T_i)}\right) \phi.$$

Thus we have convergence to $\phi < \epsilon$ in finite steps for any $\epsilon > 0$. Moreover, from the expression of e_i from Lemma 3.1, we can conclude $e_i \leq \phi$, so we also attain $e_i < \epsilon$ in finite steps. □

Lemma 3.2. *An FDT with $\sigma(z) = \max(0, \min(1, z))$ can be constructed to represent the polynomial $\sum_{i=1}^{k} x_i(x_i - 1) + \sum_{i=1}^{k} x_i s_i$ for $0 \leq x_i \leq 1$, $s_i \in \mathbb{R}$ in $O(k^2)$ time.*

Proof. By induction. The base case is $k = 0$, where the polynomial is equal to 0 and can be represented by a single leaf with value 0.

For the inductive case, we assume $k > 0$ and that we have an FDT $g(x) = \sum_{i=1}^{k-1} x_i(x_i - 1) + \sum_{i=1}^{k-1} x_i s_i$. Double the leaf values of g to get $2g$ and incorporate it into a new tree f with internal node parameters $(\boldsymbol{a}, b)$ and leaf values as shown below, where $\boldsymbol{e}_i$ is the vector with 1 at position i and 0 elsewhere.



With splitting function $\sigma(z) = \max(0, \min(1, z))$, This evaluates as follows.

$$\begin{aligned} f(\boldsymbol{x}) &= (1/2)(2g(\boldsymbol{x}) + (1/2)((4s_i - 4) \\ &\quad + (1 - x_i)(4 - 4s_i) + x_i(4x_i))) \\ &= g(\boldsymbol{x}) + x_i(1 - x_i) + x_i s_i \\ &= \sum_{i=1}^{k} x_i(x_i - 1) + \sum_{i=1}^{k} x_i s_i \end{aligned}$$

It follows by induction that an FDT can represent this polynomial for any k. Each step adds 8 nodes, so there are $O(k)$ nodes total. Each internal node requires $O(k)$ for the parameters, resulting in overall $O(k^2)$ for construction of the tree. □

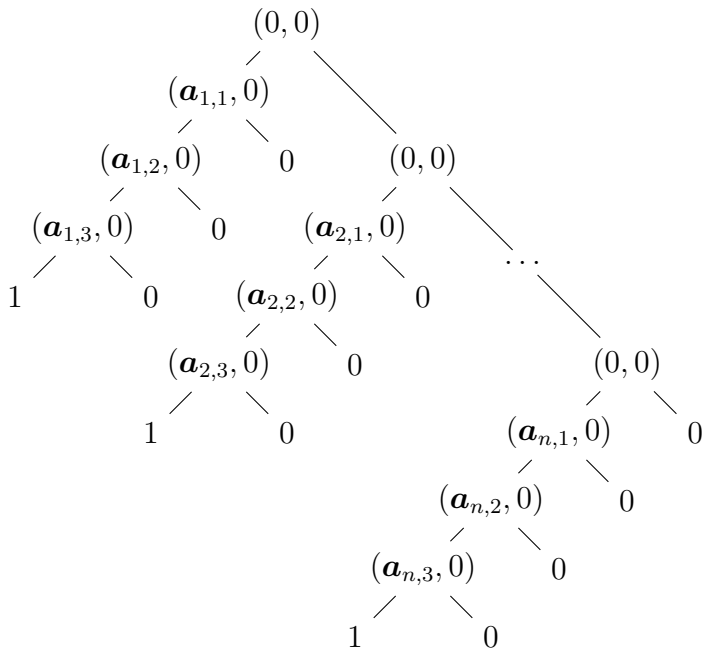
Lemma 3.3. *An FDT with $\sigma(z) = \frac{1}{2}(1 + sign(z))$ can be constructed to represent any 3-SAT formula with m variables and n clauses in $O(mn)$ time.*

Proof. A 3-SAT formula F with m variables and n clauses has the form $C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Each clause C_i for $i \in [n]$ has the form $L_{i,1} \vee L_{i,2} \vee L_{i,3}$. Each literal $L_{i,j}$ for $i \in [n], j \in [3]$ is either a variable $X_{k_{i,j}}$ or its negation $\neg X_{k_{i,j}}$ for some $k_{i,j} \in [m]$. We call these positive and negative literals, respectively. An assignment ϕ maps each variable to true or false; ϕ satisfies F if the mapping causes F to evaluate to true.

Let $\boldsymbol{e}_k$ be the indicator vector of length m with 1 at position $k \in [m]$ and 0 elsewhere. Define

$$\boldsymbol{a}_{i,j} = \begin{cases} \boldsymbol{e}_{k_{i,j}}, & L_{i,j} = X_{k_{i,j}} \\ -\boldsymbol{e}_{k_{i,j}}, & L_{i,j} = \neg X_{k_{i,j}} \end{cases}.$$

Construct an FDT $f : \mathbb{R}^m \to \mathbb{R}$ with splitting function $\sigma(z) = \frac{1}{2}(1 + \text{sign}(z))$ as follows, with parameters $(\boldsymbol{a}, b)$ shown for internal nodes and values for leaf nodes.



We now show that, for $\boldsymbol{x} \in \mathbb{R}^m$, $f(\boldsymbol{x}) = 0$ if and only if the assignments in the set $\Phi(\boldsymbol{x}) = \{\phi \mid \text{for all } k,\ x_k > 0 \implies \phi(X_k),\ x_k < 0 \implies \neg\phi(X_k)\}$ all satisfy F.

Each literal $L_{i,j}$ corresponds to the internal node with parameters $(\boldsymbol{a}_{i,j}, 0)$. If $L_{i,j}$ is positive, then

$$\sigma(\boldsymbol{a}_{i,j}{}^\top \boldsymbol{x}) = \begin{cases} 0, & x_{k_{i,j}} < 0 \\ \frac{1}{2}, & x_{k_{i,j}} = 0 \\ 1, & x_{k_{i,j}} > 0 \end{cases}.$$

If L_{ij} is negative, then

$$\sigma(\boldsymbol{a}_{i,j}{}^\top \boldsymbol{x}) = \begin{cases} 1, & x_{k_{i,j}} < 0 \\ \frac{1}{2}, & x_{k_{i,j}} = 0 \\ 0, & x_{k_{i,j}} > 0 \end{cases}.$$

Thus, each node has splitting value 1 if and only if the corresponding literal is true for all $\phi \in \Phi(\boldsymbol{x})$.

Suppose $f(\boldsymbol{x}) = 0$. Then every leaf with value 1 must have weight 0 in the computation of $f(\boldsymbol{x})$, that is, for every clause C_i, $(1/2)^i \prod_{j\in[3]}(1 - \sigma(\boldsymbol{a}_{i,j}^\top \boldsymbol{x})) = 0$. Thus, for each i, there is some j such that $\sigma(\boldsymbol{a}_{i,j}^\top \boldsymbol{x}) = 1$. It follows that, for each i, there is some j such that $L_{i,j}$ is true, so every $\phi \in \Phi(x)$ satisfies F.

Suppose instead $f(\boldsymbol{x}) \neq 0$. Then some leaf with value 1 must have positive weight in the computation of $f(\boldsymbol{x})$, that is, for some clause C_i, $(1/2)^i \prod_{j\in[3]}(1-\sigma(\boldsymbol{a}_{i,j}^\top \boldsymbol{x})) > 0$. Thus, for this i, $\sigma(\boldsymbol{a}_{i,j}^\top \boldsymbol{x}) < 1$ for all j. It follows that, for this i, there is some $\phi \in \Phi(\boldsymbol{x})$ such that $L_{i,j}$ is false for all j, so C_i is false and ϕ does not satisfy F.

Together these prove that f represents F as previously claimed. There are $8n+1$ nodes and each internal requires $O(m)$ parameters, so construction is $O(mn)$. □

Lemma 5.1. *Let φ be a quadratic function $\varphi(h) = \frac{1}{2}\langle h, Ah\rangle + \langle a, h\rangle + \alpha$ on a Hilbert space $\mathcal{H}$ with a minimum value φ^*. If A is positive with spectral gap $\sigma(A) \cap (0, c) = \emptyset$, then*

$$\|\nabla\varphi[h]\| \geq cd(h, \arg\min\varphi) \tag{5.8}$$

$$\varphi[h] \geq \varphi^* + \frac{1}{2}cd^2(h, \arg\min\varphi) \tag{5.9}$$

for any $h \in \mathcal{H}$.

Proof. Since a and α only shift φ, we may assume $a = 0$ and $\alpha = 0$ without loss of generality.

By the spectral theorem for bounded operators [143, Theorem VII.3], A is unitarily equivalent to a multiplication operator: there exists a finite measure space $(\mathcal{X}, \Sigma, \mu)$, a bounded measurable $f : \mathcal{X} \to \mathbb{R}$, and a unitary $U : \mathcal{H} \to L^2(\mathcal{X}, \mu)$ satisfying $U^{-1}T_fU = A$ where T_f is the multiplication operator $[T_fg](x) = f(x)g(x)$. The spectrum of T_f is the essential range of f.

We first show (5.8). For a given $h \in \mathcal{H}$, let $g = Uh$. Denote the projection of g onto the kernel of T_f by $g^* = P_{T_f}g$ with $h^* = U^{-1}g^*$. Denoting the zero set of f by O_f, and since U is unitary,

$$\begin{aligned}
\|\nabla\varphi[h]\|^2 &= \|Ah\|^2 \\
&= \|U^{-1}T_fUh\|^2 \\
&= \|T_fg\|^2 \\
&= \int_{\mathcal{X}} |f(x)g(x)|^2 d\mu(x) \\
&= \int_{\mathcal{X}\setminus O_f} |f(x)g(x)|^2 d\mu(x)
\end{aligned}$$

Since unitary equivalence implies equal spectrum, the essential range of f is nonnegative and takes no positive value less than c. Thus we have

$$\begin{aligned}
\|\nabla\varphi[h]\|^2 &\geq c^2 \int_{\mathcal{X}\setminus O_f} |g(x)|^2 \, d\mu(x) \\
&= c^2\|g - g^*\|^2 \\
&= c^2\|h - h^*\|^2.
\end{aligned}$$

Moreover, $\nabla\varphi[h^*] = Ah^* = U^{-1}T_fUU^{-1}g^* = U^{-1}T_fg^* = 0$, so $h^* \in \arg\min\varphi$ and (5.8) follows.

Next we show (5.9). Define g, g^*, and h^* as above. Again since U unitary,

$$\varphi[h] = \frac{1}{2}\langle h, Ah\rangle$$

$$
\begin{aligned}
&= \frac{1}{2}\langle h, U^{-1}T_f U h\rangle \\
&= \frac{1}{2}\langle Uh, T_f U h\rangle \\
&= \frac{1}{2}\langle g, T_f g\rangle \\
&= \frac{1}{2}\int_{\mathcal{X}} f(x)g(x)\overline{g(x)}\, d\mu(x) \\
&= \frac{1}{2}\int_{\mathcal{X}} f(x)|g(x)|^2 d\mu(x).
\end{aligned}
$$

Proceeding as in the proof of (5.8),

$$\varphi[h] \geq \frac{c}{2}\|h - h^*\|^2$$

and the claim follows since $\varphi^* = 0$ and $h^* \in \arg\min \varphi$. □

Lemma 5.2. *There exists some $c > 0$ such that $\sigma(\mathbf{A} + \lambda\mathbf{L}) \cap (0, c) = \emptyset$.*

Proof. Let $h \perp \ker(\mathbf{A} + \lambda\mathbf{L})$ with $||h|| = 1$. By Courant-Fisher [116, Theorem 12.1], it is enough to show that h satisfies $\langle(\mathbf{A}+\lambda\mathbf{L})\mathbf{h}, \mathbf{h}\rangle \geq c > 0$. Recall from assumptions 2 and 4 that both $\mathbf{A}$ and $\lambda\mathbf{L}$ have spectral gaps. Letting $P_A, P_L, P_A^\perp, P_L^\perp$ denote the projections onto $\ker \mathbf{A}$, $\ker \lambda\mathbf{L}$, and their orthogonal complements, self-adjointness and spectral gaps imply

$$\langle(\mathbf{A} + \lambda\mathbf{L})\mathbf{h}, \mathbf{h}\rangle = \tag{B.3}$$

$$\langle\mathbf{A}(P_A\mathbf{h} + P_A^\perp\mathbf{h}), (P_A\mathbf{h} + P_A^\perp\mathbf{h})\rangle + \lambda\langle\mathbf{L}(P_L\mathbf{h} + P_L^\perp\mathbf{h}), (P_L\mathbf{h} + P_L^\perp\mathbf{h})\rangle = \tag{B.4}$$

$$\langle\mathbf{A}P_A^\perp\mathbf{h}, P_A^\perp\mathbf{h}\rangle + \lambda\langle\mathbf{L}P_L^\perp\mathbf{h}, P_L^\perp\mathbf{h}\rangle \geq c_A||P_A^\perp\mathbf{h}||^2 + \lambda c_L||P_L^\perp\mathbf{h}||^2. \tag{B.5}$$

Consider $\overline{\mathcal{H}^n}$ the quotient of $\mathcal{H}^n$ by $\ker(\mathbf{A}+\lambda\mathbf{L})$, with norm $||\overline{\mathbf{h}}||_K = \inf_{\mathbf{g}\in\ker(\mathbf{A}+\lambda\mathbf{L})} ||\mathbf{h}-\mathbf{g}||$ for $\overline{\mathbf{h}} \in \overline{\mathcal{H}^n}$. Recalling that $\mathbf{A}, \lambda\mathbf{L}$ are positive, note that $\ker(\mathbf{A} + \lambda\mathbf{L}) = \ker(\mathbf{A}) \cap \ker(\mathbf{L})$. Define $\mu(h) = \left(\frac{1}{n}\sum_{j=1}^n h_j\right) \in \mathcal{H}$ and consider $\mathbf{k}$ in $\mathcal{H}^n$ with all elements equal to $P_0\mu(h)$ for P_0 the projection onto $\cap_{j=1}^n \ker(A_j)$. Clearly $\mathbf{k} \in \ker(\mathbf{A})\cap\ker(\mathbf{L})$ so that $||\overline{\mathbf{h}-\mathbf{k}}||_K = ||\overline{\mathbf{h}}-\overline{\mathbf{0}}||_K = 1$. Then by definition $||\mathbf{h}-\mathbf{k}||^2 = \sum_{j=1}^n ||h_j - P_0\mu(h)||^2 \geq 1$, implying $\sum_{j=1}^n ||h_j - P_0\mu(h)|| \geq 1$ by norm equivalence. Thus by the triangle inequality

$$\left(\sum_{j=1}^n ||h_j - \mu(h)|| + ||\mu(h) - P_0\mu(h)||\right) = \left(n||\mu(h) - P_0\mu(h)|| + \sum_{j=1}^n ||h_j - \mu(h)||\right) \geq 1.$$

By Lemma B.1, there must exist j such that $n^{3/2}||\mu(h) - P_{A_j}\mu(h)|| + \sum_j ||h_j - \mu(h)|| \geq 1$. Thus, again employing norm equivalence, we have that

$$n^{3/2}\sum_j ||\mu(h) - P_{A_j}\mu(h)|| + \sum_j ||h_j - \mu(h)|| \geq 1 \implies n^2||P_A^\perp P_L\mathbf{h}|| + \sqrt{n}||P_L^\perp\mathbf{h}|| \geq 1.$$

If $||P_L^\perp \mathbf{h}|| \geq \frac{1}{n}$ then we have that $c_A||P_A^\perp \mathbf{h}||^2 + \lambda c_L||P_L^\perp \mathbf{h}||^2 \geq \frac{\lambda c_L}{n^2}$. Otherwise we must have that $||P_A^\perp P_L \mathbf{h}|| \geq \frac{1-\sqrt{n}||P_L^\perp \mathbf{h}||}{n^2} > \frac{1-(1/\sqrt{n})}{n^2}$. Then it follows that

$$||P_A \mathbf{h} - P_L \mathbf{h}||^2 = \sum_{j=1}^{n} ||P_{A_j} h_j - \mu(h)||^2 = \sum_{j=1}^{n} ||P_{A_j}(h_j - \mu(h))||^2 + ||P_{A_j}^\perp \mu(h)||^2$$
$$= ||P_A P_L^\perp \mathbf{h}||^2 + ||P_A^\perp P_L \mathbf{h}||^2 \geq \left(\frac{1-(1/\sqrt{n})}{n^2}\right)^2.$$

Again by the triangle inequality

$$||P_A^\perp \mathbf{h}|| + ||P_L^\perp \mathbf{h}|| \geq ||P_A^\perp \mathbf{h} - P_L^\perp \mathbf{h}|| = ||(\mathbf{h} - P_A^\perp \mathbf{h}) - (\mathbf{h} - P_L^\perp \mathbf{h})|| = ||P_A \mathbf{h} - P_L \mathbf{h}||$$
$$\text{so that } ||P_A^\perp \mathbf{h}|| + ||P_L^\perp \mathbf{h}|| \geq \frac{1-(1/\sqrt{n})}{n^2}$$
$$\text{and } ||P_A^\perp \mathbf{h}||^2 + ||P_L^\perp \mathbf{h}||^2 \geq \frac{1}{2}\left(||P_A^\perp \mathbf{h}|| + ||P_L^\perp \mathbf{h}||\right)^2 \geq \frac{1}{2}\left(\frac{1-(1/\sqrt{n})}{n^2}\right)^2$$
$$\text{yielding } c_A||P_A^\perp \mathbf{h}||^2 + \lambda c_L||P_L^\perp \mathbf{h}||^2 \geq \frac{\min(c_A, \lambda c_L)}{2}\left(\frac{1-(1/\sqrt{n})}{n^2}\right)^2.$$

□

Lemma B.1. *Assume A_i, A_j commute for all i, j and let P_{A_j} and P_A denote the projection operators from $\mathcal{H}$ onto $\ker(A_j)$ and $\cap_{j=1}^n \ker(A_j)$ respectively. Then $||f - P_{A_j} f|| < \epsilon$ for all j implies $||f - P_A f|| < \epsilon\sqrt{n}$.*

Proof. Since A_i, A_j are self-adjoint and commute we must have that they are simultaneously diagonalizable [16, Theorem 6.5.1] (see also [58]): There exists a finite measure space $(\mathcal{X}, \Sigma, \mu)$, bounded measurable a_i, and unitary $U : \mathcal{H} \to L^2(\mathcal{X}, \mu)$ satisfying $U^{-1} T_{a_j} U = A_i$ for all j where T_{a_j} is the multiplication operator $[T_{a_j} g](x) = a_j(x) g(x)$. Letting $P_{T_{a_j}}$ and P_T denote the projection operators from $L^2(\mathcal{X}, \mu)$ to $\ker(T_{a_j})$ and $\cap_{j=1}^n \ker(T_{a_j})$, and using surjectivity of U,

$$\begin{aligned} ||f - P_{A_j} f|| &= \inf_{\{g \in \mathcal{H} \mid U^{-1} T_{a_j} U g = 0\}} ||f - g|| \\ &= \inf_{\{h \in L^2(\mathcal{X},\mu) \mid T_{a_j} h = 0\}} ||f - U^{-1} h|| \\ &= \inf_{\{h \in L^2(\mathcal{X},\mu) \mid T_{a_j} h = 0\}} ||Uf - h|| = ||Uf - P_{T_{a_j}} Uf||. \end{aligned}$$

Similarly, we have $||f - P_A f|| = ||Uf - P_T Uf||$. Thus we see that it is enough to show $h \in L^2(X, \mu)$ satisfy $||h - P_{T_{a_j}} h|| < \epsilon$ implies $||h - P_T h|| < \epsilon\sqrt{n}$.

Consider the zero sets $O_j = \{x \mid a_j(x) = 0\}$. Note that the projections have the effect of zeroing out h on these sets:

$$||h - P_{T_{a_j}} h||^2 = \inf_{\{g \in L^2(\mathcal{X},\mu) \mid g \cdot a_j = 0\}} ||h - g||^2$$

$$
\begin{aligned}
&= \inf_{\{g \in L^2(\mathcal{X},\mu) \mid g \cdot a_j = 0\}} \int_{\mathcal{X}} |h(x) - g(x)|^2 d\mu(x) \\
&= \inf_{\{g \in L^2(\mathcal{X},\mu) \mid g \cdot a_j = 0\}} \int_{O_j} |h(x) - g(x)|^2 d\mu(x) + \int_{O_j^c} |h(x) - g(x)|^2 d\mu(x) \\
&= \int_{O_j^c} |h(x)|^2 d\mu(x).
\end{aligned}
$$

Similarly, we can show $||h - P_T h||^2 = \int_{(\cap_j O_j)^c} |h(x)|^2 d\mu(x) = \int_{\cup_j O_j^c} |h(x)|^2 d\mu(x)$. Thus $||h - P_T h|| < \sqrt{n}||h - P_{T_{a_j}} h||$ follows by induction since

$$
\int_{O_i^c \cup O_j^c} |h(x)|^2 d\mu(x) \leq \int_{O_i^c} |h(x)|^2 d\mu(x) + \int_{O_j^c} |h(x)|^2 d\mu(x).
$$

□

Lemma B.2. *For a given λ, for any $\tilde{\mathbf{h}} \in \tilde{H}$,*

$$
\|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| \leq \frac{1}{\lambda} \frac{\|\mathbf{A}\|}{\nu} \sqrt{\frac{\|\mathbf{A}\|}{\mu}} d(H^*, \arg\min R) \in O(1/\lambda) \tag{B.6}
$$

where $\bar{\mathbf{h}} = \mathbf{E}\tilde{\mathbf{h}}$ is the projection of $\tilde{\mathbf{h}}$ into consensus.

Proof. Since $\bar{\mathbf{h}}$ is the projection of $\tilde{\mathbf{h}}$ onto the the minimizers of $\mathbf{h} \mapsto \langle \mathbf{h}, \mathbf{L}\mathbf{h} \rangle$, by Lemma 5.1, we have the following.

$$
\|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| \leq \frac{1}{\nu} \|\mathbf{L}\tilde{\mathbf{h}}\|
$$

By optimality of (5.6), $\nabla R[\tilde{\mathbf{h}}] + \lambda \mathbf{L}\tilde{\mathbf{h}} = 0$.

$$
= \frac{1}{\lambda\nu} \|\nabla R[\tilde{\mathbf{h}}]\|
$$

Recall that ∇R is $\|\mathbf{A}\|$-Lipschitz and $\nabla R[\mathbf{h}] = 0$ for $\mathbf{h} \in \arg\min R$.

$$
\begin{aligned}
&\leq \frac{\|\mathbf{A}\|}{\lambda\nu} d(\tilde{\mathbf{h}}, \arg\min R) \\
&= \frac{\|\mathbf{A}\|}{\lambda\nu} \sqrt{\frac{2}{\mu} \left(\frac{1}{2} \mu d^2(\tilde{\mathbf{h}}, \arg\min R) \right)}
\end{aligned}
$$

Let $R^* = \min R$ and apply Lemma 5.1.

$$
\leq \frac{\|\mathbf{A}\|}{\lambda\nu} \sqrt{\frac{2}{\mu} (R[\tilde{\mathbf{h}}] - R^*)}
$$

Let $\mathbf{h}^* \in H^*$. Since $\langle \mathbf{h}^*, \mathbf{L}\mathbf{h}^* \rangle = 0$ and $\tilde{\mathbf{h}}$ minimizes (5.6), we have $R[\tilde{\mathbf{h}}] \leq R[\mathbf{h}^*]$.

$$\leq \frac{\|\mathbf{A}\|}{\lambda\nu}\sqrt{\frac{2}{\mu}(R[\mathbf{h}^*] - R^*)}$$

Again apply the $\|\mathbf{A}\|$-Lipschitzness of ∇R.

$$\leq \frac{\|\mathbf{A}\|}{\lambda\nu}\sqrt{\frac{2}{\mu}\left(\frac{1}{2}\|\mathbf{A}\|d^2(\mathbf{h}^*, \arg\min R)\right)}$$

The claim follows by a simple manipulation. □

Theorem 5.2. *For a given λ, for any $\tilde{\mathbf{h}} \in \tilde{H}$,*

$$d(\tilde{\mathbf{h}}, H^*) \leq \frac{\|\mathbf{A}\|}{\lambda\nu}\sqrt{\frac{\|\mathbf{A}\|}{\mu}}\left(1 + \frac{n\|\mathbf{A}\|}{\mu}\right) d(H^*, \arg\min R) \in O(1/\lambda). \tag{5.11}$$

Proof. Let $\tilde{\mathbf{h}} \in \tilde{H}$, $\bar{\mathbf{h}}$ the projection of $\tilde{\mathbf{h}}$ into consensus, and $\mathbf{h}^*$ the projection of $\bar{\mathbf{h}}$ into H^*.

$$\|\tilde{\mathbf{h}} - \mathbf{h}^*\| \leq \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \|\bar{\mathbf{h}} - \mathbf{h}^*\|$$

Since both $\bar{\mathbf{h}}$ and $\mathbf{h}^*$ are in consensus, their elements are equal.

$$= \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \sqrt{n}\|\bar{h}_0 - h_0^*\|$$

By assumptions 5.1 and 5.4 with Lemma 5.1, since A_i commuting and having spectral gap μ implies $\sum_i A_i$ has spectral gap μ, the consensus risk functional $\bar{R}[h] = R[(h, \ldots, h)] = \sum_i R_i[h]$ is also convex quadratic with minimum growth rate μ away from its minimizers, of which h_0^* is one.

$$\leq \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \frac{\sqrt{n}}{\mu}\|\nabla\bar{R}[\bar{h}_0]\|$$
$$= \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \frac{n}{\mu}\|\mathbf{E}\nabla R[\bar{\mathbf{h}}]\|$$

By optimality of (5.6), $\nabla R[\tilde{\mathbf{h}}] + \lambda\mathbf{L}\tilde{\mathbf{h}} = \mathbf{0}$. Since L is a symmetric graph Laplacian, its rows and columns sum to zero, so $\mathbf{E}\mathbf{L} = \mathbf{L}\mathbf{E} = \mathbf{0}$. Then $\mathbf{E}(\nabla R[\tilde{\mathbf{h}}] + \lambda\mathbf{L}\tilde{\mathbf{h}}) = \mathbf{E}\nabla R[\tilde{\mathbf{h}}] = \mathbf{0}$.

$$= \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \frac{n}{\mu}\|\mathbf{E}\nabla R[\bar{\mathbf{h}}] - \mathbf{E}\nabla R[\tilde{\mathbf{h}}]\|$$
$$\leq \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \frac{n}{\mu}\|\nabla R[\bar{\mathbf{h}}] - \nabla R[\tilde{\mathbf{h}}]\|$$

$$\leq \|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\| + \frac{n\|\mathbf{A}\|}{\mu}\|\bar{\mathbf{h}} - \tilde{\mathbf{h}}\|$$
$$= \left(1 + \frac{n\|\mathbf{A}\|}{\mu}\right)\|\tilde{\mathbf{h}} - \bar{\mathbf{h}}\|$$

From here the result is proven by application of Theorem B.2. □

Appendix C

Experiment Details

Here we provide some experimental details omitted from the main text for the sake of brevity. Unless otherwise noted:

- Trees are fitted with Gini impurity.
- When fitting KDDTs with a Gaussian kernel, a histogram approximation with 11 bins from -3 to 3 standard deviations is used.
- When predicting with a Gaussian kernel, it is truncated at the 1st and 99th percentiles.

Details for specific experiments follow.

C.1 Data Normalization

For many experiments, we use the data described in Table 2.1. Unless otherwise noted, we normalize these data sets by scaling each feature to have mean 0 and standard deviation 1. For a few of them, we scale every feature by dividing by the largest standard deviation among features, so that after scaling, the largest standard deviation is 1 and the relative scaling is maintained. These include:

- Optical Recognition of Handwritten Digits
- Ionosphere
- Pen-Based Recognition of Handwritten Digits
- Yeast
- Connectionist Bench (Sonar, Mines vs. Rocks)
- Statlog (Landsat Satellite)

C.2 KDDT Benchmarks

These are additional details for the experiments in Section 2.3.6.

All data features are normalized to have mean 0 and standard deviation 1. The standard decision tree, random forest, and ExtraTrees implementations are from scikit-learn [138]. For the scikit-learn decision tree, we select a cost-complexity pruning α parameter from the log range of 10^-5 to 10^0. For KDDTs, we select kernel bandwidth from the log range of 10^{-2} to 10^0 by 10-fold cross-validation. This selects the same bandwidth for each feature, which is why we normalized the data. All ensembles, including KDDT ensembles, use the default settings of scikit-learn with 100 trees, subsampling the features to the square root of the number of candidates at each node, and bootstrapping data samples only for random forests.

C.3 Feature Noise

These are additional details for the feature noise experiments in Section 3.2.1.

The data are normalized as described in Appendix C.1. Uniform feature noise is drawn from $\mathcal{U}_[- \lambda, \lambda]$ for each $\lambda \in (0, 0.1, 0.2, \ldots, 1.0)$. Gaussian noise is drawn from $\mathcal{N}(0, \sigma)$ for each $\sigma \in (0, 0.1, 0.2, \ldots, 1.0)$. The kernels are box kernels with bandwidths from the same set of values. For each data set, noise type, noise rate, and kernel bandwidth, we run 5 trials with different seeds for data sets with at least 1000 samples, or 20 trials for data sets with less than 1000 samples. Each trial uses a 80% train, 20% test split. We also ran 1 trial, or 4 for data sets with less than 1000 samples, using Gaussian kernels with the same bandwidths and compared against the corresponding trials with box kernels. We found negligible difference in performance.

As a growth stopping condition for the trees, we select by 5-fold cross-validation a maximum number of leaves $\lfloor \varsigma \min(n, 1000) \rfloor$ for $\varsigma \in (2^0, 2^{-1}, \ldots, 2^{-5})$. For the main performance results, the bandwidth is selected from the candidate values by 5-fold cross-validation.

C.4 Label Noise

These are additional details for the label noise experiments in Section 3.3.4.

The data are normalized as described in Appendix C.1. Uniform label noise is applied for each error probability $\eta \in (0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3)$. For each data set and noise rate, we run 5 trials with different seeds for data sets with at least 1000 samples, or 20 trials for data sets with less than 1000 samples. Each trial uses a 80% train, 20% test split.

As a growth stopping condition for the trees, we select by 5-fold cross-validation a maximum number of leaves $\lfloor \varsigma \min(n, 1000) \rfloor$ for $\varsigma \in (2^0, 2^{-1}, \ldots, 2^{-5})$. The KDDTs

use a box kernel with bandwidth (radius) selected by 5-fold cross-validation from $(2^0, 2^{-1}, \ldots, 2^{-6})$. The impurity function is Gini impurity or credal Gini impurity.

C.5 Adversarial Perturbation

These are additional details for the adversarial perturbation experiments in Section 3.4.3.

For the Breast Cancer data set, samples with missing values are removed, as in [93]. The MNIST [46] and FMNIST [189] data sets are each a selection of two classes from the original 10-class data sets. For the MNIST and FMNIST data sets, we use the official training and test sets, and for the others, we use a 80% train, 20% test split without shuffling to get the same split as [93]. We train KDDT trees and forests with maximum number of leaves equal to the number of training data or 1000, whichever is smaller. The forests have 100 trees. Since the robustness is more sensitive to the fidelity of the smoothing, for the L2, models, we increase the number of pieces in the histogram approximate Gaussian kernel used for fitting from our usual 11 to 15, and the unapproximated Gaussian kernels are not truncated for prediction. We also include standard decision trees as a baseline with CCP-α selected using 5-fold cross-validation in a 7-value log range from 10^{-5} to 10^{-2}.

C.6 Feature Learning

These are additional details for the interpretability benchmarks and demonstrations in Section 4.3.

C.6.1 Benchmarks on Tabular Data

In Section 4.3.1, our experiments compare decision trees, random forests, ExtraTrees, and our proposed models with linear and distance-to-prototype features. All reported statistics are the average of 10-fold cross-validation, and in the additional results in Appendix D.2.1 we also report standard deviation.

All data sets were retrieved from the UCI Machine Learning Repository [49]. If there were separate training and test sets, they were combined before creating the random 10-fold split. Categorical attributes are one-hot encoded. All attributes are normalized to mean 0 and standard deviation 1. This makes feature learning more consistent, and it makes interpretation unitless.

For baseline models, we used implementations from scikit-learn [139], and our methods are implemented using Python along with PyTorch [136] for automatic differentiation. The splitting criterion for all tree-based models is Gini Index. Additional details for each model type follow.

- Decision tree. Using 10-fold cross validation (on the union of the 9 training folds for the experimental layer of cross-validation), we select the cost-complexity pruning α that results in the best accuracy. Our candidate α values include 0 and 15 evenly spaced values on the log scale from 10^{-8} to 10^{-1}, that is, $\{0, 1 \times 10^{-8}, 3.16 \times 10^{-8}, 1 \times 10^{-7}, 3.16 \times 10^{-7}, \ldots, 1 \times 10^{-1}\}$.

- Random forest. We use default settings from scikit-learn. The ensemble contains 100 trees, each trained on a bootstrap sample with the same size as the original data, and the features consider for each split are limited to a uniformly sampled subset with size equal to the square root of the total number of features. There is no pruning.

- ExtraTrees. We use default settings from scikit-learn. The ensemble contains 100 trees, the features consider for each split are limited to a uniformly sampled subset with size equal to the square root of the total number of features, and for each feature, one candidate threshold is sampled uniformly in the range of data belonging to the current subtree. There is no pruning.

- Ours. For the KDDT, we use a box kernel with radius 0.1, that is, $k(\mathbf{x}) \propto \prod_{i=1}^{p} \mathbf{1}\{|x_i| \leq 0.1\}$. Do not perform a split if it would result in a leaf with total sample weight less than 1. We select cost-complexity pruning α from $[.0001, .0003, .001, .003, .01, .03, .1]$ by 10-fold cross-validation of accuracy. We implement feature learning using PyTorch [136] for automatic differentiation and stochastic optimization. The optimizer is Adam [103] with learning rate 0.01, and the optimization procedure uses minibatch gradient descent with batch size 1024. Training runs for 10 epochs fitting the tree before each minibatch, then 1000 epochs fitting the tree once every 10 epochs. For the dry-bean and pendigits data sets, we instead train for 10 then 100 epochs because of their much larger size. This process allows the tree structure changes to be responsive early in training when the features are changing rapidly, then saves time by fitting less frequently as the features converge. We apply either L1 or L2 regularization to the feature parameters, each with coefficient 0.01. The details for each feature type follow.

  - Linear features. We use a linear transformation without bias, with the same number of outputs as inputs. We initialize either as identity or uniformly at random in the range $\pm\sqrt{6/p}$, where p is the number of inputs to the linear transformation. Results in the main paper use random initialization.
  - Distance-to-prototype features. We use a number of prototypes equal to the number of attributes in the data. Initialization is either random, with prototypes being samples from the unit Gaussian and the inverse covariance being identity, or by using the centers and inverse covariance matrices from a fitted Gaussian Mixture model from scikit-learn, with matching

constraints on the covariance. Results in the main paper use random initialization. We constrain that inverse covariance matrices be positive definite so that the features represent distance. For these experiments, we also constrain inverse covariance matrices to be diagonal for the sake of easier interpretation, that is, so that it can be interpreted as Euclidean distance with each input being scaled differently. Regularization is applied only to the covariance parameters, although L1 regularization could also be applied to the prototypes themselves to make them sparse in the sense that they only sparsely differ from the global average. This may be useful for data with many attributes, so that prototypes can be described by just a few features.

C.6.2 MNIST and Fashion-MNIST

The data is scaled into $[0, 1]$. The KDDTs also use a box kernel with radius 1. We used the Adam optimizer [103] with learning rate 0.001 and minibatch gradient descent with batch size 1024. For 10 epochs, the tree is fitted once per batch to the batch itself; the large batch size is chosen to ensure that the tree fitted to each batch is representative enough of the tree for the entire training data, while being faster than fitting to the entire MNIST training set at each batch. We then train for 100 epochs with the tree fitted to the entire training set once per epoch. The feature transformation is a linear mapping from $28^2 = 784$ to 784 features. For smaller models, a smaller number of features would be fine and would speed up training. We use L1 regularization with coefficient 0.3 and a weight image smoothing regularizer with coefficient 0.03 for MNIST, and respectively 3 and 0.01 for Fashion-MNIST. The smoothing penalty itself is calculated as the average squared difference of each pixel with its neighbors (not including diagonal).

C.6.3 Time Series Shapelets

For the time series demos, the KDDTs use a box kernel with radius 0.1. We use 50 shapelet features initialized by choosing random training data instances (or random subseries of instances in the case of sliding window shapelets) and setting the weights uniformly across each shapelet, scaled such that the standard deviation of the output on the training data is 1. We use a L1 regularization and smoothness regularization, each with coefficient 0.01. Since these are small training sets, we use a batch size equal to the size of the training set. We train for 1000 iterations, refitting the tree every 5th iteration. For ECG5000, we use a learning rate of 0.001 and CCP-α of 0.003, and for GunPoint, a learning rate of 0.01 and CCP-α of 0.01.

C.7 Semi-supervised Learning

These are additional detials for the semi-supervised learning experiments covered in Section 5.1.3.

The data is preprocessed by standardizing all features to mean 0 and standard deviation 1, with the exception of the Optical Recognition of Handwritten Digits data set, where features, representing pixel values, are scaled into $[0, 1]$ by dividing all features by the global maximum value of 16. When we uniformly sample data to be labeled in the training of models, we ensure that at least one instance of each class is represented, that is, we sample one instance uniformly at random from each class, then the rest uniformly at random (without replacement) from the entire remaining data set. For cluster sampling, there is no such restriction.

The supervised random forest and self-trained random forest models are implementations from the popular machine learning package scikit-learn [138]. We are not aware of any public implementation of SSL-PCTs, so we implement them as described in [109]. Hyperparameters are as follows.

- **Random forest**. We use `sklearn.tree.DecisionTreeClassifier` with default hyperparameters. Each has 100 fully grown trees, and features to consider for splitting are randomly subsampled to $\sqrt{p}$ at each split. If a suitable split is not found, the remaining features are searched.

- **Self-training Random Forest**. We use `sklearn.tree.DecisionTreeClassifier` as above along with `sklearn.semi_supervised.SelfTrainingClassifier` with default hyperparameters.

- **SSL-PCT random forest**. There is a hyperparameter w used to control the level of supervision. It is selected as described in [109] by 3-fold cross-validation on the labeled data. The random forest hyperparameters are as above. In order to have leaf values be defined, we set a minimum of 1 labeled sample per leaf.

- **Our methods**. Our KDDTs use a Gaussian kernel. For fitting, a piecewise-constant kernel is required, so we use a histogram approximation of Gaussian with 7 pieces truncated to 3 standard deviations. At leaf assignment and predictions, we use an unapproximated Gaussian kernel, also truncated to 3 standard deviations. We choose bandwidth (standard deviation of the Gaussian kernel) from $[0.01, 0.0215, .0464, 0.1]$. We choose number of leaves as a proportion of the number of samples in the training set, with candidate values being powers of 2 from 2^0 to 2^{-6}. During leaf assignment, we increase the weight of labeled data to $\max(1, |D_U|/|D_L|)$. This gives at least as much total weight to labeled data as unlabeled data to prevent collapse to the global majority if the data is not easily separable.

C.8 Federated Learning

These are additional details for the federated learning experiments covered in Section 5.2.5.

The MLP, which is used for both PSR and FSR experiments, consists of two hidden layers of size 50 with ReLU activations. We use batch size 200 and learning rate 0.001 with the Adam optimizer [103]. We train for 10000 iterations (batches) locally, then after each round of communication, train for another 1000 iterations with disagreement penalty. For PSR, we use cross-entropy loss and penalize disagreement as the sum of squared difference in parameters and use coefficient $\lambda = 0.1$, which we observe to work well across data sets. For FSR, we use mean squared error loss and penalize disagreement at 1000 inputs sampled uniformly at random from the domain at each batch. Error smoothing is accomplished by, at each batch, adding random noise sampled from the kernel to the training inputs.

The KDDT uses as growth stopping condition a maximum number of leaves equal to the number of training samples summed over clients or 1000, whichever is smaller.

For FSR methods, λ and the box kernel radius δ are chosen to maximize average global training accuracy. For MLPs, we select from $\lambda \in [10, 100, \ldots, 10^7]$ and $\delta \in [0.0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5]$. For KDDTs, we select from $\lambda \in [10, 100, \ldots, 10^5]$ and $\delta \in [0.05, 0.1, 0.2, 0.5]$. These values for λ may seem large, but the convergence theory suggests that sometimes they should actually be even higher. The best λ is often based more on the local learning algorithm than the convergence of the federated optimization.

In all FSR training, we scale the data such that its bounding box, including smoothing, is $[0, 1]^p$. Though we do this up-front for simplicity, it is also straightforward to accomplish this dynamically on a network by communicating data bounding boxes along with models. This scaling is not theoretically necessary, but it makes the choice of hyperparameters more consistent across data sets and prevents the measure of the domain, which scales the disagreement penalty, from taking on extreme values that may be computationally unfavorable.

C.9 Ensemble Merging

These are additional details for the ensemble merging experiments covered in Section 5.1.3.

The random forests have 100 trees. As a growth stopping condition for the trees, we select by 5-fold cross-validation a maximum number of leaves $\lfloor \varsigma \min(n, 1000) \rfloor$ for $\varsigma \in (2^0, 2^{-1}, \ldots, 2^{-5})$. The impurity function is Gini impurity or credal Gini impurity. Each experiment is repeated 5 times, each with an independently sampled 80% train, 20% test split.

Appendix D

Additional Experiment Results

Here we show some expanded experiment results that were too lengthy to include in the main text.

D.1 Feature Noise

Additional results for the feature noise experiments in Section 3.2.1 are shown in Figures D.1 through D.4.

D.2 Interpretability

The following are additional results and visualizations for benchmarks and demonstrations in Section 4.3.

D.2.1 Benchmarks on Tabular Data

These results supplement Section 4.3.1. In these comprehensive experiment results, we show the mean and standard deviation from 10-fold cross-validation with five evaluation metrics:

1. Validation accuracy.
2. Total number of nodes.
3. Average length of a validation decision path.
4. Average Gini impurity (a good measure of sparsity [94], lower is sparser) of the parameters for each feature, weighted by each feature's usage on the respective validation fold.
5. Total time for inference on the validation fold, in milliseconds.

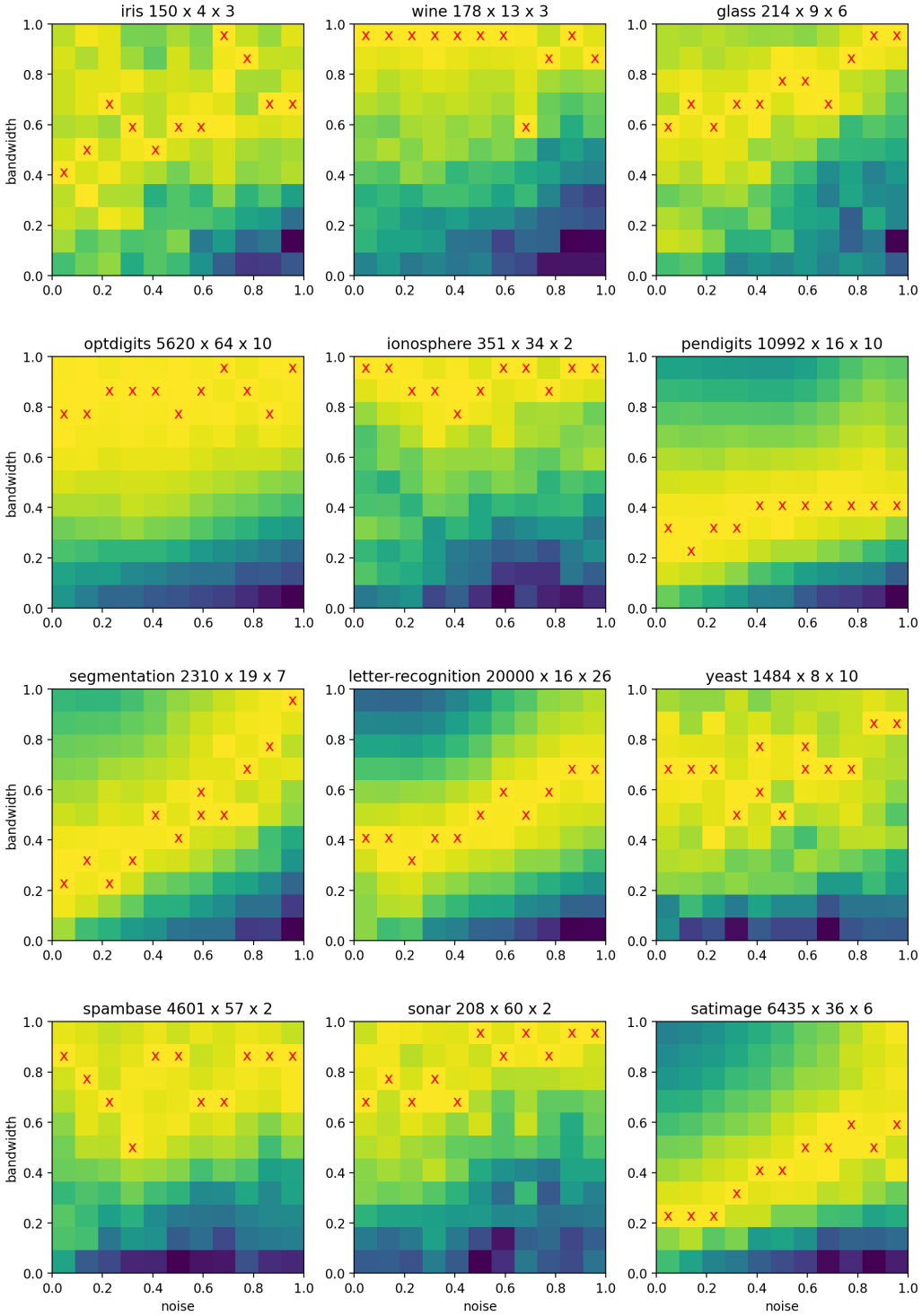


Figure D.1: Performance relative to best of noise level (marked with 'x') for smoothed KDDTs with box kernels trained on data with added uniform noise.

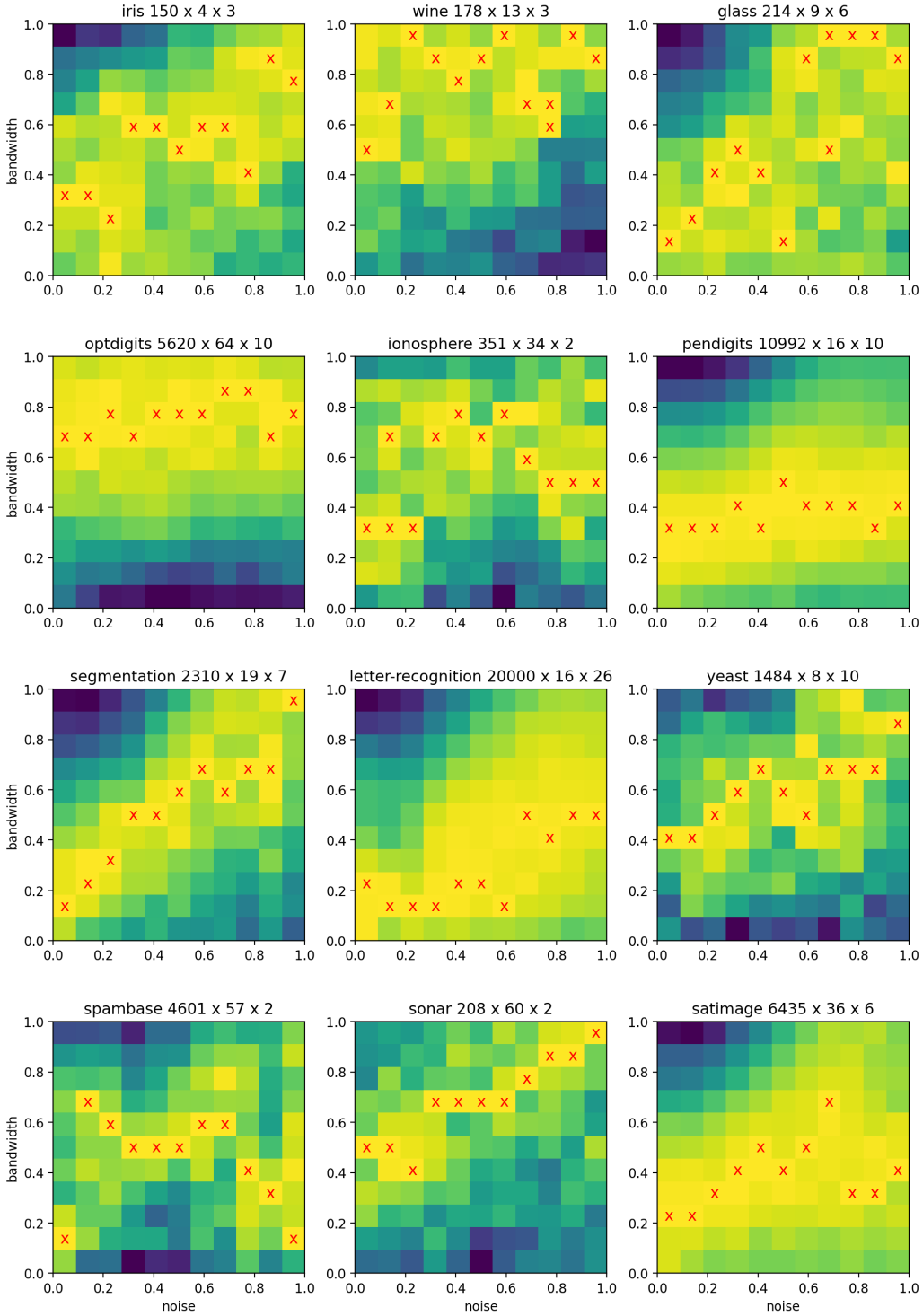


Figure D.2: Performance relative to best of noise level (marked with 'x') for unsmoothed KDDTs with box kernels trained on data with added uniform noise.

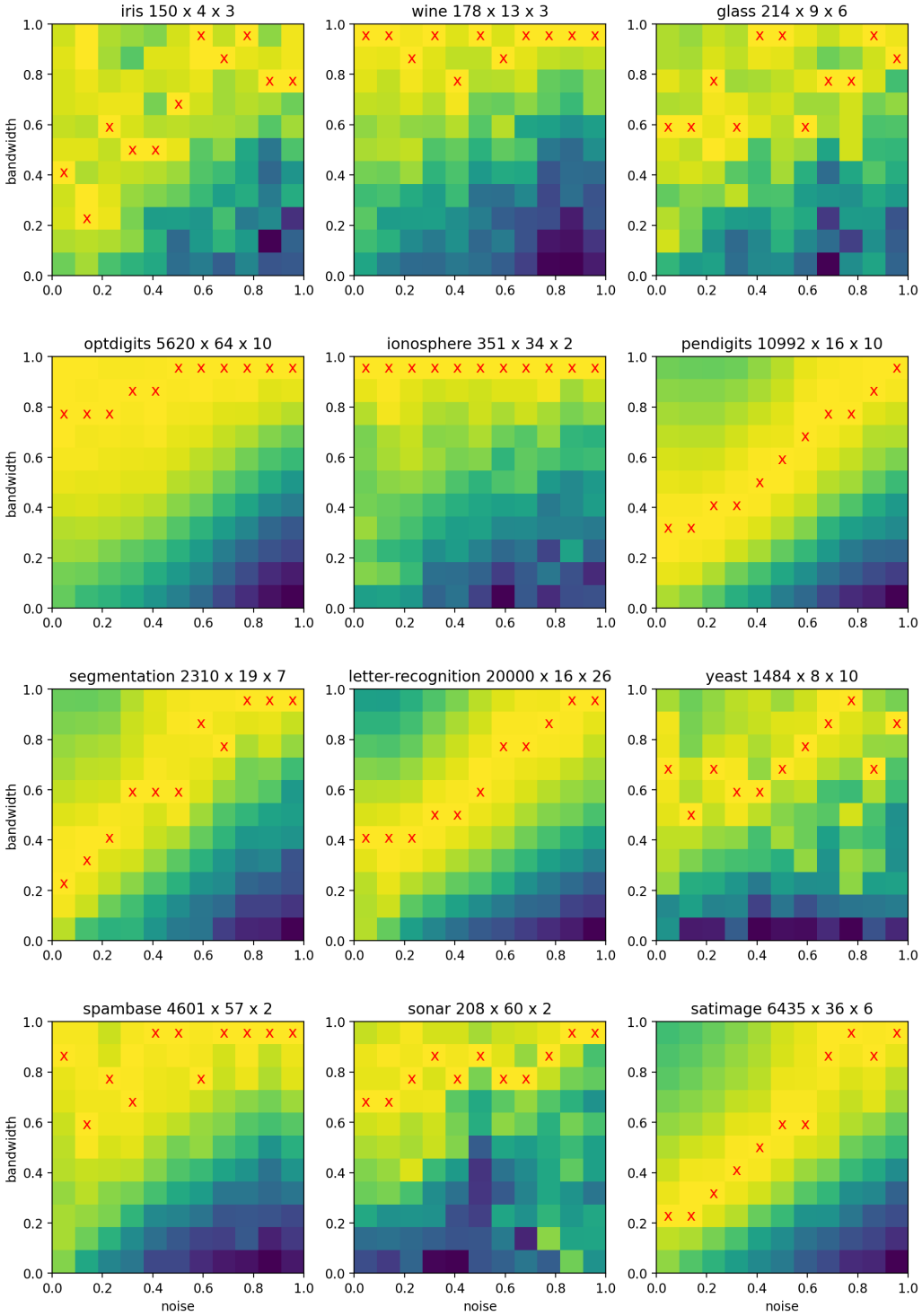


Figure D.3: Performance relative to best of noise level (marked with 'x') for smoothed KDDTs with box kernels trained on data with added Gaussian noise.

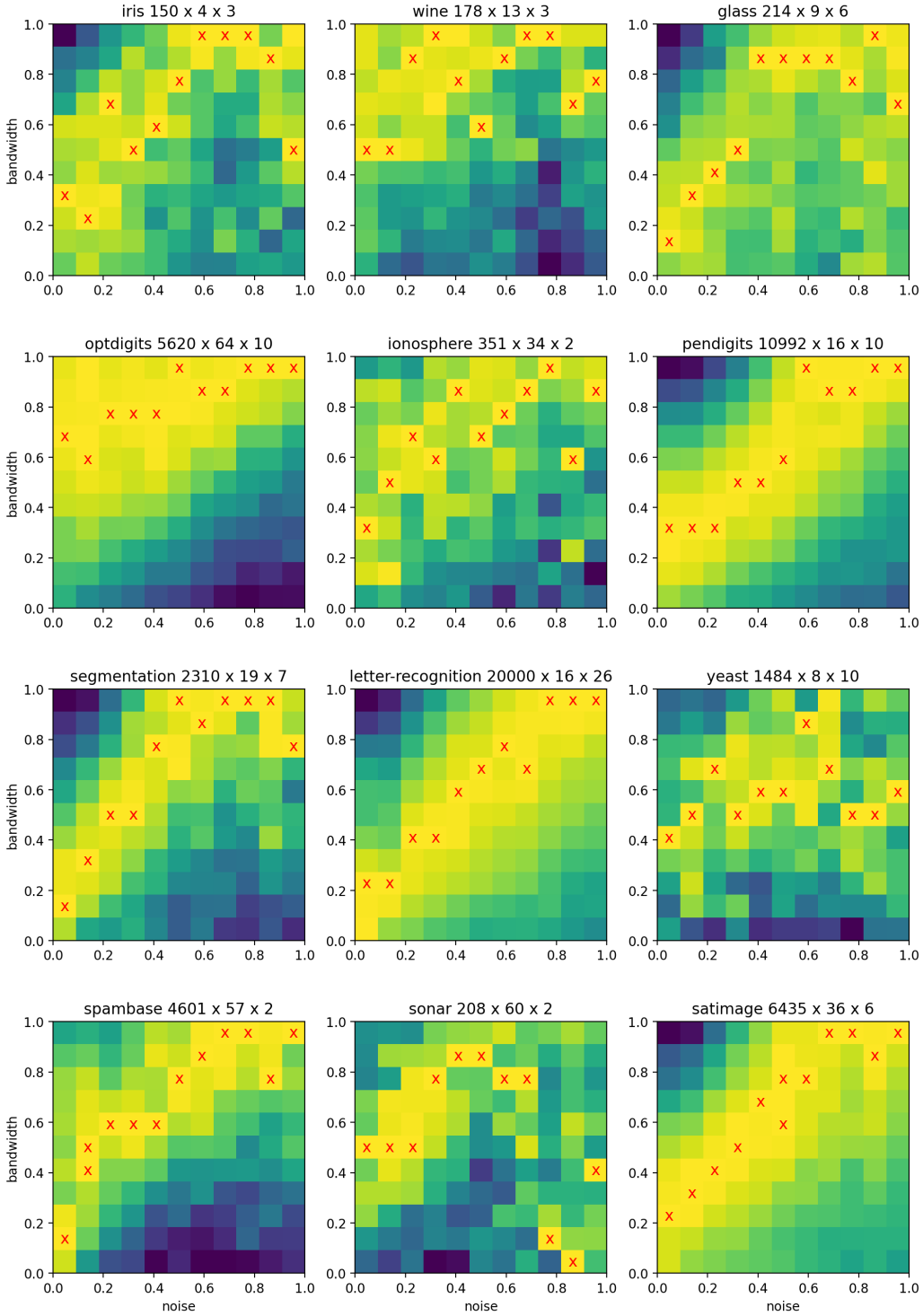


Figure D.4: Performance relative to best of noise level (marked with 'x') for unsmoothed KDDTs with box kernels trained on data with added Gaussian noise.

For the proposed method, we include linear and distance-to-prototype features, each with L1 and L2 regularization, random and non-random intialization, and crisp and fuzzy inference. Each table shows results for every CCP-α value, which controls tree size; smaller α yields larger trees. See Appendix C.6.1 for full details.

We make some observations based on the additional results:

- Random initialization more often results in better-performing models.

- Even when one of our models has many nodes, which may make global interpretation difficult, the average path length grows much less, so local interpretation is still simple.

- L1 regularization does result in sparser features compared to L2 regularization. With L2 regularization, features are usually very dense, with Gini index near 1.

- Our models have faster inference time than similarly performing ensembles.

data n, p, q	metric	LR	MLP	DT	RF	ET	XGB
iris	acc	.960 ± .044	.953 ± .052	.947 ± .058	.947 ± .065	.953 ± .052	.947 ± .058
150, 4 (4), 3	nodes	0.0 ± 0.0	0.0 ± 0.0	6.4 ± 2.2	720.4 ± 50.7	2057.3 ± 109.7	432.6 ± 45.0
	path len	0.00 ± 0.00	0.00 ± 0.00	2.48 ± 0.38	261.74 ± 17.31	449.09 ± 37.55	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.7 ± 0.2	9.7 ± 26.6	0.6 ± 0.0	128.4 ± 45.5	34.2 ± 18.8	78.7 ± 63.7
heart-disease	acc	.822 ± .021	.792 ± .067	.707 ± .060	.802 ± .065	.795 ± .052	.792 ± .038
303, 13 (20), 2	nodes	0.0 ± 0.0	0.0 ± 0.0	13.9 ± 16.1	4827.2 ± 108.1	10648.6 ± 206.0	788.6 ± 20.1
	path len	0.00 ± 0.00	0.00 ± 0.00	3.03 ± 1.71	584.43 ± 15.94	771.86 ± 30.70	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.4 ± 0.2	0.4 ± 0.0	0.3 ± 0.0	6.4 ± 0.1	6.6 ± 0.1	3.3 ± 0.7
dry-bean	acc	.925 ± .007	.934 ± .005	.912 ± .008	.923 ± .006	.921 ± .007	.928 ± .006
13611, 16 (16), 7	nodes	0.0 ± 0.0	0.0 ± 0.0	99.8 ± 3.8	66504.9 ± 530.6	197338.7 ± 1264.0	12907.8 ± 166.2
	path len	0.00 ± 0.00	0.00 ± 0.00	7.05 ± 0.14	1142.00 ± 7.96	1287.98 ± 12.75	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.9 ± 0.2	1.4 ± 0.0	0.7 ± 0.0	38.3 ± 0.3	49.2 ± 0.3	4.3 ± 0.2
wine	acc	.983 ± .025	.989 ± .022	.904 ± .077	.977 ± .028	.989 ± .022	.955 ± .043
178, 13 (13), 3	nodes	0.0 ± 0.0	0.0 ± 0.0	8.5 ± 2.1	936.4 ± 19.5	3315.1 ± 43.1	242.1 ± 15.0
	path len	0.00 ± 0.00	0.00 ± 0.00	3.30 ± 0.40	332.74 ± 6.59	579.64 ± 16.11	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.6 ± 0.2	0.4 ± 0.0	0.3 ± 0.0	6.2 ± 0.2	6.2 ± 0.0	1.7 ± 0.2
car	acc	.926 ± .021	.992 ± .007	.977 ± .012	.964 ± .013	.971 ± .011	.994 ± .006
1728, 6 (21), 4	nodes	0.0 ± 0.0	0.0 ± 0.0	95.3 ± 6.6	23031.0 ± 243.0	31240.4 ± 330.8	4478.2 ± 48.9
	path len	0.00 ± 0.00	0.00 ± 0.00	4.48 ± 0.27	610.58 ± 21.80	617.75 ± 23.81	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.8 ± 0.1	1.1 ± 0.0	0.7 ± 0.1	16.4 ± 0.3	17.0 ± 0.5	3.7 ± 0.3
wdbc	acc	.974 ± .021	.975 ± .024	.935 ± .032	.965 ± .019	.970 ± .028	.968 ± .021
569, 30 (30), 2	nodes	0.0 ± 0.0	0.0 ± 0.0	13.0 ± 6.5	1881.4 ± 59.9	6045.5 ± 193.9	274.4 ± 8.2
	path len	0.00 ± 0.00	0.00 ± 0.00	3.96 ± 1.25	462.02 ± 12.92	667.99 ± 21.11	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.8 ± 0.1	0.8 ± 0.0	0.6 ± 0.0	7.7 ± 2.1	6.6 ± 0.1	2.0 ± 0.4
sonar	acc	.755 ± .094	.879 ± .064	.735 ± .096	.826 ± .104	.880 ± .058	.855 ± .063
208, 60 (60), 2	nodes	0.0 ± 0.0	0.0 ± 0.0	14.1 ± 6.7	2022.6 ± 16.4	5586.1 ± 77.9	301.1 ± 6.0
	path len	0.00 ± 0.00	0.00 ± 0.00	3.86 ± 1.46	490.86 ± 10.09	708.90 ± 25.75	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.4 ± 0.2	0.4 ± 0.0	0.3 ± 0.0	6.2 ± 0.2	6.4 ± 0.1	1.8 ± 0.2
pendigits	acc	.952 ± .005	.994 ± .003	.964 ± .004	.993 ± .002	.994 ± .002	.991 ± .002
10992, 16 (16), 10	nodes	0.0 ± 0.0	0.0 ± 0.0	322.0 ± 13.4	38475.5 ± 232.0	98345.3 ± 655.6	8464.5 ± 60.7
	path len	0.00 ± 0.00	0.00 ± 0.00	10.13 ± 0.22	974.59 ± 5.43	1142.83 ± 7.26	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	29.8 ± 37.4	36.1 ± 63.6	9.3 ± 13.0	271.2 ± 80.6	43.2 ± 0.8	4.6 ± 0.7
ionosphere	acc	.875 ± .069	.917 ± .062	.892 ± .048	.934 ± .056	.943 ± .049	.943 ± .053
351, 34 (34), 2	nodes	0.0 ± 0.0	0.0 ± 0.0	15.5 ± 9.0	2205.7 ± 88.3	5919.3 ± 267.2	335.4 ± 20.4
	path len	0.00 ± 0.00	0.00 ± 0.00	5.17 ± 2.43	645.54 ± 39.76	889.93 ± 48.63	0.00 ± 0.00
	gini	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000	.000 ± .000
	time	0.4 ± 0.2	0.4 ± 0.0	0.3 ± 0.0	6.4 ± 0.3	6.5 ± 0.1	1.8 ± 0.2

Ours: linear features, L2 regularization, non-random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.960 ± .044	.960 ± .044	.953 ± .043	.953 ± .052	.967 ± .033	.960 ± .044	.967 ± .033
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.8 ± 0.7	5.0 ± 0.9	8.7 ± 1.7	12.6 ± 1.5	16.1 ± 1.6
	path len	1.67 ± 0.09	1.67 ± 0.09	2.28 ± 0.31	2.41 ± 0.22	2.82 ± 0.28	3.16 ± 0.40	3.66 ± 0.41
	gini	.618 ± .010	.618 ± .014	.458 ± .081	.500 ± .059	.548 ± .071	.565 ± .052	.561 ± .061
	time	1.2 ± 0.0	4.3 ± 9.5	3.7 ± 7.6	1.5 ± 0.2	2.4 ± 0.4	3.3 ± 0.3	4.1 ± 0.4
heart	acc	.799 ± .053	.795 ± .028	.776 ± .043	.766 ± .051	.782 ± .058	.743 ± .041	.762 ± .058
	nodes	1.0 ± 0.0	1.7 ± 0.5	5.7 ± 1.7	19.5 ± 3.3	21.7 ± 1.7	26.8 ± 2.6	27.1 ± 3.6
	path len	1.00 ± 0.00	1.40 ± 0.28	2.67 ± 0.62	4.15 ± 0.51	4.84 ± 0.58	5.45 ± 0.63	6.00 ± 0.98
	gini	.888 ± .010	.881 ± .025	.862 ± .014	.824 ± .045	.830 ± .034	.814 ± .041	.836 ± .032
	time	0.7 ± 0.0	0.9 ± 0.2	2.1 ± 0.5	3.9 ± 0.6	4.3 ± 0.3	5.4 ± 0.5	5.5 ± 0.7
dry-bean	acc	.792 ± .005	.913 ± .009	.919 ± .006	.918 ± .004	.920 ± .006	.925 ± .006	.925 ± .005
	nodes	4.0 ± 0.0	6.0 ± 0.0	8.0 ± 0.0	9.9 ± 0.7	18.2 ± 1.0	41.2 ± 3.2	94.3 ± 3.5
	path len	2.83 ± 0.03	3.22 ± 0.05	3.40 ± 0.08	3.61 ± 0.16	4.72 ± 0.13	6.13 ± 0.32	7.48 ± 0.33
	gini	.902 ± .002	.902 ± .003	.910 ± .002	.909 ± .003	.909 ± .002	.906 ± .003	.908 ± .002
	time	3.3 ± 0.1	4.2 ± 0.1	5.4 ± 0.1	6.2 ± 0.3	10.2 ± 0.4	15.4 ± 1.2	31.1 ± 0.9
wine	acc	.961 ± .026	.972 ± .028	.972 ± .028	.972 ± .028	.966 ± .027	.977 ± .028	.972 ± .028
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.8 ± 0.4	2.7 ± 0.5	2.9 ± 0.3	2.8 ± 0.4	2.7 ± 0.5
	path len	1.67 ± 0.09	1.67 ± 0.09	1.92 ± 0.15	1.90 ± 0.16	1.95 ± 0.15	1.96 ± 0.18	1.89 ± 0.18
	gini	.871 ± .011	.875 ± .009	.884 ± .011	.885 ± .007	.891 ± .005	.886 ± .009	.885 ± .009
	time	0.6 ± 0.0	0.6 ± 0.0	0.7 ± 0.1	0.7 ± 0.1	0.7 ± 0.0	0.7 ± 0.1	0.7 ± 0.1
car	acc	.700 ± .044	.914 ± .034	.965 ± .013	.986 ± .009	.992 ± .012	.992 ± .009	.992 ± .009
	nodes	0.0 ± 0.0	2.8 ± 0.4	5.0 ± 0.8	12.3 ± 1.6	18.8 ± 2.7	23.7 ± 2.1	24.5 ± 4.1
	path len	0.00 ± 0.00	1.60 ± 0.16	1.90 ± 0.18	2.86 ± 0.34	3.23 ± 0.52	3.61 ± 0.29	3.70 ± 0.63
	gini	.000 ± .000	.905 ± .010	.897 ± .015	.902 ± .015	.902 ± .014	.903 ± .007	.901 ± .010
	time	0.2 ± 0.0	1.2 ± 0.1	1.7 ± 0.2	3.6 ± 0.4	5.3 ± 0.7	6.6 ± 0.7	6.8 ± 1.2
wdbc	acc	.965 ± .025	.967 ± .023	.960 ± .027	.963 ± .021	.968 ± .025	.960 ± .024	.963 ± .024
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	2.1 ± 0.9	5.9 ± 2.1	13.9 ± 4.2	19.6 ± 3.6
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.59 ± 0.48	2.51 ± 0.31	3.76 ± 0.68	4.54 ± 0.39
	gini	.954 ± .003	.955 ± .003	.954 ± .003	.951 ± .003	.951 ± .003	.950 ± .003	.951 ± .004
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	0.9 ± 0.3	1.8 ± 0.5	3.8 ± 1.0	5.3 ± 1.0
sonar	acc	.765 ± .093	.802 ± .043	.798 ± .080	.817 ± .088	.779 ± .074	.856 ± .038	.807 ± .063
	nodes	1.0 ± 0.0	2.2 ± 0.6	4.5 ± 0.8	5.6 ± 0.9	6.8 ± 3.0	7.6 ± 2.1	8.0 ± 2.4
	path len	1.00 ± 0.00	1.60 ± 0.24	2.53 ± 0.31	2.84 ± 0.50	3.34 ± 0.93	3.53 ± 0.70	3.54 ± 0.77
	gini	.971 ± .002	.972 ± .001	.973 ± .001	.950 ± .073	.973 ± .002	.973 ± .001	.973 ± .002
	time	0.5 ± 0.0	0.9 ± 0.2	1.5 ± 0.2	1.7 ± 0.2	2.0 ± 0.7	2.2 ± 0.5	2.3 ± 0.6
pendigits	acc	.094 ± .003	.911 ± .022	.955 ± .011	.974 ± .006	.986 ± .003	.989 ± .003	.991 ± .002
	nodes	0.0 ± 0.0	9.3 ± 0.5	12.1 ± 1.0	17.9 ± 1.7	27.4 ± 3.0	57.5 ± 6.6	125.9 ± 9.5
	path len	0.00 ± 0.00	4.18 ± 0.32	4.46 ± 0.31	5.18 ± 0.23	5.64 ± 0.45	6.58 ± 0.42	8.19 ± 0.60
	gini	.000 ± .000	.891 ± .008	.892 ± .005	.891 ± .006	.896 ± .006	.899 ± .002	.897 ± .005
	time	0.3 ± 0.0	4.5 ± 0.1	5.5 ± 0.5	7.7 ± 0.5	10.7 ± 1.0	19.6 ± 1.9	37.9 ± 2.6
ionosphere	acc	.857 ± .057	.932 ± .060	.932 ± .048	.935 ± .034	.934 ± .050	.929 ± .048	.946 ± .038
	nodes	1.0 ± 0.0	2.1 ± 0.3	3.8 ± 0.4	4.8 ± 0.7	8.5 ± 1.9	9.5 ± 1.8	11.8 ± 2.0
	path len	1.00 ± 0.00	1.77 ± 0.13	2.91 ± 0.24	3.52 ± 0.57	4.49 ± 0.51	4.70 ± 0.85	5.39 ± 0.79
	gini	.948 ± .003	.929 ± .012	.925 ± .005	.926 ± .005	.939 ± .005	.934 ± .007	.931 ± .004
	time	0.7 ± 0.1	1.1 ± 0.1	1.7 ± 0.1	2.0 ± 0.3	3.4 ± 0.7	2.4 ± 0.4	3.0 ± 0.5

Ours: linear features, L2 regularization, non-random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.960 ± .044	.960 ± .044	.953 ± .043	.967 ± .033	.960 ± .044	.960 ± .053	.933 ± .067
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.8 ± 0.7	5.0 ± 0.9	8.7 ± 1.7	12.6 ± 1.5	16.1 ± 1.6
	path len	1.67 ± 0.09	1.67 ± 0.09	2.29 ± 0.33	2.38 ± 0.22	2.81 ± 0.25	3.15 ± 0.44	3.66 ± 0.42
	gini	.584 ± .103	.580 ± .111	.459 ± .078	.483 ± .082	.554 ± .039	.565 ± .054	.564 ± .047
	time	0.7 ± 0.0	0.7 ± 0.0	1.1 ± 0.2	1.4 ± 0.2	2.1 ± 0.3	2.9 ± 0.3	3.6 ± 0.3
heart	acc	.799 ± .055	.795 ± .028	.776 ± .043	.763 ± .052	.782 ± .066	.749 ± .046	.762 ± .058
	nodes	1.0 ± 0.0	1.7 ± 0.5	5.7 ± 1.7	19.5 ± 3.3	21.7 ± 1.7	26.8 ± 2.6	27.1 ± 3.6
	path len	1.00 ± 0.00	1.41 ± 0.29	2.66 ± 0.63	4.12 ± 0.49	4.82 ± 0.58	5.45 ± 0.62	6.01 ± 1.01
	gini	.888 ± .010	.881 ± .024	.862 ± .014	.824 ± .045	.830 ± .034	.814 ± .041	.836 ± .032
	time	0.6 ± 0.0	0.8 ± 0.1	1.9 ± 0.5	3.5 ± 0.5	3.6 ± 0.3	4.6 ± 0.5	4.8 ± 0.6
dry-bean	acc	.788 ± .007	.910 ± .007	.917 ± .007	.919 ± .006	.917 ± .008	.923 ± .005	.922 ± .006
	nodes	4.0 ± 0.0	6.0 ± 0.0	8.0 ± 0.0	9.9 ± 0.7	18.2 ± 1.0	41.2 ± 3.2	94.3 ± 3.5
	path len	2.84 ± 0.03	3.24 ± 0.05	3.40 ± 0.09	3.60 ± 0.17	4.74 ± 0.15	6.19 ± 0.34	7.62 ± 0.33
	gini	.902 ± .002	.902 ± .003	.910 ± .002	.909 ± .003	.909 ± .002	.906 ± .004	.908 ± .002
	time	3.1 ± 0.1	3.9 ± 0.1	4.9 ± 0.1	5.6 ± 0.3	8.8 ± 0.3	12.7 ± 0.9	25.2 ± 0.7
wine	acc	.961 ± .026	.972 ± .028	.972 ± .028	.972 ± .028	.966 ± .027	.977 ± .028	.972 ± .028
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.8 ± 0.4	2.7 ± 0.5	2.9 ± 0.3	2.8 ± 0.4	2.7 ± 0.5
	path len	1.67 ± 0.09	1.68 ± 0.09	1.92 ± 0.16	1.90 ± 0.16	1.95 ± 0.15	1.96 ± 0.18	1.89 ± 0.17
	gini	.871 ± .011	.875 ± .009	.884 ± .011	.885 ± .007	.891 ± .005	.886 ± .008	.885 ± .009
	time	0.5 ± 0.0	0.5 ± 0.0	0.6 ± 0.1	0.6 ± 0.1	0.7 ± 0.0	0.6 ± 0.1	0.6 ± 0.1
car	acc	.700 ± .044	.908 ± .031	.965 ± .013	.986 ± .009	.992 ± .012	.992 ± .009	.992 ± .010
	nodes	0.0 ± 0.0	2.8 ± 0.4	5.0 ± 0.8	12.3 ± 1.6	18.8 ± 2.7	23.7 ± 2.1	24.5 ± 4.1
	path len	0.00 ± 0.00	1.60 ± 0.16	1.91 ± 0.18	2.86 ± 0.34	3.23 ± 0.52	3.61 ± 0.28	3.70 ± 0.64
	gini	.000 ± .000	.905 ± .010	.897 ± .014	.902 ± .015	.902 ± .014	.903 ± .007	.901 ± .009
	time	0.2 ± 0.0	1.1 ± 0.1	1.6 ± 0.2	3.2 ± 0.4	4.7 ± 0.6	5.9 ± 0.6	6.0 ± 1.0
wdbc	acc	.965 ± .025	.967 ± .023	.960 ± .027	.965 ± .022	.967 ± .027	.951 ± .025	.961 ± .037
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	2.1 ± 0.9	5.9 ± 2.1	13.9 ± 4.2	19.6 ± 3.6
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.59 ± 0.48	2.52 ± 0.32	3.75 ± 0.69	4.51 ± 0.41
	gini	.954 ± .003	.955 ± .003	.954 ± .003	.951 ± .003	.951 ± .003	.950 ± .003	.951 ± .004
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	0.8 ± 0.2	1.6 ± 0.4	3.4 ± 0.9	4.6 ± 0.7
sonar	acc	.760 ± .099	.817 ± .045	.788 ± .081	.813 ± .092	.775 ± .076	.856 ± .038	.807 ± .063
	nodes	1.0 ± 0.0	2.2 ± 0.6	4.5 ± 0.8	5.6 ± 0.9	6.8 ± 3.0	7.6 ± 2.1	8.0 ± 2.4
	path len	1.00 ± 0.00	1.59 ± 0.24	2.52 ± 0.32	2.87 ± 0.53	3.34 ± 0.94	3.55 ± 0.71	3.56 ± 0.75
	gini	.971 ± .002	.972 ± .001	.973 ± .001	.949 ± .074	.973 ± .002	.974 ± .001	.973 ± .002
	time	0.5 ± 0.0	0.8 ± 0.2	1.3 ± 0.2	1.6 ± 0.2	1.8 ± 0.6	2.0 ± 0.4	2.1 ± 0.5
pendigits	acc	.094 ± .003	.910 ± .021	.953 ± .011	.971 ± .008	.983 ± .004	.987 ± .004	.988 ± .003
	nodes	0.0 ± 0.0	9.3 ± 0.5	12.1 ± 1.0	17.9 ± 1.7	27.4 ± 3.0	57.5 ± 6.6	125.9 ± 9.5
	path len	0.00 ± 0.00	4.18 ± 0.32	4.46 ± 0.31	5.18 ± 0.23	5.64 ± 0.45	6.59 ± 0.43	8.20 ± 0.61
	gini	.000 ± .000	.891 ± .008	.892 ± .005	.891 ± .005	.896 ± .006	.899 ± .002	.897 ± .005
	time	0.3 ± 0.0	4.2 ± 0.1	5.0 ± 0.4	6.9 ± 0.5	9.3 ± 0.8	16.7 ± 1.5	32.2 ± 2.2
ionosphere	acc	.855 ± .054	.932 ± .060	.926 ± .048	.937 ± .036	.923 ± .048	.923 ± .051	.946 ± .039
	nodes	1.0 ± 0.0	2.1 ± 0.3	3.8 ± 0.4	4.8 ± 0.7	8.5 ± 1.9	9.5 ± 1.8	11.8 ± 2.0
	path len	1.00 ± 0.00	1.78 ± 0.14	2.95 ± 0.24	3.58 ± 0.62	4.58 ± 0.55	4.77 ± 0.89	5.50 ± 0.79
	gini	.948 ± .003	.929 ± .012	.925 ± .005	.927 ± .005	.939 ± .005	.934 ± .008	.931 ± .004
	time	0.7 ± 0.0	1.0 ± 0.1	1.6 ± 0.1	1.9 ± 0.2	3.0 ± 0.6	2.2 ± 0.4	2.6 ± 0.4

Ours: prototype features, L2 regularization, non-random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.927 ± .092	.933 ± .094	.953 ± .067	.967 ± .054	.947 ± .065	.947 ± .065	.947 ± .065
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.7 ± 1.0	4.9 ± 0.8	5.7 ± 0.9	5.6 ± 0.9	5.6 ± 0.9
	path len	1.67 ± 0.09	1.67 ± 0.09	2.09 ± 0.25	2.23 ± 0.23	2.31 ± 0.26	2.28 ± 0.24	2.28 ± 0.24
	gini	.580 ± .024	.578 ± .023	.580 ± .030	.584 ± .038	.578 ± .034	.576 ± .032	.577 ± .032
	time	1.1 ± 0.0	1.1 ± 0.0	1.6 ± 0.3	1.9 ± 0.2	2.2 ± 0.3	2.1 ± 0.3	2.1 ± 0.3
heart	acc	.756 ± .061	.746 ± .055	.776 ± .054	.765 ± .065	.766 ± .058	.729 ± .069	.749 ± .046
	nodes	1.0 ± 0.0	1.1 ± 0.3	6.4 ± 0.9	25.0 ± 3.6	48.0 ± 4.3	51.4 ± 5.9	52.5 ± 5.8
	path len	1.00 ± 0.00	1.04 ± 0.13	2.80 ± 0.25	4.41 ± 0.46	5.75 ± 0.52	5.94 ± 0.32	5.84 ± 0.41
	gini	.910 ± .007	.911 ± .007	.881 ± .011	.868 ± .014	.863 ± .011	.862 ± .016	.863 ± .016
	time	0.9 ± 0.0	0.9 ± 0.1	2.5 ± 0.3	8.0 ± 1.1	14.4 ± 1.2	15.5 ± 1.6	15.8 ± 1.6
dry-bean	acc	.681 ± .119	.880 ± .016	.885 ± .016	.902 ± .007	.909 ± .003	.897 ± .010	.904 ± .013
	nodes	3.2 ± 1.0	6.1 ± 0.3	8.4 ± 0.7	16.1 ± 2.1	31.0 ± 3.7	60.2 ± 5.9	172.5 ± 12.8
	path len	2.37 ± 0.56	3.08 ± 0.15	3.37 ± 0.16	4.26 ± 0.30	5.38 ± 0.24	6.03 ± 0.29	7.53 ± 0.38
	gini	.858 ± .021	.832 ± .019	.737 ± .063	.658 ± .056	.619 ± .078	.729 ± .031	.714 ± .027
	time	3.1 ± 0.7	4.5 ± 0.1	5.8 ± 0.4	9.5 ± 1.0	16.4 ± 2.1	20.2 ± 1.8	50.1 ± 3.2
wine	acc	.939 ± .052	.898 ± .056	.893 ± .047	.939 ± .068	.871 ± .066	.894 ± .084	.933 ± .054
	nodes	2.0 ± 0.0	2.6 ± 0.7	6.1 ± 1.6	7.3 ± 2.5	8.9 ± 3.3	10.0 ± 2.6	8.8 ± 1.5
	path len	1.69 ± 0.09	1.92 ± 0.28	2.99 ± 0.49	3.08 ± 0.60	3.31 ± 0.57	3.66 ± 0.69	3.47 ± 0.34
	gini	.861 ± .019	.862 ± .020	.789 ± .059	.792 ± .051	.766 ± .060	.763 ± .025	.765 ± .044
	time	0.9 ± 0.1	1.1 ± 0.2	1.9 ± 0.4	2.2 ± 0.6	2.5 ± 0.7	2.8 ± 0.6	2.5 ± 0.3
car	acc	.700 ± .044	.700 ± .044	.701 ± .045	.780 ± .087	.794 ± .116	.825 ± .098	.821 ± .109
	nodes	0.0 ± 0.0	0.2 ± 0.4	2.7 ± 1.9	13.9 ± 7.0	43.7 ± 27.1	96.5 ± 61.5	116.4 ± 67.0
	path len	0.00 ± 0.00	0.20 ± 0.40	1.68 ± 1.06	3.63 ± 0.62	4.85 ± 0.70	6.30 ± 1.18	6.61 ± 1.10
	gini	.000 ± .000	.182 ± .363	.716 ± .358	.890 ± .016	.869 ± .015	.873 ± .015	.869 ± .017
	time	0.3 ± 0.0	0.4 ± 0.2	1.3 ± 0.6	4.2 ± 1.8	11.2 ± 6.1	23.0 ± 13.6	27.5 ± 14.4
wdbc	acc	.930 ± .047	.946 ± .045	.951 ± .030	.933 ± .041	.931 ± .028	.930 ± .034	.910 ± .033
	nodes	1.0 ± 0.0	1.0 ± 0.0	2.9 ± 0.7	9.6 ± 2.2	14.9 ± 3.8	19.5 ± 2.8	21.8 ± 3.0
	path len	1.00 ± 0.00	1.00 ± 0.00	1.83 ± 0.31	3.65 ± 0.77	4.55 ± 0.92	5.35 ± 0.81	5.81 ± 0.47
	gini	.936 ± .011	.942 ± .011	.938 ± .008	.888 ± .018	.875 ± .023	.855 ± .034	.866 ± .016
	time	0.7 ± 0.0	0.7 ± 0.0	1.2 ± 0.2	2.9 ± 0.6	4.2 ± 0.9	5.3 ± 0.7	5.9 ± 0.8
sonar	acc	.716 ± .111	.730 ± .050	.721 ± .052	.730 ± .088	.711 ± .114	.721 ± .089	.770 ± .075
	nodes	1.0 ± 0.0	4.4 ± 1.0	11.3 ± 1.3	16.2 ± 3.1	18.0 ± 1.8	18.6 ± 2.7	18.0 ± 2.9
	path len	1.00 ± 0.00	2.32 ± 0.35	4.11 ± 0.51	4.92 ± 0.82	5.06 ± 0.61	5.05 ± 0.75	5.14 ± 0.97
	gini	.888 ± .069	.929 ± .026	.921 ± .029	.908 ± .036	.911 ± .038	.902 ± .025	.905 ± .031
	time	0.7 ± 0.0	1.6 ± 0.2	3.1 ± 0.3	4.3 ± 0.7	4.7 ± 0.4	4.8 ± 0.6	4.6 ± 0.7
pendigits	acc	.094 ± .003	.709 ± .031	.811 ± .017	.865 ± .015	.909 ± .011	.926 ± .011	.931 ± .007
	nodes	0.0 ± 0.0	8.1 ± 0.7	15.6 ± 1.7	31.8 ± 2.3	75.5 ± 3.5	174.3 ± 12.1	402.2 ± 24.6
	path len	0.00 ± 0.00	3.74 ± 0.20	4.60 ± 0.23	5.66 ± 0.25	6.79 ± 0.28	7.72 ± 0.25	9.76 ± 0.39
	gini	.000 ± .000	.558 ± .083	.428 ± .108	.348 ± .094	.299 ± .073	.283 ± .049	.241 ± .042
	time	0.4 ± 0.0	4.4 ± 0.4	7.0 ± 0.7	12.0 ± 0.8	23.9 ± 1.1	48.5 ± 3.0	101.7 ± 6.1
ionosphere	acc	.878 ± .057	.889 ± .050	.923 ± .036	.903 ± .043	.903 ± .045	.903 ± .041	.900 ± .039
	nodes	1.0 ± 0.0	1.9 ± 0.5	4.0 ± 1.3	9.9 ± 3.2	19.3 ± 3.8	20.6 ± 3.1	21.6 ± 4.1
	path len	1.00 ± 0.00	1.65 ± 0.40	2.60 ± 0.55	3.92 ± 0.65	6.19 ± 1.03	6.44 ± 1.23	6.81 ± 1.56
	gini	.748 ± .009	.771 ± .047	.765 ± .023	.770 ± .037	.780 ± .026	.794 ± .023	.764 ± .047
	time	0.9 ± 0.0	1.3 ± 0.2	2.0 ± 0.4	4.0 ± 1.1	5.1 ± 1.0	7.2 ± 1.0	7.3 ± 1.2

Ours: prototype features, L2 regularization, non-random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.927 ± .092	.933 ± .094	.953 ± .067	.973 ± .044	.953 ± .060	.953 ± .060	.953 ± .060
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.7 ± 1.0	4.9 ± 0.8	5.7 ± 0.9	5.6 ± 0.9	5.6 ± 0.9
	path len	1.67 ± 0.09	1.67 ± 0.09	2.09 ± 0.25	2.23 ± 0.23	2.31 ± 0.26	2.29 ± 0.24	2.29 ± 0.24
	gini	.580 ± .024	.578 ± .023	.580 ± .030	.584 ± .038	.578 ± .034	.576 ± .032	.577 ± .032
	time	1.0 ± 0.0	1.0 ± 0.0	1.5 ± 0.3	1.8 ± 0.2	2.0 ± 0.2	2.0 ± 0.2	2.0 ± 0.2
heart	acc	.756 ± .061	.746 ± .055	.779 ± .055	.769 ± .060	.766 ± .065	.736 ± .069	.743 ± .051
	nodes	1.0 ± 0.0	1.1 ± 0.3	6.4 ± 0.9	24.9 ± 3.5	48.0 ± 4.3	51.4 ± 5.9	52.5 ± 5.8
	path len	1.00 ± 0.00	1.04 ± 0.13	2.80 ± 0.25	4.41 ± 0.46	5.75 ± 0.53	5.94 ± 0.32	5.84 ± 0.41
	gini	.910 ± .007	.911 ± .007	.881 ± .011	.868 ± .014	.863 ± .011	.862 ± .016	.863 ± .016
	time	0.8 ± 0.1	0.8 ± 0.1	2.3 ± 0.3	7.1 ± 0.9	12.9 ± 1.1	20.5 ± 2.2	14.2 ± 1.4
dry-bean	acc	.680 ± .118	.877 ± .016	.883 ± .015	.894 ± .006	.902 ± .004	.894 ± .010	.902 ± .013
	nodes	3.2 ± 1.0	6.1 ± 0.3	8.4 ± 0.7	16.1 ± 2.1	31.0 ± 3.7	60.2 ± 5.9	172.5 ± 12.8
	path len	2.37 ± 0.57	3.08 ± 0.15	3.38 ± 0.16	4.26 ± 0.31	5.39 ± 0.24	6.03 ± 0.30	7.54 ± 0.39
	gini	.858 ± .021	.832 ± .019	.738 ± .063	.659 ± .057	.618 ± .080	.730 ± .032	.714 ± .027
	time	2.9 ± 0.6	4.3 ± 0.1	5.2 ± 0.3	8.3 ± 0.9	13.5 ± 1.5	18.2 ± 1.4	43.6 ± 2.9
wine	acc	.933 ± .060	.898 ± .056	.893 ± .047	.939 ± .068	.882 ± .076	.894 ± .084	.933 ± .054
	nodes	2.0 ± 0.0	2.6 ± 0.7	6.1 ± 1.6	7.3 ± 2.5	8.9 ± 3.3	10.0 ± 2.6	8.8 ± 1.5
	path len	1.69 ± 0.09	1.92 ± 0.28	3.01 ± 0.49	3.07 ± 0.59	3.30 ± 0.59	3.68 ± 0.69	3.46 ± 0.35
	gini	.861 ± .019	.862 ± .020	.790 ± .057	.793 ± .051	.764 ± .062	.763 ± .026	.766 ± .044
	time	0.9 ± 0.1	1.0 ± 0.1	1.7 ± 0.3	2.0 ± 0.5	2.3 ± 0.7	2.5 ± 0.5	2.3 ± 0.3
car	acc	.700 ± .044	.700 ± .044	.701 ± .045	.780 ± .087	.795 ± .118	.824 ± .096	.821 ± .114
	nodes	0.0 ± 0.0	0.2 ± 0.4	2.7 ± 1.9	13.9 ± 7.0	43.7 ± 27.1	96.5 ± 61.5	116.6 ± 67.2
	path len	0.00 ± 0.00	0.20 ± 0.40	1.68 ± 1.06	3.64 ± 0.62	4.85 ± 0.70	6.30 ± 1.18	6.60 ± 1.10
	gini	.000 ± .000	.182 ± .363	.716 ± .358	.890 ± .016	.869 ± .015	.873 ± .015	.869 ± .017
	time	0.3 ± 0.0	0.4 ± 0.2	1.2 ± 0.6	3.8 ± 1.6	10.1 ± 5.5	20.2 ± 11.9	24.3 ± 12.9
wdbc	acc	.930 ± .047	.946 ± .045	.947 ± .029	.933 ± .041	.930 ± .027	.930 ± .034	.910 ± .033
	nodes	1.0 ± 0.0	1.0 ± 0.0	2.9 ± 0.7	9.6 ± 2.2	14.9 ± 3.8	19.5 ± 2.8	21.8 ± 3.0
	path len	1.00 ± 0.00	1.00 ± 0.00	1.83 ± 0.31	3.65 ± 0.78	4.55 ± 0.91	5.35 ± 0.81	5.82 ± 0.48
	gini	.936 ± .011	.942 ± .011	.938 ± .008	.888 ± .019	.876 ± .020	.855 ± .034	.865 ± .016
	time	0.7 ± 0.0	0.7 ± 0.0	1.1 ± 0.2	2.6 ± 0.5	3.8 ± 0.9	4.8 ± 0.6	5.3 ± 0.6
sonar	acc	.716 ± .111	.730 ± .050	.716 ± .050	.730 ± .088	.716 ± .113	.730 ± .092	.770 ± .078
	nodes	1.0 ± 0.0	4.4 ± 1.0	11.3 ± 1.3	16.2 ± 3.1	18.0 ± 1.8	18.6 ± 2.7	18.0 ± 2.9
	path len	1.00 ± 0.00	2.32 ± 0.35	4.10 ± 0.51	4.92 ± 0.82	5.07 ± 0.62	5.07 ± 0.73	5.16 ± 0.96
	gini	.888 ± .069	.929 ± .026	.921 ± .029	.908 ± .036	.911 ± .038	.903 ± .024	.905 ± .031
	time	0.7 ± 0.0	1.4 ± 0.2	2.8 ± 0.3	3.9 ± 0.6	4.2 ± 0.3	4.3 ± 0.5	4.2 ± 0.6
pendigits	acc	.094 ± .003	.708 ± .031	.809 ± .018	.864 ± .015	.908 ± .011	.925 ± .011	.930 ± .007
	nodes	0.0 ± 0.0	8.1 ± 0.7	15.6 ± 1.7	31.8 ± 2.3	75.5 ± 3.5	174.3 ± 12.1	402.2 ± 24.6
	path len	0.00 ± 0.00	3.74 ± 0.21	4.60 ± 0.23	5.66 ± 0.25	6.79 ± 0.28	7.72 ± 0.25	9.76 ± 0.38
	gini	.000 ± .000	.558 ± .083	.428 ± .108	.348 ± .094	.300 ± .073	.283 ± .049	.241 ± .042
	time	0.3 ± 0.0	4.0 ± 0.3	6.4 ± 0.6	10.9 ± 0.7	21.5 ± 0.9	43.2 ± 2.7	89.6 ± 5.1
ionosphere	acc	.878 ± .057	.892 ± .049	.923 ± .036	.903 ± .043	.897 ± .048	.903 ± .041	.900 ± .039
	nodes	1.0 ± 0.0	1.9 ± 0.5	4.0 ± 1.3	9.9 ± 3.2	19.2 ± 3.8	20.6 ± 3.1	21.6 ± 4.1
	path len	1.00 ± 0.00	1.65 ± 0.40	2.60 ± 0.55	3.92 ± 0.65	6.20 ± 1.03	6.44 ± 1.23	6.80 ± 1.57
	gini	.748 ± .009	.771 ± .047	.765 ± .023	.770 ± .037	.782 ± .027	.794 ± .023	.764 ± .047
	time	0.7 ± 0.0	0.9 ± 0.2	1.3 ± 0.2	2.6 ± 0.7	4.6 ± 0.8	6.6 ± 0.9	6.7 ± 1.2

Ours: linear features, L1 regularization, non-random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.953 ± .043	.967 ± .033	.967 ± .033	.960 ± .033	.953 ± .043	.953 ± .043
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.7 ± 0.5	5.0 ± 1.2	9.5 ± 1.3	13.9 ± 2.2	18.8 ± 4.1
	path len	1.67 ± 0.09	1.67 ± 0.09	2.25 ± 0.21	2.37 ± 0.34	3.09 ± 0.42	3.51 ± 0.45	3.86 ± 0.57
	gini	.067 ± .168	.072 ± .168	.106 ± .049	.077 ± .051	.095 ± .077	.082 ± .086	.077 ± .072
	time	0.6 ± 0.0	0.7 ± 0.0	1.0 ± 0.1	1.6 ± 0.3	2.6 ± 0.3	3.6 ± 0.5	4.7 ± 0.9
heart	acc	.772 ± .056	.799 ± .052	.796 ± .065	.772 ± .036	.763 ± .074	.766 ± .082	.766 ± .096
	nodes	1.0 ± 0.0	1.3 ± 0.5	6.4 ± 1.5	25.5 ± 3.8	36.2 ± 3.8	41.3 ± 3.2	42.9 ± 5.8
	path len	1.00 ± 0.00	1.19 ± 0.30	2.62 ± 0.28	4.57 ± 0.35	5.20 ± 0.34	5.47 ± 0.44	5.65 ± 0.56
	gini	.432 ± .072	.505 ± .139	.347 ± .101	.220 ± .081	.164 ± .039	.182 ± .069	.165 ± .051
	time	0.6 ± 0.0	0.7 ± 0.2	1.8 ± 0.5	6.4 ± 2.0	7.1 ± 0.7	8.1 ± 0.6	8.4 ± 1.1
dry-bean	acc	.636 ± .062	.902 ± .009	.914 ± .007	.912 ± .004	.915 ± .006	.920 ± .007	.920 ± .007
	nodes	2.8 ± 0.4	6.0 ± 0.0	7.7 ± 0.5	12.1 ± 1.4	27.7 ± 2.0	59.6 ± 2.0	88.1 ± 10.7
	path len	2.14 ± 0.25	3.13 ± 0.22	3.31 ± 0.12	3.83 ± 0.19	5.28 ± 0.23	6.37 ± 0.14	6.46 ± 0.31
	gini	.619 ± .031	.658 ± .019	.674 ± .026	.535 ± .076	.510 ± .043	.359 ± .042	.406 ± .033
	time	2.6 ± 0.3	4.5 ± 0.1	5.3 ± 0.3	7.3 ± 0.7	14.5 ± 0.8	20.9 ± 0.7	51.1 ± 24.9
wine	acc	.972 ± .028	.989 ± .022	.972 ± .028	.960 ± .026	.966 ± .027	.983 ± .026	.978 ± .027
	nodes	2.0 ± 0.0	2.1 ± 0.3	3.0 ± 0.4	3.0 ± 0.4	5.0 ± 1.5	9.1 ± 2.7	13.1 ± 3.0
	path len	1.68 ± 0.08	1.72 ± 0.21	2.10 ± 0.18	2.03 ± 0.17	2.55 ± 0.36	3.39 ± 0.48	4.23 ± 0.57
	gini	.795 ± .015	.792 ± .034	.754 ± .043	.749 ± .062	.752 ± .060	.737 ± .033	.733 ± .025
	time	0.6 ± 0.0	0.6 ± 0.0	0.8 ± 0.1	0.7 ± 0.1	1.1 ± 0.3	1.8 ± 0.5	2.5 ± 0.5
car	acc	.700 ± .044	.891 ± .032	.946 ± .038	.976 ± .013	.992 ± .006	.996 ± .006	.992 ± .006
	nodes	0.0 ± 0.0	2.5 ± 0.8	5.9 ± 1.5	14.0 ± 2.4	29.8 ± 6.0	32.4 ± 3.7	39.3 ± 7.5
	path len	0.00 ± 0.00	1.53 ± 0.29	2.08 ± 0.26	2.77 ± 0.14	3.39 ± 0.25	3.64 ± 0.42	3.64 ± 0.34
	gini	.000 ± .000	.580 ± .133	.566 ± .035	.542 ± .071	.580 ± .082	.572 ± .062	.507 ± .043
	time	0.2 ± 0.0	1.1 ± 0.3	1.9 ± 0.4	4.0 ± 0.7	8.0 ± 1.7	8.6 ± 1.2	10.2 ± 1.5
wdbc	acc	.975 ± .020	.974 ± .021	.974 ± .016	.968 ± .015	.970 ± .019	.967 ± .029	.963 ± .030
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	3.0 ± 0.0	7.2 ± 2.5	17.1 ± 4.4	31.5 ± 4.7
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	2.00 ± 0.00	2.79 ± 0.62	3.55 ± 0.63	4.67 ± 0.62
	gini	.816 ± .018	.812 ± .023	.810 ± .021	.824 ± .034	.933 ± .007	.936 ± .013	.937 ± .008
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	1.1 ± 0.0	2.2 ± 0.6	3.2 ± 0.8	8.2 ± 1.5
sonar	acc	.711 ± .088	.769 ± .095	.822 ± .058	.783 ± .075	.803 ± .083	.851 ± .069	.836 ± .081
	nodes	1.0 ± 0.0	2.5 ± 0.7	7.3 ± 1.2	7.2 ± 1.0	13.3 ± 3.8	18.7 ± 4.8	27.5 ± 6.2
	path len	1.00 ± 0.00	1.75 ± 0.19	3.04 ± 0.25	3.15 ± 0.44	3.93 ± 0.51	4.90 ± 0.68	5.63 ± 0.63
	gini	.946 ± .008	.910 ± .028	.854 ± .043	.871 ± .037	.893 ± .041	.920 ± .022	.928 ± .018
	time	0.5 ± 0.0	1.0 ± 0.2	2.1 ± 0.3	2.1 ± 0.3	3.6 ± 1.0	5.0 ± 1.3	5.9 ± 1.5
pendigits	acc	.094 ± .003	.888 ± .014	.942 ± .010	.964 ± .007	.974 ± .007	.977 ± .005	.984 ± .004
	nodes	0.0 ± 0.0	9.1 ± 0.3	11.7 ± 0.9	18.6 ± 1.5	42.1 ± 5.0	120.7 ± 7.1	290.0 ± 12.9
	path len	0.00 ± 0.00	3.83 ± 0.27	4.40 ± 0.39	5.08 ± 0.20	5.92 ± 0.29	7.49 ± 0.37	8.95 ± 0.31
	gini	.000 ± .000	.803 ± .013	.795 ± .027	.751 ± .014	.689 ± .042	.503 ± .029	.329 ± .032
	time	0.3 ± 0.0	4.5 ± 0.1	5.5 ± 0.3	7.9 ± 0.5	15.2 ± 1.5	55.7 ± 12.3	142.1 ± 25.0
ionosphere	acc	.829 ± .044	.895 ± .057	.920 ± .038	.923 ± .034	.906 ± .056	.897 ± .069	.937 ± .031
	nodes	1.0 ± 0.0	2.2 ± 0.4	4.4 ± 0.5	6.2 ± 1.2	14.9 ± 3.1	26.4 ± 4.8	30.4 ± 5.4
	path len	1.00 ± 0.00	1.91 ± 0.32	3.43 ± 0.41	3.89 ± 0.44	5.51 ± 0.84	7.59 ± 0.87	8.98 ± 1.28
	gini	.447 ± .056	.640 ± .147	.532 ± .130	.657 ± .071	.691 ± .060	.772 ± .080	.781 ± .076
	time	0.5 ± 0.0	0.9 ± 0.1	1.4 ± 0.1	1.8 ± 0.3	3.9 ± 0.8	6.3 ± 1.1	7.9 ± 1.5

Ours: linear features, L1 regularization, non-random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.953 ± .043	.980 ± .031	.960 ± .033	.973 ± .033	.920 ± .065	.967 ± .033
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.7 ± 0.5	5.0 ± 1.2	9.5 ± 1.3	13.9 ± 2.2	18.8 ± 4.1
	path len	1.67 ± 0.09	1.67 ± 0.09	2.25 ± 0.21	2.38 ± 0.33	3.05 ± 0.44	3.52 ± 0.45	3.87 ± 0.55
	gini	.063 ± .170	.071 ± .168	.101 ± .046	.079 ± .057	.091 ± .075	.079 ± .086	.063 ± .046
	time	0.7 ± 0.0	0.7 ± 0.0	1.1 ± 0.1	1.4 ± 0.2	2.3 ± 0.3	3.1 ± 0.4	4.1 ± 0.8
heart	acc	.772 ± .056	.789 ± .056	.796 ± .061	.779 ± .034	.766 ± .078	.766 ± .078	.762 ± .095
	nodes	1.0 ± 0.0	1.3 ± 0.5	6.4 ± 1.5	25.5 ± 3.8	36.1 ± 3.8	41.3 ± 3.2	42.9 ± 5.8
	path len	1.00 ± 0.00	1.19 ± 0.30	2.61 ± 0.30	4.55 ± 0.37	5.21 ± 0.31	5.47 ± 0.42	5.61 ± 0.53
	gini	.432 ± .072	.505 ± .140	.346 ± .102	.220 ± .081	.165 ± .039	.181 ± .070	.165 ± .051
	time	0.6 ± 0.0	0.7 ± 0.1	1.6 ± 0.5	6.0 ± 1.6	6.2 ± 0.6	7.1 ± 0.5	7.4 ± 1.0
dry-bean	acc	.632 ± .061	.891 ± .010	.910 ± .010	.905 ± .006	.906 ± .006	.913 ± .009	.908 ± .008
	nodes	2.8 ± 0.4	6.0 ± 0.0	7.7 ± 0.5	12.1 ± 1.4	27.7 ± 2.0	44.1 ± 3.2	88.1 ± 10.7
	path len	2.15 ± 0.25	3.15 ± 0.23	3.31 ± 0.12	3.85 ± 0.20	5.33 ± 0.25	5.57 ± 0.17	6.49 ± 0.32
	gini	.619 ± .031	.658 ± .019	.674 ± .026	.536 ± .076	.511 ± .044	.360 ± .038	.406 ± .033
	time	2.4 ± 0.3	4.0 ± 0.1	4.8 ± 0.3	6.3 ± 0.5	11.5 ± 0.6	42.8 ± 15.5	50.7 ± 14.2
wine	acc	.972 ± .028	.989 ± .022	.972 ± .028	.955 ± .034	.961 ± .026	.978 ± .027	.972 ± .028
	nodes	2.0 ± 0.0	2.1 ± 0.3	3.0 ± 0.4	3.0 ± 0.4	5.0 ± 1.5	9.1 ± 2.7	13.1 ± 3.0
	path len	1.68 ± 0.07	1.73 ± 0.21	2.09 ± 0.18	2.03 ± 0.17	2.53 ± 0.37	3.41 ± 0.49	4.24 ± 0.58
	gini	.795 ± .015	.792 ± .033	.755 ± .042	.749 ± .062	.752 ± .063	.737 ± .034	.733 ± .025
	time	0.5 ± 0.0	0.5 ± 0.0	0.7 ± 0.1	0.7 ± 0.1	1.0 ± 0.2	1.6 ± 0.4	2.2 ± 0.4
car	acc	.700 ± .044	.890 ± .033	.945 ± .038	.976 ± .013	.992 ± .006	.996 ± .006	.992 ± .006
	nodes	0.0 ± 0.0	2.5 ± 0.8	5.9 ± 1.5	14.0 ± 2.4	29.8 ± 6.0	32.4 ± 3.7	39.3 ± 7.5
	path len	0.00 ± 0.00	1.53 ± 0.29	2.09 ± 0.26	2.78 ± 0.14	3.39 ± 0.25	3.64 ± 0.42	3.65 ± 0.35
	gini	.000 ± .000	.580 ± .133	.566 ± .036	.542 ± .071	.580 ± .081	.573 ± .062	.507 ± .043
	time	0.2 ± 0.0	1.0 ± 0.2	1.8 ± 0.4	3.6 ± 0.6	7.1 ± 1.5	7.6 ± 1.0	9.1 ± 1.5
wdbc	acc	.975 ± .020	.974 ± .021	.974 ± .016	.968 ± .015	.965 ± .025	.963 ± .035	.961 ± .028
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	3.0 ± 0.0	7.2 ± 2.5	17.1 ± 4.4	31.5 ± 4.7
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	2.00 ± 0.00	2.80 ± 0.62	3.54 ± 0.66	4.71 ± 0.58
	gini	.816 ± .018	.812 ± .023	.810 ± .021	.824 ± .034	.933 ± .007	.936 ± .013	.937 ± .008
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	1.0 ± 0.0	1.9 ± 0.5	2.8 ± 0.6	7.0 ± 1.2
sonar	acc	.712 ± .100	.769 ± .083	.812 ± .063	.774 ± .105	.818 ± .062	.846 ± .083	.822 ± .098
	nodes	1.0 ± 0.0	2.5 ± 0.7	7.3 ± 1.2	7.2 ± 1.0	13.3 ± 3.8	18.7 ± 4.8	27.5 ± 6.2
	path len	1.00 ± 0.00	1.77 ± 0.18	3.04 ± 0.27	3.16 ± 0.46	3.87 ± 0.53	4.84 ± 0.65	5.64 ± 0.61
	gini	.946 ± .008	.910 ± .028	.853 ± .044	.872 ± .036	.894 ± .042	.920 ± .023	.928 ± .019
	time	0.5 ± 0.0	0.9 ± 0.2	1.9 ± 0.2	1.9 ± 0.2	3.2 ± 0.8	4.3 ± 1.1	5.2 ± 1.2
pendigits	acc	.094 ± .003	.885 ± .013	.938 ± .011	.960 ± .010	.968 ± .005	.969 ± .006	.976 ± .004
	nodes	0.0 ± 0.0	9.1 ± 0.3	11.7 ± 0.9	18.6 ± 1.5	42.1 ± 5.0	120.7 ± 7.1	290.0 ± 12.9
	path len	0.00 ± 0.00	3.83 ± 0.27	4.41 ± 0.39	5.08 ± 0.20	5.91 ± 0.29	7.49 ± 0.40	8.97 ± 0.34
	gini	.000 ± .000	.803 ± .013	.795 ± .027	.751 ± .014	.689 ± .042	.503 ± .029	.329 ± .032
	time	0.3 ± 0.0	5.2 ± 0.2	6.3 ± 0.4	9.1 ± 0.6	17.4 ± 1.7	40.2 ± 2.4	86.1 ± 5.0
ionosphere	acc	.835 ± .044	.892 ± .057	.917 ± .043	.920 ± .038	.897 ± .054	.909 ± .055	.909 ± .069
	nodes	1.0 ± 0.0	2.2 ± 0.4	4.4 ± 0.5	6.2 ± 1.2	14.9 ± 3.1	26.4 ± 4.8	30.4 ± 5.4
	path len	1.00 ± 0.00	1.92 ± 0.32	3.51 ± 0.44	3.94 ± 0.44	5.54 ± 0.85	7.67 ± 0.90	9.23 ± 1.41
	gini	.447 ± .056	.640 ± .147	.533 ± .129	.658 ± .070	.691 ± .060	.772 ± .081	.781 ± .077
	time	0.5 ± 0.0	0.8 ± 0.1	1.2 ± 0.1	1.6 ± 0.3	3.4 ± 0.6	5.8 ± 1.0	6.7 ± 1.2

Ours: prototype features, L1 regularization, non-random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.940 ± .055	.940 ± .055	.933 ± .067	.933 ± .052	.927 ± .063	.900 ± .086	.920 ± .088
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.3 ± 0.8	5.3 ± 0.6	5.9 ± 0.9	6.3 ± 1.0	6.1 ± 0.8
	path len	1.66 ± 0.09	1.66 ± 0.09	2.02 ± 0.30	2.24 ± 0.23	2.30 ± 0.14	2.34 ± 0.19	2.32 ± 0.22
	gini	.571 ± .025	.567 ± .024	.542 ± .042	.535 ± .029	.552 ± .045	.538 ± .045	.527 ± .046
	time	1.1 ± 0.0	1.1 ± 0.0	1.3 ± 0.3	2.1 ± 0.2	2.2 ± 0.3	2.3 ± 0.3	2.3 ± 0.2
heart	acc	.756 ± .065	.743 ± .060	.759 ± .071	.786 ± .063	.746 ± .066	.753 ± .061	.736 ± .062
	nodes	1.0 ± 0.0	1.3 ± 0.6	7.0 ± 2.2	27.3 ± 5.2	46.9 ± 6.7	52.4 ± 6.2	54.1 ± 6.2
	path len	1.00 ± 0.00	1.13 ± 0.29	2.88 ± 0.57	4.73 ± 0.68	5.61 ± 0.69	6.27 ± 0.53	6.33 ± 0.62
	gini	.912 ± .009	.908 ± .015	.872 ± .022	.852 ± .019	.844 ± .017	.850 ± .014	.853 ± .015
	time	0.9 ± 0.1	1.0 ± 0.2	2.7 ± 0.7	8.7 ± 1.5	11.0 ± 1.5	12.4 ± 1.4	12.6 ± 1.3
dry-bean	acc	.714 ± .086	.863 ± .007	.871 ± .014	.878 ± .010	.890 ± .010	.893 ± .009	.895 ± .010
	nodes	3.6 ± 0.7	6.6 ± 0.5	8.5 ± 0.9	13.9 ± 1.8	26.0 ± 2.1	43.0 ± 7.4	97.5 ± 12.4
	path len	2.36 ± 0.31	3.17 ± 0.17	3.50 ± 0.23	4.23 ± 0.29	5.07 ± 0.25	5.44 ± 0.30	6.73 ± 0.31
	gini	.830 ± .033	.806 ± .029	.719 ± .047	.583 ± .066	.524 ± .066	.516 ± .052	.509 ± .030
	time	3.3 ± 0.4	4.9 ± 0.3	6.0 ± 0.6	8.4 ± 0.9	13.9 ± 1.3	70.2 ± 22.0	120.3 ± 23.2
wine	acc	.893 ± .030	.927 ± .051	.905 ± .050	.911 ± .051	.938 ± .047	.922 ± .044	.949 ± .060
	nodes	2.0 ± 0.0	2.4 ± 0.5	5.7 ± 2.1	7.4 ± 1.9	8.1 ± 1.0	9.2 ± 2.4	10.7 ± 3.0
	path len	1.67 ± 0.12	1.79 ± 0.21	2.77 ± 0.64	3.12 ± 0.37	3.21 ± 0.23	3.53 ± 0.39	3.73 ± 0.50
	gini	.815 ± .043	.811 ± .054	.729 ± .056	.699 ± .054	.677 ± .082	.685 ± .068	.680 ± .057
	time	0.9 ± 0.1	1.0 ± 0.1	1.8 ± 0.5	2.2 ± 0.4	2.4 ± 0.2	2.6 ± 0.5	2.9 ± 0.7
car	acc	.700 ± .044	.700 ± .044	.740 ± .098	.780 ± .102	.800 ± .094	.806 ± .111	.790 ± .126
	nodes	0.0 ± 0.0	0.3 ± 0.5	4.9 ± 3.2	15.4 ± 5.8	39.5 ± 23.4	83.5 ± 57.5	105.4 ± 59.8
	path len	0.00 ± 0.00	0.30 ± 0.46	2.43 ± 1.05	4.16 ± 0.92	5.10 ± 0.92	6.22 ± 1.67	6.62 ± 1.34
	gini	.000 ± .000	.266 ± .407	.896 ± .015	.889 ± .012	.879 ± .014	.870 ± .015	.871 ± .012
	time	0.3 ± 0.0	0.4 ± 0.2	1.9 ± 0.9	4.8 ± 1.4	10.4 ± 5.5	20.6 ± 13.0	25.3 ± 13.1
wdbc	acc	.937 ± .029	.942 ± .029	.942 ± .029	.928 ± .041	.928 ± .036	.924 ± .038	.930 ± .044
	nodes	1.0 ± 0.0	1.1 ± 0.3	2.9 ± 0.8	10.0 ± 2.9	18.2 ± 1.6	22.2 ± 2.8	25.7 ± 2.5
	path len	1.00 ± 0.00	1.04 ± 0.13	1.77 ± 0.37	3.37 ± 0.62	4.56 ± 0.60	5.03 ± 0.85	6.05 ± 0.94
	gini	.917 ± .022	.915 ± .022	.925 ± .012	.878 ± .025	.842 ± .023	.816 ± .027	.786 ± .054
	time	0.7 ± 0.0	0.8 ± 0.1	1.2 ± 0.2	3.0 ± 0.8	5.0 ± 0.5	6.0 ± 0.7	6.9 ± 0.6
sonar	acc	.716 ± .102	.735 ± .085	.764 ± .052	.754 ± .084	.764 ± .118	.774 ± .084	.763 ± .089
	nodes	1.0 ± 0.0	3.9 ± 0.7	11.6 ± 1.4	19.2 ± 2.7	18.7 ± 2.1	18.9 ± 2.2	19.5 ± 2.2
	path len	1.00 ± 0.00	2.25 ± 0.20	4.01 ± 0.46	5.25 ± 0.55	5.10 ± 0.63	5.07 ± 0.65	5.07 ± 0.73
	gini	.888 ± .069	.921 ± .035	.890 ± .037	.875 ± .034	.891 ± .020	.888 ± .044	.886 ± .025
	time	0.7 ± 0.0	1.5 ± 0.2	3.2 ± 0.3	4.9 ± 0.7	4.8 ± 0.6	4.8 ± 0.5	5.0 ± 0.5
pendigits	acc	.094 ± .003	.718 ± .041	.791 ± .024	.869 ± .018	.902 ± .011	.922 ± .010	.931 ± .006
	nodes	0.0 ± 0.0	8.4 ± 0.9	15.0 ± 1.6	34.3 ± 2.2	78.0 ± 3.3	180.0 ± 12.1	408.8 ± 21.5
	path len	0.00 ± 0.00	3.81 ± 0.27	4.46 ± 0.26	5.79 ± 0.17	6.77 ± 0.26	8.06 ± 0.30	9.26 ± 0.19
	gini	.000 ± .000	.527 ± .070	.407 ± .104	.317 ± .071	.294 ± .063	.295 ± .047	.288 ± .036
	time	0.4 ± 0.0	5.6 ± 0.6	8.8 ± 0.8	17.0 ± 0.9	33.9 ± 1.2	78.3 ± 12.4	143.9 ± 8.4
ionosphere	acc	.875 ± .048	.903 ± .055	.914 ± .036	.906 ± .060	.903 ± .045	.912 ± .049	.897 ± .050
	nodes	1.0 ± 0.0	1.9 ± 0.3	3.7 ± 1.4	10.5 ± 2.4	17.1 ± 4.1	22.3 ± 3.3	21.3 ± 3.0
	path len	1.00 ± 0.00	1.64 ± 0.22	2.42 ± 0.49	4.20 ± 0.86	6.03 ± 1.07	7.26 ± 1.46	7.16 ± 1.34
	gini	.750 ± .009	.767 ± .040	.769 ± .031	.768 ± .038	.783 ± .042	.774 ± .045	.787 ± .045
	time	0.7 ± 0.0	1.0 ± 0.1	1.3 ± 0.3	2.9 ± 0.6	4.5 ± 1.0	14.4 ± 2.0	7.4 ± 1.0

Ours: prototype features, L1 regularization, non-random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.940 ± .055	.940 ± .055	.933 ± .067	.927 ± .055	.927 ± .063	.900 ± .086	.933 ± .067
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.3 ± 0.8	5.3 ± 0.6	5.9 ± 0.9	6.3 ± 1.0	6.1 ± 0.8
	path len	1.66 ± 0.09	1.66 ± 0.09	2.02 ± 0.30	2.27 ± 0.22	2.30 ± 0.15	2.35 ± 0.18	2.27 ± 0.18
	gini	.571 ± .025	.569 ± .025	.542 ± .042	.524 ± .052	.552 ± .045	.539 ± .045	.535 ± .036
	time	1.0 ± 0.0	1.0 ± 0.0	1.4 ± 0.2	1.9 ± 0.2	2.0 ± 0.2	2.1 ± 0.3	2.1 ± 0.2
heart	acc	.756 ± .065	.743 ± .060	.759 ± .071	.782 ± .066	.743 ± .073	.753 ± .060	.733 ± .059
	nodes	1.0 ± 0.0	1.3 ± 0.6	7.0 ± 2.2	27.3 ± 5.2	46.9 ± 6.7	52.4 ± 6.2	54.1 ± 6.2
	path len	1.00 ± 0.00	1.14 ± 0.29	2.88 ± 0.57	4.73 ± 0.68	5.60 ± 0.69	6.27 ± 0.53	6.34 ± 0.62
	gini	.912 ± .009	.908 ± .015	.872 ± .022	.852 ± .019	.844 ± .016	.850 ± .014	.853 ± .015
	time	0.7 ± 0.0	0.8 ± 0.2	2.0 ± 0.5	6.1 ± 1.2	9.5 ± 1.2	10.6 ± 1.2	11.0 ± 1.1
dry-bean	acc	.713 ± .086	.863 ± .007	.870 ± .016	.876 ± .011	.886 ± .010	.890 ± .008	.891 ± .009
	nodes	3.6 ± 0.7	6.6 ± 0.5	8.5 ± 0.9	13.9 ± 1.8	26.0 ± 2.1	43.0 ± 7.4	97.5 ± 12.4
	path len	2.37 ± 0.31	3.17 ± 0.17	3.50 ± 0.22	4.23 ± 0.29	5.07 ± 0.25	5.45 ± 0.31	6.74 ± 0.32
	gini	.830 ± .033	.806 ± .029	.720 ± .047	.583 ± .066	.525 ± .066	.516 ± .052	.509 ± .030
	time	3.0 ± 0.4	4.6 ± 0.3	5.5 ± 0.5	7.7 ± 0.8	12.4 ± 1.0	65.5 ± 21.2	75.9 ± 20.9
wine	acc	.899 ± .033	.927 ± .051	.905 ± .050	.905 ± .050	.938 ± .047	.911 ± .051	.949 ± .060
	nodes	2.0 ± 0.0	2.4 ± 0.5	5.7 ± 2.1	7.4 ± 1.9	8.1 ± 1.0	9.2 ± 2.4	10.7 ± 3.0
	path len	1.66 ± 0.12	1.78 ± 0.22	2.77 ± 0.64	3.12 ± 0.38	3.19 ± 0.22	3.54 ± 0.41	3.70 ± 0.49
	gini	.815 ± .043	.811 ± .054	.730 ± .056	.697 ± .050	.680 ± .075	.684 ± .069	.682 ± .055
	time	0.9 ± 0.1	1.0 ± 0.1	1.7 ± 0.4	2.0 ± 0.4	2.1 ± 0.2	2.4 ± 0.5	2.6 ± 0.6
car	acc	.700 ± .044	.700 ± .044	.740 ± .098	.779 ± .103	.797 ± .096	.807 ± .111	.794 ± .126
	nodes	0.0 ± 0.0	0.3 ± 0.5	4.9 ± 3.2	15.4 ± 5.8	39.5 ± 23.4	83.5 ± 57.5	105.4 ± 59.8
	path len	0.00 ± 0.00	0.30 ± 0.46	2.43 ± 1.05	4.16 ± 0.92	5.10 ± 0.92	6.22 ± 1.67	6.62 ± 1.34
	gini	.000 ± .000	.266 ± .407	.896 ± .015	.889 ± .012	.879 ± .014	.870 ± .014	.871 ± .012
	time	0.3 ± 0.0	0.4 ± 0.2	1.7 ± 0.8	4.3 ± 1.3	9.3 ± 4.9	18.5 ± 11.9	22.3 ± 11.6
wdbc	acc	.937 ± .029	.942 ± .029	.945 ± .032	.928 ± .041	.930 ± .037	.923 ± .036	.931 ± .046
	nodes	1.0 ± 0.0	1.1 ± 0.3	2.9 ± 0.8	10.0 ± 2.9	18.2 ± 1.6	22.2 ± 2.8	25.7 ± 2.5
	path len	1.00 ± 0.00	1.04 ± 0.13	1.77 ± 0.36	3.38 ± 0.61	4.56 ± 0.60	5.03 ± 0.85	6.05 ± 0.95
	gini	.917 ± .022	.915 ± .022	.925 ± .012	.878 ± .025	.842 ± .023	.815 ± .027	.786 ± .054
	time	0.7 ± 0.0	0.7 ± 0.1	1.1 ± 0.2	2.7 ± 0.7	4.5 ± 0.4	5.4 ± 0.7	6.1 ± 0.6
sonar	acc	.716 ± .102	.735 ± .085	.769 ± .060	.754 ± .080	.759 ± .121	.778 ± .076	.763 ± .074
	nodes	1.0 ± 0.0	3.9 ± 0.7	11.7 ± 1.5	19.2 ± 2.7	18.7 ± 2.1	18.9 ± 2.2	19.5 ± 2.2
	path len	1.00 ± 0.00	2.24 ± 0.20	4.04 ± 0.46	5.23 ± 0.56	5.12 ± 0.63	5.07 ± 0.66	5.08 ± 0.75
	gini	.888 ± .069	.921 ± .035	.890 ± .036	.875 ± .034	.891 ± .020	.888 ± .045	.886 ± .025
	time	0.7 ± 0.0	1.3 ± 0.2	2.9 ± 0.3	4.4 ± 0.6	4.3 ± 0.5	4.3 ± 0.5	4.5 ± 0.5
pendigits	acc	.094 ± .003	.718 ± .041	.790 ± .024	.868 ± .018	.901 ± .012	.921 ± .011	.930 ± .006
	nodes	0.0 ± 0.0	8.4 ± 0.9	15.0 ± 1.6	34.3 ± 2.2	78.0 ± 3.3	180.0 ± 12.1	408.8 ± 21.5
	path len	0.00 ± 0.00	3.81 ± 0.27	4.46 ± 0.26	5.79 ± 0.17	6.77 ± 0.26	8.06 ± 0.30	9.26 ± 0.19
	gini	.000 ± .000	.527 ± .070	.407 ± .104	.317 ± .071	.294 ± .063	.295 ± .047	.288 ± .036
	time	0.4 ± 0.0	5.2 ± 0.5	6.3 ± 0.5	11.6 ± 0.6	22.3 ± 0.8	55.3 ± 3.1	114.9 ± 6.4
ionosphere	acc	.875 ± .048	.903 ± .055	.917 ± .037	.909 ± .058	.906 ± .041	.912 ± .049	.897 ± .050
	nodes	1.0 ± 0.0	1.9 ± 0.3	3.7 ± 1.4	10.5 ± 2.4	17.1 ± 4.1	22.3 ± 3.3	21.3 ± 3.0
	path len	1.00 ± 0.00	1.64 ± 0.22	2.42 ± 0.50	4.20 ± 0.87	6.05 ± 1.09	7.22 ± 1.46	7.18 ± 1.35
	gini	.750 ± .009	.767 ± .040	.769 ± .031	.768 ± .038	.783 ± .042	.774 ± .045	.787 ± .045
	time	0.7 ± 0.0	0.9 ± 0.1	1.2 ± 0.3	2.6 ± 0.5	4.1 ± 0.9	6.8 ± 0.9	6.6 ± 0.9

Ours: linear features, L2 regularization, random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.960 ± .044	.947 ± .065	.960 ± .053	.947 ± .050	.953 ± .043	.960 ± .033	.980 ± .031
	nodes	2.0 ± 0.0	2.8 ± 0.4	2.4 ± 0.5	7.0 ± 1.9	10.6 ± 0.9	10.9 ± 1.5	19.0 ± 2.2
	path len	1.67 ± 0.09	1.96 ± 0.23	1.79 ± 0.20	2.59 ± 0.31	2.91 ± 0.42	2.86 ± 0.28	3.82 ± 0.46
	gini	.627 ± .005	.602 ± .048	.611 ± .018	.681 ± .036	.681 ± .031	.601 ± .045	.633 ± .036
	time	0.6 ± 0.0	0.8 ± 0.1	0.7 ± 0.1	1.8 ± 0.4	2.8 ± 0.2	2.9 ± 0.3	4.7 ± 0.5
heart	acc	.819 ± .048	.802 ± .066	.772 ± .048	.753 ± .078	.756 ± .066	.763 ± .039	.743 ± .063
	nodes	1.0 ± 0.0	1.2 ± 0.4	6.2 ± 0.7	19.7 ± 2.2	22.6 ± 2.5	25.9 ± 2.5	26.1 ± 3.6
	path len	1.00 ± 0.00	1.10 ± 0.21	3.12 ± 0.46	4.76 ± 0.98	5.82 ± 0.53	7.00 ± 0.70	6.12 ± 0.82
	gini	.924 ± .006	.925 ± .004	.920 ± .008	.921 ± .004	.919 ± .003	.919 ± .003	.920 ± .004
	time	0.5 ± 0.0	0.6 ± 0.1	1.9 ± 0.2	5.1 ± 0.5	4.2 ± 0.4	4.7 ± 0.4	4.8 ± 0.6
dry-bean	acc	.668 ± .012	.906 ± .008	.917 ± .006	.918 ± .005	.920 ± .006	.923 ± .007	.924 ± .005
	nodes	3.0 ± 0.0	6.0 ± 0.0	8.0 ± 0.0	10.0 ± 1.3	19.9 ± 1.4	28.4 ± 5.4	66.0 ± 7.9
	path len	2.00 ± 0.00	2.95 ± 0.11	3.38 ± 0.04	3.71 ± 0.29	5.14 ± 0.43	4.88 ± 0.46	6.05 ± 0.37
	gini	.907 ± .005	.905 ± .003	.908 ± .002	.908 ± .003	.908 ± .004	.906 ± .003	.908 ± .002
	time	2.6 ± 0.1	4.3 ± 0.1	5.4 ± 0.1	6.4 ± 0.7	11.3 ± 0.8	23.8 ± 6.3	66.0 ± 9.6
wine	acc	.966 ± .027	.961 ± .026	.960 ± .026	.960 ± .026	.966 ± .028	.966 ± .027	.966 ± .028
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.1 ± 0.3	2.2 ± 0.4	2.1 ± 0.3	2.1 ± 0.3	2.4 ± 0.7
	path len	1.66 ± 0.10	1.66 ± 0.10	1.69 ± 0.14	1.72 ± 0.17	1.69 ± 0.14	1.68 ± 0.14	1.79 ± 0.25
	gini	.872 ± .009	.873 ± .009	.874 ± .007	.876 ± .010	.873 ± .010	.873 ± .011	.876 ± .013
	time	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.1	0.6 ± 0.1	0.6 ± 0.0	0.6 ± 0.1	0.6 ± 0.1
car	acc	.700 ± .044	.910 ± .025	.940 ± .022	.983 ± .011	.993 ± .009	.989 ± .011	.991 ± .006
	nodes	0.0 ± 0.0	2.0 ± 0.0	4.1 ± 0.9	10.1 ± 1.8	17.6 ± 2.2	26.4 ± 4.8	25.8 ± 5.6
	path len	0.00 ± 0.00	1.32 ± 0.04	1.68 ± 0.18	2.47 ± 0.21	3.04 ± 0.50	4.33 ± 0.86	3.99 ± 0.85
	gini	.000 ± .000	.921 ± .001	.914 ± .008	.905 ± .012	.904 ± .007	.915 ± .011	.909 ± .009
	time	0.2 ± 0.0	0.9 ± 0.0	1.5 ± 0.3	3.0 ± 0.5	4.9 ± 0.6	7.4 ± 1.3	7.2 ± 1.5
wdbc	acc	.963 ± .025	.965 ± .024	.967 ± .025	.961 ± .023	.963 ± .025	.961 ± .030	.968 ± .028
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	2.5 ± 1.4	5.2 ± 1.4	13.1 ± 2.8	19.6 ± 4.4
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.54 ± 0.40	2.42 ± 0.38	3.53 ± 0.51	4.51 ± 0.63
	gini	.955 ± .003	.955 ± .002	.955 ± .003	.953 ± .002	.951 ± .003	.951 ± .002	.952 ± .002
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	1.0 ± 0.4	1.7 ± 0.4	3.7 ± 0.8	5.3 ± 1.1
sonar	acc	.736 ± .109	.788 ± .070	.807 ± .058	.807 ± .089	.793 ± .118	.860 ± .034	.822 ± .081
	nodes	1.0 ± 0.0	2.3 ± 0.5	4.8 ± 1.2	5.5 ± 1.0	5.7 ± 1.4	6.4 ± 2.2	6.5 ± 1.9
	path len	1.00 ± 0.00	1.65 ± 0.21	2.65 ± 0.55	2.82 ± 0.54	2.81 ± 0.50	3.14 ± 0.69	3.16 ± 0.69
	gini	.976 ± .001	.975 ± .001	.975 ± .001	.974 ± .001	.974 ± .001	.974 ± .001	.974 ± .001
	time	0.5 ± 0.0	0.9 ± 0.1	1.5 ± 0.3	1.7 ± 0.3	1.8 ± 0.4	1.9 ± 0.5	2.0 ± 0.4
pendigits	acc	.094 ± .003	.906 ± .025	.948 ± .012	.977 ± .007	.983 ± .004	.989 ± .003	.989 ± .003
	nodes	0.0 ± 0.0	9.2 ± 0.6	12.4 ± 1.6	20.1 ± 2.0	31.5 ± 3.2	56.1 ± 6.0	123.5 ± 10.5
	path len	0.00 ± 0.00	3.99 ± 0.12	4.74 ± 0.45	5.01 ± 0.42	5.80 ± 0.29	6.33 ± 0.36	8.10 ± 0.53
	gini	.000 ± .000	.908 ± .001	.900 ± .006	.899 ± .005	.898 ± .004	.900 ± .005	.897 ± .004
	time	0.3 ± 0.0	4.6 ± 0.2	5.9 ± 0.6	8.6 ± 0.7	12.2 ± 1.0	25.6 ± 2.9	47.6 ± 4.0
ionosphere	acc	.843 ± .093	.909 ± .040	.912 ± .039	.923 ± .048	.926 ± .039	.935 ± .040	.926 ± .043
	nodes	1.0 ± 0.0	2.8 ± 0.4	3.6 ± 0.7	4.5 ± 0.7	8.2 ± 1.8	12.4 ± 2.9	13.5 ± 3.6
	path len	1.00 ± 0.00	2.25 ± 0.26	2.75 ± 0.49	3.28 ± 0.33	3.96 ± 0.61	5.22 ± 0.88	5.13 ± 0.85
	gini	.952 ± .004	.934 ± .007	.927 ± .010	.925 ± .004	.933 ± .011	.935 ± .007	.938 ± .005
	time	0.5 ± 0.0	1.0 ± 0.1	1.2 ± 0.2	1.4 ± 0.2	2.3 ± 0.5	3.3 ± 0.7	3.6 ± 1.0

Ours: linear features, L2 regularization, random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.967 ± .045	.953 ± .067	.960 ± .053	.953 ± .052	.953 ± .043	.967 ± .033	.980 ± .031
	nodes	2.0 ± 0.0	2.8 ± 0.4	2.4 ± 0.5	7.0 ± 1.9	10.6 ± 0.9	10.9 ± 1.5	19.0 ± 2.2
	path len	1.67 ± 0.09	1.95 ± 0.23	1.79 ± 0.20	2.62 ± 0.34	2.95 ± 0.50	2.90 ± 0.26	3.82 ± 0.48
	gini	.627 ± .005	.602 ± .048	.611 ± .018	.680 ± .037	.681 ± .030	.599 ± .044	.633 ± .036
	time	0.7 ± 0.0	0.9 ± 0.1	0.8 ± 0.1	1.8 ± 0.4	2.5 ± 0.2	2.5 ± 0.3	4.1 ± 0.4
heart	acc	.815 ± .051	.802 ± .066	.762 ± .046	.753 ± .070	.753 ± .068	.762 ± .053	.746 ± .051
	nodes	1.0 ± 0.0	1.2 ± 0.4	6.2 ± 0.7	19.7 ± 2.2	22.6 ± 2.5	25.9 ± 2.5	26.1 ± 3.6
	path len	1.00 ± 0.00	1.11 ± 0.22	3.13 ± 0.46	4.78 ± 0.99	5.86 ± 0.54	7.10 ± 0.78	6.16 ± 0.79
	gini	.924 ± .006	.925 ± .004	.919 ± .008	.921 ± .004	.919 ± .003	.919 ± .003	.920 ± .003
	time	0.5 ± 0.0	0.6 ± 0.1	1.7 ± 0.2	4.5 ± 0.5	3.7 ± 0.4	4.2 ± 0.4	4.2 ± 0.6
dry-bean	acc	.673 ± .011	.903 ± .009	.914 ± .006	.915 ± .007	.913 ± .008	.920 ± .008	.920 ± .006
	nodes	3.0 ± 0.0	6.0 ± 0.0	8.0 ± 0.0	10.0 ± 1.3	19.9 ± 1.4	28.4 ± 5.4	66.0 ± 7.9
	path len	2.00 ± 0.00	2.95 ± 0.12	3.38 ± 0.04	3.72 ± 0.30	5.17 ± 0.44	4.90 ± 0.48	6.09 ± 0.39
	gini	.907 ± .005	.905 ± .003	.908 ± .002	.908 ± .003	.908 ± .004	.906 ± .003	.908 ± .002
	time	2.5 ± 0.1	4.0 ± 0.1	4.9 ± 0.1	5.8 ± 0.6	9.9 ± 0.7	36.6 ± 13.6	38.4 ± 9.7
wine	acc	.966 ± .027	.961 ± .026	.960 ± .026	.960 ± .026	.966 ± .028	.966 ± .027	.966 ± .028
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.1 ± 0.3	2.2 ± 0.4	2.1 ± 0.3	2.1 ± 0.3	2.4 ± 0.7
	path len	1.66 ± 0.10	1.66 ± 0.10	1.68 ± 0.14	1.72 ± 0.17	1.68 ± 0.14	1.69 ± 0.14	1.79 ± 0.25
	gini	.872 ± .009	.873 ± .009	.874 ± .007	.876 ± .010	.873 ± .010	.873 ± .011	.876 ± .013
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.6 ± 0.1
car	acc	.700 ± .044	.897 ± .026	.937 ± .025	.983 ± .011	.993 ± .009	.989 ± .011	.991 ± .006
	nodes	0.0 ± 0.0	2.0 ± 0.0	4.1 ± 0.9	10.1 ± 1.8	17.6 ± 2.2	26.4 ± 4.8	25.8 ± 5.6
	path len	0.00 ± 0.00	1.32 ± 0.04	1.68 ± 0.18	2.48 ± 0.22	3.05 ± 0.50	4.33 ± 0.85	3.98 ± 0.84
	gini	.000 ± .000	.921 ± .001	.914 ± .008	.905 ± .011	.904 ± .007	.915 ± .011	.909 ± .009
	time	0.2 ± 0.0	0.8 ± 0.0	1.3 ± 0.2	2.7 ± 0.4	4.4 ± 0.5	6.5 ± 1.1	6.3 ± 1.3
wdbc	acc	.961 ± .025	.963 ± .023	.967 ± .025	.961 ± .025	.960 ± .022	.961 ± .025	.963 ± .025
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	2.5 ± 1.4	5.2 ± 1.4	13.1 ± 2.8	19.6 ± 4.4
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.54 ± 0.41	2.42 ± 0.38	3.53 ± 0.53	4.53 ± 0.62
	gini	.955 ± .003	.955 ± .002	.955 ± .003	.953 ± .002	.951 ± .003	.951 ± .002	.952 ± .002
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	0.9 ± 0.3	1.5 ± 0.3	3.2 ± 0.7	4.6 ± 0.9
sonar	acc	.731 ± .111	.783 ± .067	.807 ± .058	.803 ± .093	.789 ± .115	.851 ± .045	.817 ± .077
	nodes	1.0 ± 0.0	2.3 ± 0.5	4.8 ± 1.2	5.5 ± 1.0	5.7 ± 1.4	6.4 ± 2.2	6.5 ± 1.9
	path len	1.00 ± 0.00	1.65 ± 0.22	2.65 ± 0.56	2.79 ± 0.56	2.81 ± 0.50	3.15 ± 0.67	3.16 ± 0.70
	gini	.976 ± .001	.975 ± .001	.975 ± .001	.974 ± .001	.974 ± .001	.974 ± .001	.974 ± .001
	time	0.5 ± 0.0	0.9 ± 0.1	1.4 ± 0.2	1.6 ± 0.2	1.6 ± 0.3	1.7 ± 0.4	1.8 ± 0.4
pendigits	acc	.094 ± .003	.905 ± .025	.946 ± .012	.974 ± .007	.980 ± .004	.987 ± .002	.989 ± .003
	nodes	0.0 ± 0.0	9.2 ± 0.6	12.4 ± 1.6	20.1 ± 2.0	31.5 ± 3.2	56.1 ± 6.0	123.5 ± 10.5
	path len	0.00 ± 0.00	3.99 ± 0.13	4.74 ± 0.45	5.01 ± 0.42	5.80 ± 0.28	6.33 ± 0.36	8.12 ± 0.53
	gini	.000 ± .000	.908 ± .001	.900 ± .006	.899 ± .005	.898 ± .004	.900 ± .005	.898 ± .004
	time	0.3 ± 0.0	4.2 ± 0.2	5.3 ± 0.6	7.7 ± 0.6	10.8 ± 0.9	20.8 ± 2.1	41.1 ± 3.3
ionosphere	acc	.843 ± .093	.915 ± .042	.914 ± .042	.932 ± .053	.926 ± .043	.929 ± .041	.923 ± .051
	nodes	1.0 ± 0.0	2.8 ± 0.4	3.6 ± 0.7	4.5 ± 0.7	8.2 ± 1.8	12.4 ± 2.9	13.5 ± 3.6
	path len	1.00 ± 0.00	2.26 ± 0.26	2.79 ± 0.53	3.35 ± 0.37	4.02 ± 0.63	5.30 ± 0.91	5.22 ± 0.87
	gini	.952 ± .004	.934 ± .007	.928 ± .010	.925 ± .004	.934 ± .011	.935 ± .007	.938 ± .005
	time	0.5 ± 0.0	0.9 ± 0.1	1.1 ± 0.1	1.3 ± 0.1	2.0 ± 0.4	3.0 ± 0.6	3.3 ± 1.0

Ours: prototype features, L2 regularization, random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.947 ± .065	.947 ± .050	.953 ± .043	.953 ± .043	.953 ± .052	.947 ± .040
	nodes	2.0 ± 0.0	2.1 ± 0.3	3.2 ± 1.1	5.6 ± 1.9	10.5 ± 2.9	17.4 ± 3.1	22.1 ± 3.6
	path len	1.67 ± 0.09	1.71 ± 0.17	2.01 ± 0.30	2.52 ± 0.24	2.96 ± 0.49	3.49 ± 0.34	3.97 ± 0.57
	gini	.674 ± .041	.666 ± .018	.670 ± .041	.664 ± .027	.670 ± .032	.670 ± .033	.679 ± .032
	time	1.1 ± 0.0	1.2 ± 0.2	1.5 ± 0.3	2.2 ± 0.6	5.3 ± 1.3	8.4 ± 1.4	7.0 ± 1.1
heart	acc	.809 ± .038	.812 ± .035	.795 ± .041	.769 ± .034	.762 ± .059	.749 ± .061	.743 ± .035
	nodes	1.0 ± 0.0	1.0 ± 0.0	4.5 ± 0.8	13.9 ± 2.0	37.8 ± 3.7	60.1 ± 5.3	85.7 ± 5.1
	path len	1.00 ± 0.00	1.00 ± 0.00	2.40 ± 0.32	4.00 ± 0.43	5.30 ± 0.58	6.49 ± 0.53	7.67 ± 0.53
	gini	.935 ± .003	.937 ± .003	.931 ± .004	.916 ± .005	.904 ± .006	.894 ± .008	.894 ± .008
	time	0.7 ± 0.1	0.7 ± 0.1	1.6 ± 0.2	3.8 ± 0.4	9.3 ± 0.7	14.2 ± 1.2	20.1 ± 1.2
dry-bean	acc	.665 ± .045	.875 ± .037	.905 ± .011	.911 ± .008	.913 ± .006	.913 ± .009	.915 ± .007
	nodes	2.9 ± 0.3	5.6 ± 0.5	7.3 ± 0.8	14.1 ± 1.5	31.1 ± 2.5	56.0 ± 6.0	127.0 ± 9.0
	path len	2.04 ± 0.26	2.77 ± 0.11	3.25 ± 0.26	4.17 ± 0.30	5.23 ± 0.27	5.94 ± 0.33	7.56 ± 0.53
	gini	.881 ± .004	.871 ± .005	.855 ± .017	.784 ± .027	.708 ± .040	.647 ± .041	.559 ± .056
	time	2.9 ± 0.2	4.4 ± 0.3	5.4 ± 0.5	9.1 ± 0.8	17.3 ± 1.2	49.4 ± 7.9	67.0 ± 9.7
wine	acc	.955 ± .033	.978 ± .027	.967 ± .037	.960 ± .036	.961 ± .050	.956 ± .042	.949 ± .039
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.6 ± 0.5	4.9 ± 1.6	6.9 ± 2.4	12.2 ± 4.4	18.6 ± 6.7
	path len	1.66 ± 0.10	1.66 ± 0.09	1.90 ± 0.25	2.51 ± 0.41	2.91 ± 0.47	3.75 ± 0.72	4.83 ± 0.80
	gini	.892 ± .004	.886 ± .005	.875 ± .011	.842 ± .033	.825 ± .032	.802 ± .050	.758 ± .031
	time	0.9 ± 0.0	0.9 ± 0.0	1.1 ± 0.2	1.6 ± 0.4	2.1 ± 0.5	3.3 ± 1.0	4.7 ± 1.5
car	acc	.700 ± .044	.801 ± .060	.910 ± .028	.956 ± .018	.975 ± .016	.985 ± .015	.992 ± .009
	nodes	0.0 ± 0.0	0.7 ± 0.5	4.5 ± 1.7	13.1 ± 4.5	24.4 ± 5.2	39.8 ± 5.9	55.1 ± 8.3
	path len	0.00 ± 0.00	0.70 ± 0.46	2.00 ± 0.30	2.88 ± 0.50	3.36 ± 0.40	3.80 ± 0.63	3.65 ± 0.49
	gini	.000 ± .000	.644 ± .422	.896 ± .011	.875 ± .009	.839 ± .020	.803 ± .034	.775 ± .036
	time	0.3 ± 0.0	0.7 ± 0.2	1.8 ± 0.6	3.9 ± 1.1	6.7 ± 1.3	10.2 ± 1.3	13.6 ± 2.0
wdbc	acc	.967 ± .012	.979 ± .017	.968 ± .013	.968 ± .022	.965 ± .026	.960 ± .029	.958 ± .033
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.8 ± 0.6	4.3 ± 1.0	11.6 ± 2.5	30.4 ± 4.2	52.9 ± 6.4
	path len	1.00 ± 0.00	1.00 ± 0.00	1.42 ± 0.33	2.44 ± 0.36	3.80 ± 0.47	5.09 ± 0.77	6.08 ± 0.48
	gini	.942 ± .002	.942 ± .001	.931 ± .010	.930 ± .011	.902 ± .015	.845 ± .019	.841 ± .031
	time	0.7 ± 0.0	0.7 ± 0.0	1.0 ± 0.2	1.5 ± 0.2	3.3 ± 0.7	8.0 ± 1.1	13.3 ± 1.6
sonar	acc	.534 ± .144	.821 ± .074	.889 ± .058	.826 ± .052	.822 ± .060	.860 ± .094	.812 ± .057
	nodes	0.0 ± 0.0	2.6 ± 0.5	7.8 ± 1.2	12.3 ± 1.7	14.7 ± 2.5	20.4 ± 7.4	30.2 ± 7.4
	path len	0.00 ± 0.00	1.99 ± 0.33	3.69 ± 0.45	4.54 ± 0.41	4.90 ± 0.44	5.69 ± 0.64	6.40 ± 1.05
	gini	.000 ± .000	.972 ± .001	.970 ± .001	.964 ± .003	.952 ± .009	.953 ± .007	.934 ± .012
	time	0.3 ± 0.0	1.2 ± 0.1	2.3 ± 0.3	3.5 ± 0.6	4.0 ± 0.7	5.3 ± 1.7	8.1 ± 2.3
pendigits	acc	.094 ± .003	.855 ± .026	.898 ± .011	.913 ± .014	.935 ± .009	.949 ± .008	.962 ± .005
	nodes	0.0 ± 0.0	8.7 ± 0.5	11.7 ± 1.0	21.2 ± 2.3	55.0 ± 6.0	135.4 ± 8.4	347.3 ± 25.0
	path len	0.00 ± 0.00	4.81 ± 0.38	4.83 ± 0.28	5.05 ± 0.42	6.27 ± 0.29	7.58 ± 0.26	9.45 ± 0.48
	gini	.000 ± .000	.869 ± .011	.846 ± .022	.736 ± .027	.612 ± .046	.533 ± .042	.478 ± .030
	time	0.4 ± 0.0	5.0 ± 0.2	6.2 ± 0.4	9.6 ± 0.9	20.2 ± 1.9	57.6 ± 9.5	135.9 ± 13.0
ionosphere	acc	.914 ± .057	.915 ± .059	.920 ± .046	.926 ± .062	.909 ± .049	.920 ± .033	.917 ± .047
	nodes	1.5 ± 0.5	2.5 ± 0.8	4.2 ± 1.2	8.2 ± 2.6	16.4 ± 1.8	22.2 ± 4.0	22.9 ± 3.4
	path len	1.24 ± 0.25	1.65 ± 0.40	2.25 ± 0.45	3.20 ± 0.54	4.90 ± 0.67	5.58 ± 1.09	5.53 ± 0.88
	gini	.953 ± .003	.950 ± .010	.952 ± .004	.935 ± .015	.872 ± .037	.842 ± .031	.820 ± .018
	time	0.8 ± 0.1	1.0 ± 0.2	1.4 ± 0.3	2.4 ± 0.7	4.4 ± 0.5	9.9 ± 7.4	8.1 ± 1.1

Ours: prototype features, L2 regularization, random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.947 ± .065	.947 ± .050	.947 ± .050	.967 ± .045	.933 ± .052	.953 ± .052
	nodes	2.0 ± 0.0	2.1 ± 0.3	3.2 ± 1.1	5.6 ± 1.9	10.5 ± 2.9	17.4 ± 3.1	22.1 ± 3.6
	path len	1.67 ± 0.09	1.71 ± 0.17	2.00 ± 0.29	2.51 ± 0.25	2.90 ± 0.46	3.49 ± 0.35	3.95 ± 0.57
	gini	.674 ± .041	.666 ± .017	.670 ± .041	.664 ± .027	.670 ± .032	.669 ± .032	.678 ± .032
	time	1.0 ± 0.0	1.0 ± 0.1	1.3 ± 0.3	2.0 ± 0.5	3.2 ± 0.7	5.0 ± 0.8	6.1 ± 1.0
heart	acc	.805 ± .044	.809 ± .040	.782 ± .043	.753 ± .044	.749 ± .067	.733 ± .042	.716 ± .040
	nodes	1.0 ± 0.0	1.0 ± 0.0	4.5 ± 0.8	13.9 ± 2.0	37.8 ± 3.7	60.1 ± 5.3	85.7 ± 5.1
	path len	1.00 ± 0.00	1.00 ± 0.00	2.41 ± 0.32	4.03 ± 0.48	5.23 ± 0.51	6.49 ± 0.62	7.79 ± 0.50
	gini	.935 ± .003	.937 ± .003	.931 ± .004	.916 ± .006	.904 ± .006	.894 ± .008	.894 ± .008
	time	0.7 ± 0.0	0.7 ± 0.0	1.5 ± 0.2	3.4 ± 0.4	8.0 ± 0.7	12.2 ± 1.0	17.1 ± 0.8
dry-bean	acc	.665 ± .039	.872 ± .037	.904 ± .012	.903 ± .008	.907 ± .006	.909 ± .006	.906 ± .010
	nodes	2.9 ± 0.3	5.6 ± 0.5	7.3 ± 0.8	14.1 ± 1.5	31.1 ± 2.5	56.0 ± 6.0	127.0 ± 9.0
	path len	2.04 ± 0.26	2.77 ± 0.12	3.25 ± 0.27	4.18 ± 0.31	5.26 ± 0.29	5.96 ± 0.38	7.70 ± 0.65
	gini	.881 ± .004	.871 ± .005	.855 ± .017	.784 ± .027	.708 ± .040	.647 ± .041	.560 ± .054
	time	2.7 ± 0.2	4.0 ± 0.2	4.9 ± 0.4	7.8 ± 0.6	13.9 ± 0.9	24.5 ± 7.9	43.8 ± 4.6
wine	acc	.955 ± .033	.978 ± .027	.972 ± .028	.960 ± .036	.949 ± .052	.939 ± .058	.939 ± .039
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.6 ± 0.5	4.9 ± 1.6	6.9 ± 2.4	12.2 ± 4.4	18.6 ± 6.7
	path len	1.66 ± 0.11	1.66 ± 0.10	1.90 ± 0.25	2.53 ± 0.41	2.93 ± 0.49	3.74 ± 0.75	4.89 ± 0.88
	gini	.892 ± .004	.886 ± .005	.875 ± .011	.842 ± .033	.825 ± .032	.802 ± .051	.757 ± .031
	time	0.9 ± 0.1	0.8 ± 0.1	1.0 ± 0.1	1.5 ± 0.3	1.9 ± 0.5	2.9 ± 0.8	4.1 ± 1.2
car	acc	.700 ± .044	.765 ± .036	.906 ± .024	.955 ± .018	.975 ± .016	.984 ± .014	.991 ± .009
	nodes	0.0 ± 0.0	0.7 ± 0.5	4.5 ± 1.7	13.1 ± 4.5	24.4 ± 5.2	39.8 ± 5.9	55.1 ± 8.3
	path len	0.00 ± 0.00	0.70 ± 0.46	2.00 ± 0.30	2.89 ± 0.51	3.35 ± 0.42	3.80 ± 0.63	3.65 ± 0.48
	gini	.000 ± .000	.644 ± .422	.896 ± .011	.875 ± .009	.839 ± .020	.803 ± .034	.775 ± .036
	time	0.3 ± 0.0	0.6 ± 0.2	1.6 ± 0.4	3.4 ± 1.0	6.0 ± 1.0	9.0 ± 1.1	11.8 ± 1.7
wdbc	acc	.967 ± .012	.979 ± .017	.968 ± .013	.968 ± .022	.951 ± .022	.954 ± .030	.947 ± .029
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.8 ± 0.6	4.3 ± 1.0	11.6 ± 2.5	30.4 ± 4.2	52.9 ± 6.4
	path len	1.00 ± 0.00	1.00 ± 0.00	1.42 ± 0.33	2.44 ± 0.36	3.82 ± 0.49	5.12 ± 0.79	6.15 ± 0.52
	gini	.942 ± .002	.942 ± .001	.931 ± .010	.930 ± .011	.902 ± .015	.845 ± .019	.841 ± .031
	time	0.7 ± 0.0	0.7 ± 0.0	0.9 ± 0.2	1.4 ± 0.2	2.9 ± 0.5	6.8 ± 0.9	11.1 ± 1.2
sonar	acc	.534 ± .144	.793 ± .073	.865 ± .061	.851 ± .063	.827 ± .069	.860 ± .066	.808 ± .053
	nodes	0.0 ± 0.0	2.6 ± 0.5	7.8 ± 1.2	12.3 ± 1.7	14.7 ± 2.5	20.4 ± 7.4	30.2 ± 7.4
	path len	0.00 ± 0.00	2.01 ± 0.33	3.73 ± 0.49	4.65 ± 0.48	4.93 ± 0.45	5.91 ± 0.84	6.55 ± 1.10
	gini	.000 ± .000	.972 ± .001	.970 ± .001	.964 ± .003	.952 ± .008	.953 ± .007	.934 ± .012
	time	0.3 ± 0.0	1.1 ± 0.1	2.1 ± 0.2	3.1 ± 0.4	3.6 ± 0.5	4.8 ± 1.4	6.6 ± 1.5
pendigits	acc	.094 ± .003	.852 ± .026	.897 ± .012	.907 ± .015	.923 ± .012	.938 ± .008	.951 ± .007
	nodes	0.0 ± 0.0	8.7 ± 0.5	11.7 ± 1.0	21.2 ± 2.3	55.0 ± 6.0	135.4 ± 8.4	347.3 ± 25.0
	path len	0.00 ± 0.00	4.82 ± 0.38	4.84 ± 0.28	5.06 ± 0.43	6.28 ± 0.31	7.57 ± 0.27	9.46 ± 0.50
	gini	.000 ± .000	.869 ± .011	.846 ± .022	.736 ± .027	.612 ± .046	.533 ± .042	.478 ± .030
	time	0.4 ± 0.0	4.5 ± 0.2	5.4 ± 0.3	8.3 ± 0.7	16.9 ± 1.5	43.9 ± 2.6	99.0 ± 6.9
ionosphere	acc	.917 ± .052	.917 ± .059	.917 ± .049	.923 ± .060	.900 ± .059	.906 ± .043	.900 ± .050
	nodes	1.5 ± 0.5	2.5 ± 0.8	4.2 ± 1.2	8.2 ± 2.6	16.4 ± 1.8	22.2 ± 4.0	22.9 ± 3.4
	path len	1.24 ± 0.25	1.65 ± 0.39	2.23 ± 0.47	3.22 ± 0.58	4.88 ± 0.69	5.66 ± 1.17	5.56 ± 0.91
	gini	.953 ± .003	.950 ± .010	.952 ± .004	.935 ± .016	.872 ± .037	.842 ± .030	.819 ± .019
	time	0.8 ± 0.1	1.0 ± 0.1	1.3 ± 0.3	2.2 ± 0.5	3.9 ± 0.4	7.4 ± 2.0	6.8 ± 0.9

Ours: linear features, L1 regularization, random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.960 ± .044	.967 ± .033	.960 ± .053	.960 ± .033	.967 ± .045	.960 ± .044
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.4 ± 1.0	5.9 ± 1.2	10.6 ± 1.7	17.1 ± 1.8	27.1 ± 1.8
	path len	1.67 ± 0.09	1.67 ± 0.09	2.12 ± 0.35	2.55 ± 0.26	3.14 ± 0.40	3.28 ± 0.34	4.32 ± 0.48
	gini	.573 ± .032	.566 ± .018	.547 ± .078	.480 ± .074	.562 ± .049	.481 ± .016	.097 ± .035
	time	0.7 ± 0.1	0.9 ± 0.0	2.4 ± 2.9	22.1 ± 40.2	14.7 ± 35.8	6.5 ± 1.0	36.9 ± 19.5
heart	acc	.819 ± .038	.828 ± .036	.802 ± .046	.789 ± .070	.812 ± .038	.766 ± .064	.786 ± .061
	nodes	1.0 ± 0.0	1.2 ± 0.6	5.3 ± 1.7	19.3 ± 3.8	26.8 ± 5.0	39.7 ± 5.0	57.2 ± 6.8
	path len	1.00 ± 0.00	1.10 ± 0.30	2.72 ± 0.44	4.74 ± 0.55	5.48 ± 0.53	5.57 ± 0.40	6.42 ± 0.58
	gini	.886 ± .017	.864 ± .056	.759 ± .062	.721 ± .056	.710 ± .045	.620 ± .079	.557 ± .064
	time	0.6 ± 0.0	0.6 ± 0.2	1.7 ± 0.4	4.0 ± 0.7	4.1 ± 0.7	5.9 ± 0.7	8.4 ± 1.0
dry-bean	acc	.528 ± .008	.901 ± .009	.913 ± .006	.913 ± .005	.913 ± .008	.919 ± .006	.920 ± .007
	nodes	2.0 ± 0.0	6.0 ± 0.0	7.8 ± 0.4	11.5 ± 1.1	28.0 ± 0.9	58.2 ± 2.9	139.2 ± 15.3
	path len	1.59 ± 0.01	3.04 ± 0.20	3.37 ± 0.09	3.76 ± 0.19	5.41 ± 0.14	6.24 ± 0.26	7.93 ± 0.37
	gini	.753 ± .013	.676 ± .033	.677 ± .040	.540 ± .084	.470 ± .054	.387 ± .070	.398 ± .040
	time	2.4 ± 0.1	5.9 ± 4.5	5.2 ± 0.2	6.8 ± 0.4	13.1 ± 0.4	23.6 ± 0.8	52.3 ± 6.2
wine	acc	.983 ± .025	.977 ± .028	.966 ± .037	.960 ± .036	.966 ± .027	.966 ± .027	.972 ± .028
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.5 ± 0.5	3.1 ± 0.7	4.7 ± 1.1	7.5 ± 1.8	13.2 ± 2.6
	path len	1.67 ± 0.10	1.66 ± 0.10	1.86 ± 0.23	2.10 ± 0.32	2.46 ± 0.19	3.20 ± 0.42	4.11 ± 0.46
	gini	.818 ± .011	.789 ± .011	.814 ± .019	.775 ± .045	.779 ± .032	.756 ± .035	.740 ± .045
	time	0.5 ± 0.0	0.5 ± 0.0	0.5 ± 0.1	0.6 ± 0.1	0.9 ± 0.2	1.3 ± 0.3	2.1 ± 0.4
car	acc	.700 ± .044	.700 ± .044	.949 ± .013	.977 ± .013	.990 ± .011	.991 ± .009	.990 ± .010
	nodes	0.0 ± 0.0	0.0 ± 0.0	5.5 ± 1.0	15.6 ± 4.1	31.3 ± 3.1	36.1 ± 7.3	41.1 ± 9.8
	path len	0.00 ± 0.00	0.00 ± 0.00	2.13 ± 0.25	2.98 ± 0.42	3.41 ± 0.46	3.50 ± 0.50	3.61 ± 0.69
	gini	.000 ± .000	.000 ± .000	.720 ± .019	.703 ± .045	.703 ± .032	.655 ± .033	.663 ± .031
	time	0.3 ± 0.0	0.3 ± 0.0	2.0 ± 0.3	4.7 ± 1.1	8.8 ± 0.9	9.9 ± 2.0	11.3 ± 2.8
wdbc	acc	.970 ± .019	.970 ± .019	.974 ± .018	.968 ± .017	.979 ± .020	.968 ± .022	.979 ± .015
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	3.2 ± 0.6	7.0 ± 3.0	17.4 ± 4.1	34.9 ± 8.9
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	2.01 ± 0.02	2.57 ± 0.49	3.93 ± 0.41	4.73 ± 0.60
	gini	.817 ± .019	.819 ± .022	.816 ± .019	.835 ± .045	.936 ± .008	.936 ± .006	.936 ± .007
	time	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	1.2 ± 0.2	2.3 ± 0.8	4.9 ± 1.1	9.2 ± 2.3
sonar	acc	.655 ± .169	.841 ± .074	.812 ± .070	.807 ± .078	.817 ± .074	.812 ± .084	.846 ± .076
	nodes	0.9 ± 0.3	2.6 ± 0.7	5.6 ± 0.9	7.0 ± 1.6	12.6 ± 4.8	21.7 ± 3.7	28.9 ± 7.4
	path len	0.90 ± 0.30	1.86 ± 0.28	2.73 ± 0.35	2.98 ± 0.29	3.95 ± 0.65	5.00 ± 0.84	5.67 ± 0.82
	gini	.849 ± .283	.939 ± .014	.899 ± .029	.899 ± .019	.915 ± .020	.933 ± .022	.923 ± .033
	time	0.8 ± 0.1	1.4 ± 0.2	2.6 ± 0.3	3.1 ± 0.6	5.2 ± 1.8	8.4 ± 1.4	8.6 ± 3.1
pendigits	acc	.094 ± .003	.901 ± .008	.940 ± .011	.968 ± .002	.976 ± .004	.977 ± .005	.981 ± .003
	nodes	0.0 ± 0.0	9.0 ± 0.0	11.9 ± 1.6	17.9 ± 1.6	42.2 ± 4.0	120.1 ± 7.1	298.2 ± 9.1
	path len	0.00 ± 0.00	3.96 ± 0.04	4.55 ± 0.45	4.84 ± 0.26	5.91 ± 0.30	7.40 ± 0.18	8.89 ± 0.32
	gini	.000 ± .000	.809 ± .010	.813 ± .024	.764 ± .022	.696 ± .041	.492 ± .046	.320 ± .027
	time	9.5 ± 8.9	7.6 ± 3.3	139.3 ± 83.5	109.5 ± 27.3	202.4 ± 82.2	392.0 ± 106.4	1032.4 ± 182.0
ionosphere	acc	.855 ± .079	.909 ± .062	.926 ± .038	.923 ± .054	.926 ± .050	.920 ± .064	.940 ± .035
	nodes	1.0 ± 0.0	2.4 ± 0.5	3.4 ± 0.5	6.3 ± 1.6	15.7 ± 4.0	32.2 ± 4.6	52.3 ± 5.4
	path len	1.00 ± 0.00	2.02 ± 0.33	2.73 ± 0.41	3.83 ± 0.55	5.60 ± 1.09	8.38 ± 0.77	11.72 ± 0.88
	gini	.910 ± .010	.828 ± .094	.699 ± .090	.663 ± .067	.712 ± .063	.785 ± .048	.853 ± .028
	time	0.6 ± 0.0	1.0 ± 0.1	1.3 ± 0.1	2.0 ± 0.4	2.8 ± 0.8	5.1 ± 0.7	10.4 ± 2.9

Ours: linear features, L1 regularization, random initialization, crisp

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.953 ± .043	.960 ± .044	.960 ± .033	.953 ± .052	.960 ± .044	.960 ± .044	.967 ± .045
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.4 ± 1.0	5.9 ± 1.2	10.6 ± 1.7	17.1 ± 1.8	27.1 ± 1.8
	path len	1.67 ± 0.09	1.67 ± 0.09	2.12 ± 0.35	2.53 ± 0.26	3.14 ± 0.44	3.25 ± 0.35	4.45 ± 0.58
	gini	.573 ± .032	.566 ± .018	.547 ± .078	.469 ± .082	.563 ± .049	.480 ± .015	.099 ± .038
	time	0.4 ± 0.0	0.4 ± 0.0	0.6 ± 0.1	0.9 ± 0.1	1.5 ± 0.2	2.2 ± 0.2	3.4 ± 0.2
heart	acc	.819 ± .039	.828 ± .036	.786 ± .051	.780 ± .066	.812 ± .037	.756 ± .071	.786 ± .064
	nodes	1.0 ± 0.0	1.2 ± 0.6	5.3 ± 1.7	19.3 ± 3.8	26.8 ± 5.0	39.7 ± 5.0	57.2 ± 6.8
	path len	1.00 ± 0.00	1.10 ± 0.30	2.72 ± 0.44	4.72 ± 0.53	5.49 ± 0.53	5.56 ± 0.42	6.42 ± 0.66
	gini	.886 ± .017	.864 ± .056	.760 ± .063	.720 ± .056	.709 ± .046	.618 ± .079	.559 ± .065
	time	0.4 ± 0.0	0.5 ± 0.1	1.2 ± 0.3	3.6 ± 0.7	3.6 ± 0.6	5.1 ± 0.6	7.1 ± 0.8
dry-bean	acc	.530 ± .007	.892 ± .012	.909 ± .007	.905 ± .008	.907 ± .007	.913 ± .007	.910 ± .005
	nodes	2.0 ± 0.0	6.0 ± 0.0	7.8 ± 0.4	11.5 ± 1.1	28.0 ± 0.9	58.2 ± 2.9	139.2 ± 15.3
	path len	1.60 ± 0.01	3.05 ± 0.23	3.36 ± 0.08	3.77 ± 0.19	5.46 ± 0.15	6.27 ± 0.26	8.13 ± 0.46
	gini	.753 ± .013	.676 ± .033	.677 ± .040	.540 ± .083	.471 ± .054	.387 ± .070	.398 ± .040
	time	2.3 ± 0.0	4.1 ± 0.1	4.8 ± 0.2	5.9 ± 0.3	10.5 ± 0.3	18.0 ± 0.7	37.1 ± 3.7
wine	acc	.983 ± .025	.977 ± .028	.966 ± .037	.960 ± .036	.967 ± .027	.967 ± .027	.966 ± .027
	nodes	2.0 ± 0.0	2.0 ± 0.0	2.5 ± 0.5	3.1 ± 0.7	4.7 ± 1.1	7.5 ± 1.8	13.2 ± 2.6
	path len	1.67 ± 0.09	1.66 ± 0.09	1.85 ± 0.23	2.10 ± 0.31	2.45 ± 0.20	3.19 ± 0.44	4.12 ± 0.48
	gini	.817 ± .011	.789 ± .011	.814 ± .019	.775 ± .045	.779 ± .032	.756 ± .035	.740 ± .045
	time	0.4 ± 0.0	0.4 ± 0.0	0.5 ± 0.1	0.6 ± 0.1	0.8 ± 0.1	1.1 ± 0.2	1.8 ± 0.3
car	acc	.700 ± .044	.700 ± .044	.949 ± .013	.978 ± .013	.990 ± .011	.992 ± .009	.990 ± .010
	nodes	0.0 ± 0.0	0.0 ± 0.0	5.5 ± 1.0	15.6 ± 4.1	31.3 ± 3.1	36.1 ± 7.3	41.1 ± 9.8
	path len	0.00 ± 0.00	0.00 ± 0.00	2.13 ± 0.25	2.98 ± 0.42	3.41 ± 0.46	3.50 ± 0.50	3.62 ± 0.69
	gini	.000 ± .000	.000 ± .000	.720 ± .020	.703 ± .045	.703 ± .032	.655 ± .033	.663 ± .031
	time	0.3 ± 0.0	0.3 ± 0.0	1.9 ± 0.3	4.2 ± 1.0	7.8 ± 0.8	8.8 ± 1.7	9.9 ± 2.2
wdbc	acc	.970 ± .019	.970 ± .019	.974 ± .018	.968 ± .017	.977 ± .016	.965 ± .022	.970 ± .025
	nodes	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	3.2 ± 0.6	7.0 ± 3.0	17.4 ± 4.1	34.9 ± 8.9
	path len	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	2.00 ± 0.01	2.57 ± 0.49	3.94 ± 0.41	4.72 ± 0.61
	gini	.817 ± .019	.819 ± .022	.816 ± .019	.835 ± .045	.936 ± .008	.936 ± .006	.936 ± .007
	time	0.6 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	1.1 ± 0.1	2.0 ± 0.7	4.3 ± 0.9	7.8 ± 1.8
sonar	acc	.665 ± .174	.841 ± .071	.807 ± .081	.797 ± .079	.827 ± .065	.803 ± .084	.832 ± .071
	nodes	0.9 ± 0.3	2.6 ± 0.7	5.6 ± 0.9	7.0 ± 1.6	12.6 ± 4.8	21.7 ± 3.7	28.9 ± 7.4
	path len	0.90 ± 0.30	1.86 ± 0.28	2.70 ± 0.36	3.00 ± 0.26	3.97 ± 0.66	4.96 ± 0.83	5.68 ± 0.81
	gini	.849 ± .283	.939 ± .013	.899 ± .029	.900 ± .018	.915 ± .020	.933 ± .022	.924 ± .032
	time	0.5 ± 0.1	1.0 ± 0.2	1.6 ± 0.2	2.0 ± 0.3	3.2 ± 1.0	5.2 ± 0.8	3.9 ± 0.9
pendigits	acc	.094 ± .003	.899 ± .009	.936 ± .009	.963 ± .004	.970 ± .006	.970 ± .004	.975 ± .005
	nodes	0.0 ± 0.0	9.0 ± 0.0	11.9 ± 1.6	17.9 ± 1.6	42.2 ± 4.0	120.1 ± 7.1	292.1 ± 10.3
	path len	0.00 ± 0.00	3.97 ± 0.03	4.56 ± 0.45	4.84 ± 0.26	5.90 ± 0.31	7.38 ± 0.19	8.78 ± 0.33
	gini	.000 ± .000	.809 ± .010	.813 ± .024	.764 ± .022	.695 ± .041	.492 ± .045	.321 ± .027
	time	11.3 ± 5.0	72.8 ± 33.8	83.9 ± 14.0	112.8 ± 51.7	220.7 ± 82.2	31.2 ± 1.6	84.9 ± 8.8
ionosphere	acc	.857 ± .077	.909 ± .060	.923 ± .038	.917 ± .055	.926 ± .045	.900 ± .058	.915 ± .044
	nodes	1.0 ± 0.0	2.4 ± 0.5	3.4 ± 0.5	6.3 ± 1.6	15.7 ± 4.0	32.2 ± 4.6	52.3 ± 5.4
	path len	1.00 ± 0.00	2.02 ± 0.34	2.77 ± 0.39	3.89 ± 0.55	5.63 ± 1.16	8.56 ± 0.85	12.33 ± 0.88
	gini	.910 ± .010	.828 ± .093	.700 ± .088	.665 ± .067	.711 ± .062	.786 ± .047	.855 ± .027
	time	0.5 ± 0.0	0.9 ± 0.1	1.2 ± 0.1	1.8 ± 0.4	2.4 ± 0.7	4.3 ± 0.6	8.7 ± 2.8

Ours: prototype features, L1 regularization, random initialization, fuzzy

data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.947 ± .065	.940 ± .047	.947 ± .050	.967 ± .045	.947 ± .072	.933 ± .060	.947 ± .050
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.1 ± 0.7	6.2 ± 1.1	9.5 ± 2.2	18.3 ± 4.4	26.3 ± 7.3
	path len	1.67 ± 0.09	1.67 ± 0.09	2.02 ± 0.22	2.58 ± 0.32	2.92 ± 0.54	3.95 ± 0.63	4.40 ± 0.69
	gini	.616 ± .043	.646 ± .040	.604 ± .059	.575 ± .061	.577 ± .094	.538 ± .054	.580 ± .045
	time	1.1 ± 0.0	1.1 ± 0.0	1.5 ± 0.2	2.4 ± 0.3	2.7 ± 0.5	6.0 ± 1.3	8.4 ± 2.2
heart	acc	.812 ± .057	.805 ± .048	.775 ± .049	.756 ± .054	.766 ± .081	.756 ± .052	.798 ± .036
	nodes	1.0 ± 0.0	1.2 ± 0.4	3.9 ± 1.1	13.5 ± 1.7	45.9 ± 5.0	71.1 ± 6.8	97.1 ± 9.0
	path len	1.00 ± 0.00	1.08 ± 0.16	2.19 ± 0.34	3.85 ± 0.45	5.48 ± 0.48	6.99 ± 0.75	7.68 ± 0.84
	gini	.936 ± .003	.935 ± .002	.929 ± .005	.900 ± .012	.852 ± .022	.839 ± .018	.828 ± .031
	time	0.9 ± 0.0	1.0 ± 0.1	1.8 ± 0.3	4.8 ± 0.6	14.7 ± 1.5	22.3 ± 2.1	22.9 ± 2.3
dry-bean	acc	.617 ± .068	.869 ± .035	.894 ± .011	.902 ± .006	.905 ± .006	.909 ± .005	.915 ± .006
	nodes	2.6 ± 0.5	5.6 ± 0.5	7.5 ± 1.0	16.1 ± 1.9	37.1 ± 2.8	79.7 ± 7.0	190.1 ± 15.1
	path len	1.92 ± 0.31	2.83 ± 0.17	3.15 ± 0.29	4.38 ± 0.19	5.59 ± 0.31	6.95 ± 0.34	8.86 ± 0.71
	gini	.724 ± .052	.718 ± .027	.596 ± .078	.385 ± .058	.356 ± .046	.310 ± .046	.303 ± .042
	time	2.7 ± 0.4	4.5 ± 0.2	5.8 ± 0.8	10.2 ± 0.9	20.6 ± 1.3	38.1 ± 3.7	86.0 ± 9.2
wine	acc	.955 ± .043	.960 ± .036	.955 ± .042	.950 ± .058	.955 ± .042	.939 ± .046	.939 ± .052
	nodes	2.0 ± 0.0	2.2 ± 0.4	3.1 ± 0.5	8.3 ± 2.2	14.1 ± 3.2	25.2 ± 6.4	35.1 ± 9.5
	path len	1.66 ± 0.10	1.74 ± 0.20	2.01 ± 0.25	3.26 ± 0.49	4.17 ± 0.49	5.18 ± 0.69	5.70 ± 0.72
	gini	.845 ± .012	.847 ± .020	.815 ± .026	.614 ± .111	.529 ± .085	.418 ± .094	.403 ± .070
	time	0.9 ± 0.0	1.0 ± 0.1	1.2 ± 0.1	2.4 ± 0.5	3.7 ± 0.7	6.2 ± 1.4	8.4 ± 2.1
car	acc	.700 ± .044	.770 ± .052	.833 ± .046	.953 ± .017	.969 ± .014	.974 ± .015	.983 ± .009
	nodes	0.0 ± 0.0	1.1 ± 0.7	2.8 ± 1.2	12.9 ± 2.9	26.5 ± 6.8	49.8 ± 12.4	70.4 ± 13.1
	path len	0.00 ± 0.00	0.97 ± 0.39	1.61 ± 0.38	2.86 ± 0.40	3.38 ± 0.54	3.70 ± 0.59	4.00 ± 0.49
	gini	.000 ± .000	.699 ± .236	.816 ± .040	.769 ± .035	.707 ± .056	.658 ± .035	.596 ± .038
	time	0.3 ± 0.0	0.8 ± 0.2	1.3 ± 0.3	3.9 ± 0.7	7.1 ± 1.5	12.2 ± 2.8	16.7 ± 2.9
wdbc	acc	.961 ± .025	.956 ± .024	.961 ± .027	.970 ± .019	.963 ± .024	.961 ± .022	.953 ± .024
	nodes	1.0 ± 0.0	1.1 ± 0.3	1.8 ± 0.7	6.3 ± 1.3	19.1 ± 1.8	37.4 ± 3.1	68.8 ± 4.4
	path len	1.00 ± 0.00	1.07 ± 0.22	1.36 ± 0.37	2.69 ± 0.28	3.90 ± 0.41	4.75 ± 0.32	6.08 ± 0.48
	gini	.864 ± .015	.851 ± .039	.821 ± .038	.779 ± .049	.646 ± .062	.569 ± .064	.633 ± .045
	time	0.7 ± 0.0	0.8 ± 0.1	0.9 ± 0.2	2.0 ± 0.3	5.0 ± 0.4	9.3 ± 0.8	16.7 ± 1.0
sonar	acc	.534 ± .144	.759 ± .091	.798 ± .072	.789 ± .052	.774 ± .046	.779 ± .081	.812 ± .058
	nodes	0.0 ± 0.0	2.0 ± 0.4	7.6 ± 1.4	18.7 ± 2.6	29.5 ± 4.3	39.3 ± 3.3	50.4 ± 6.3
	path len	0.00 ± 0.00	1.64 ± 0.28	3.38 ± 0.29	4.70 ± 0.36	5.30 ± 0.50	5.77 ± 0.44	6.24 ± 0.67
	gini	.000 ± .000	.931 ± .014	.905 ± .020	.844 ± .024	.770 ± .029	.743 ± .031	.731 ± .037
	time	0.3 ± 0.0	1.0 ± 0.1	2.4 ± 0.5	4.9 ± 0.6	7.3 ± 1.0	9.6 ± 0.8	12.0 ± 1.5
pendigits	acc	.094 ± .003	.783 ± .057	.854 ± .024	.881 ± .019	.912 ± .012	.939 ± .009	.949 ± .011
	nodes	0.0 ± 0.0	8.2 ± 0.7	12.8 ± 1.2	30.2 ± 3.7	69.7 ± 10.1	166.7 ± 10.7	372.4 ± 23.3
	path len	0.00 ± 0.00	4.10 ± 0.47	4.56 ± 0.56	5.28 ± 0.34	6.58 ± 0.27	7.93 ± 0.34	9.49 ± 0.18
	gini	.000 ± .000	.770 ± .029	.684 ± .043	.475 ± .090	.317 ± .030	.353 ± .045	.354 ± .031
	time	0.4 ± 0.0	4.7 ± 0.3	6.6 ± 0.5	12.2 ± 1.2	24.2 ± 2.8	62.9 ± 4.2	179.6 ± 18.3
ionosphere	acc	.903 ± .045	.892 ± .057	.920 ± .045	.909 ± .035	.897 ± .092	.917 ± .052	.906 ± .054
	nodes	1.1 ± 0.3	1.8 ± 0.6	3.8 ± 1.1	10.5 ± 2.3	23.3 ± 4.2	40.5 ± 7.4	58.8 ± 6.4
	path len	1.03 ± 0.10	1.42 ± 0.34	2.29 ± 0.46	4.35 ± 0.66	5.96 ± 0.88	7.48 ± 0.88	9.28 ± 1.87
	gini	.894 ± .012	.887 ± .029	.839 ± .033	.706 ± .022	.616 ± .050	.596 ± .051	.594 ± .058
	time	0.7 ± 0.1	0.9 ± 0.2	1.4 ± 0.3	2.9 ± 0.6	6.1 ± 0.9	13.2 ± 2.3	18.9 ± 1.9

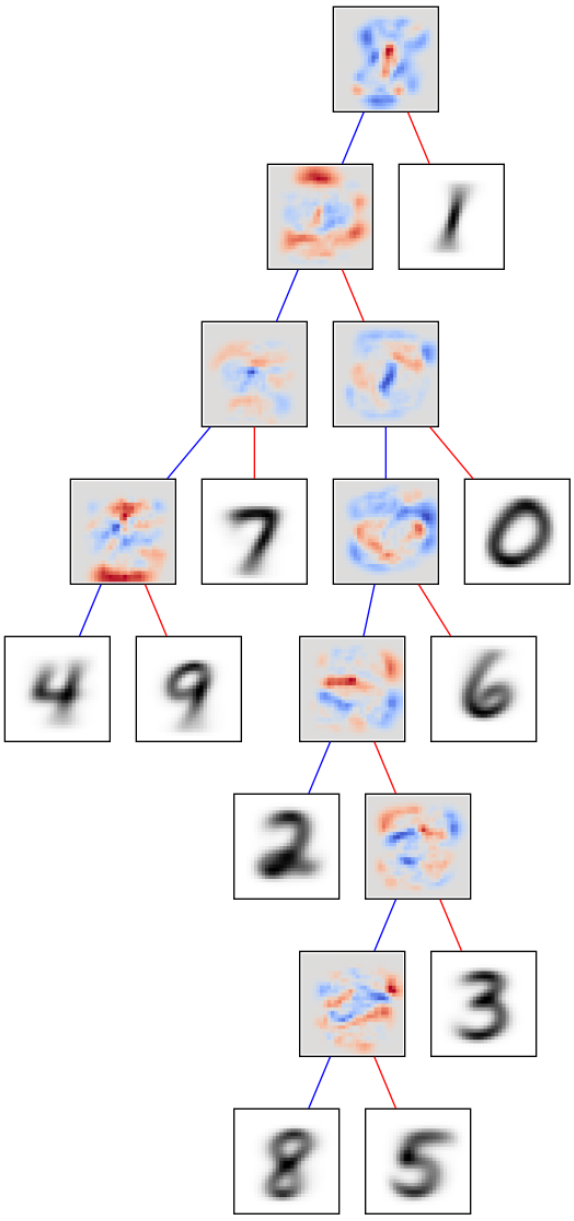
Ours: prototype features, L1 regularization, random initialization, crisp

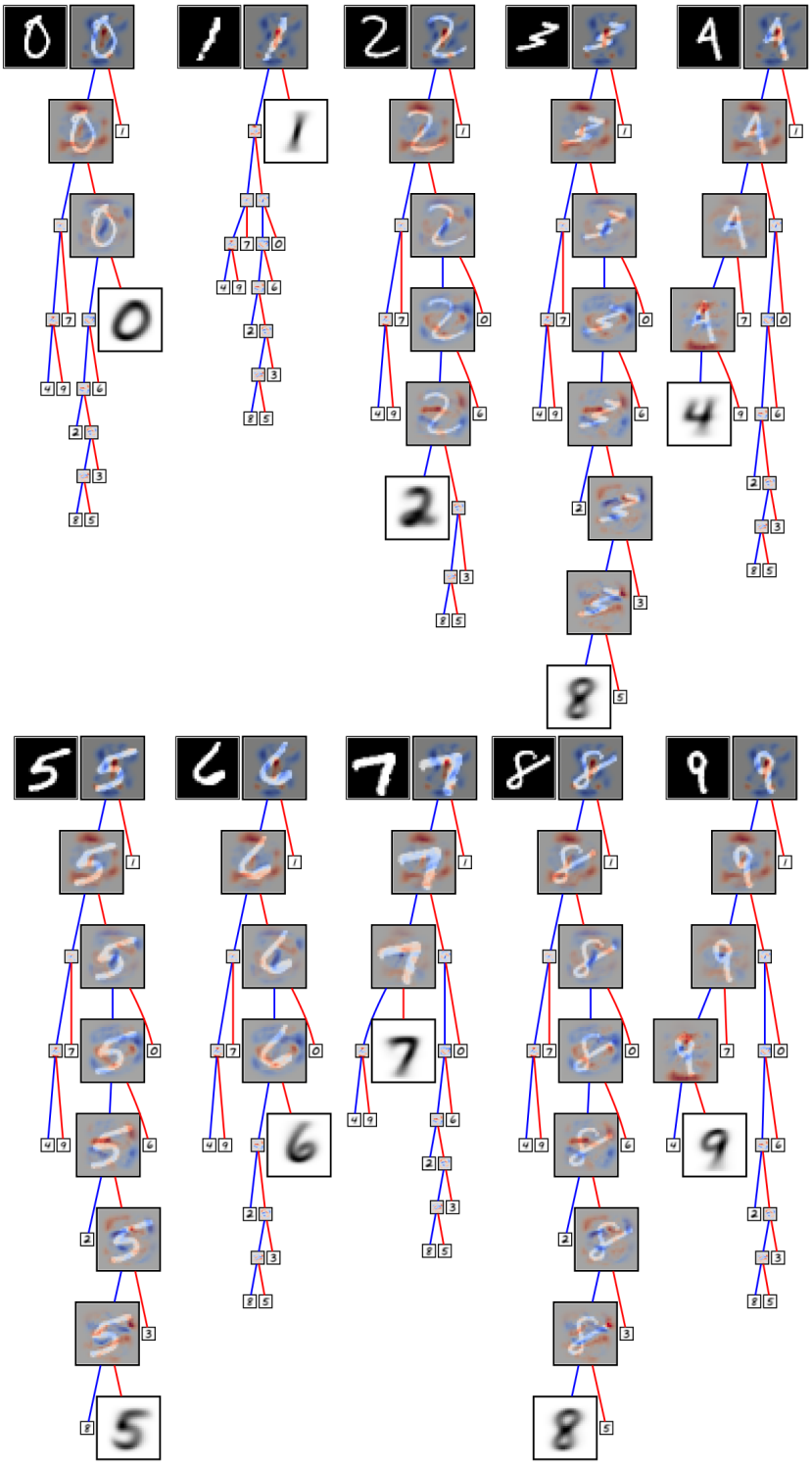
data	metric	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4	1e-4
iris	acc	.947 ± .065	.940 ± .047	.947 ± .050	.947 ± .050	.947 ± .058	.927 ± .063	.947 ± .050
	nodes	2.0 ± 0.0	2.0 ± 0.0	3.1 ± 0.7	6.2 ± 1.1	9.5 ± 2.2	18.3 ± 4.4	26.3 ± 7.3
	path len	1.67 ± 0.09	1.67 ± 0.09	2.03 ± 0.21	2.56 ± 0.33	2.89 ± 0.53	4.00 ± 0.61	4.43 ± 0.71
	gini	.616 ± .043	.646 ± .040	.604 ± .058	.574 ± .061	.575 ± .095	.538 ± .056	.578 ± .048
	time	1.0 ± 0.0	1.0 ± 0.0	1.4 ± 0.2	1.7 ± 0.2	2.4 ± 0.5	5.3 ± 1.1	7.4 ± 1.9
heart	acc	.809 ± .055	.802 ± .036	.772 ± .061	.743 ± .066	.737 ± .082	.750 ± .061	.792 ± .041
	nodes	1.0 ± 0.0	1.2 ± 0.4	3.9 ± 1.1	13.5 ± 1.7	45.9 ± 5.0	71.1 ± 6.8	97.1 ± 9.0
	path len	1.00 ± 0.00	1.09 ± 0.17	2.19 ± 0.37	3.84 ± 0.48	5.47 ± 0.53	7.09 ± 0.80	7.81 ± 0.98
	gini	.936 ± .003	.935 ± .002	.929 ± .005	.900 ± .012	.852 ± .022	.840 ± .020	.828 ± .031
	time	0.7 ± 0.0	0.7 ± 0.1	1.3 ± 0.2	3.3 ± 0.4	9.4 ± 0.9	14.1 ± 1.3	19.1 ± 1.8
dry-bean	acc	.612 ± .065	.861 ± .035	.887 ± .011	.890 ± .007	.892 ± .007	.894 ± .006	.893 ± .017
	nodes	2.6 ± 0.5	5.6 ± 0.5	7.5 ± 1.0	16.1 ± 1.9	37.1 ± 2.8	79.7 ± 7.0	190.1 ± 15.1
	path len	1.93 ± 0.31	2.84 ± 0.18	3.14 ± 0.29	4.39 ± 0.20	5.65 ± 0.33	7.07 ± 0.40	9.45 ± 1.11
	gini	.724 ± .052	.718 ± .026	.596 ± .078	.383 ± .059	.357 ± .046	.313 ± .045	.308 ± .045
	time	2.5 ± 0.3	4.1 ± 0.2	5.0 ± 0.5	8.6 ± 0.7	15.8 ± 1.0	27.2 ± 2.6	56.9 ± 4.7
wine	acc	.949 ± .040	.960 ± .036	.939 ± .058	.944 ± .066	.933 ± .054	.922 ± .067	.916 ± .037
	nodes	2.0 ± 0.0	2.2 ± 0.4	3.1 ± 0.5	8.3 ± 2.2	14.1 ± 3.2	25.2 ± 6.4	35.1 ± 9.5
	path len	1.67 ± 0.11	1.74 ± 0.20	2.02 ± 0.26	3.26 ± 0.52	4.22 ± 0.55	5.25 ± 0.76	5.76 ± 0.75
	gini	.845 ± .012	.847 ± .020	.815 ± .026	.612 ± .110	.529 ± .085	.420 ± .094	.403 ± .071
	time	0.8 ± 0.1	0.9 ± 0.1	1.1 ± 0.1	2.2 ± 0.4	3.3 ± 0.6	5.4 ± 1.2	7.2 ± 1.8
car	acc	.700 ± .044	.771 ± .053	.832 ± .044	.953 ± .017	.969 ± .013	.974 ± .015	.982 ± .009
	nodes	0.0 ± 0.0	1.1 ± 0.7	2.8 ± 1.2	12.9 ± 2.9	26.5 ± 6.8	49.8 ± 12.4	70.4 ± 13.1
	path len	0.00 ± 0.00	0.97 ± 0.39	1.61 ± 0.39	2.85 ± 0.39	3.38 ± 0.54	3.70 ± 0.59	4.01 ± 0.49
	gini	.000 ± .000	.700 ± .236	.816 ± .040	.769 ± .035	.707 ± .056	.658 ± .035	.596 ± .038
	time	0.3 ± 0.0	0.8 ± 0.2	1.2 ± 0.3	3.4 ± 0.6	6.3 ± 1.3	10.9 ± 2.2	14.8 ± 2.5
wdbc	acc	.961 ± .025	.956 ± .024	.961 ± .027	.958 ± .026	.947 ± .025	.951 ± .020	.937 ± .037
	nodes	1.0 ± 0.0	1.1 ± 0.3	1.8 ± 0.7	6.3 ± 1.3	19.1 ± 1.8	37.4 ± 3.1	68.8 ± 4.4
	path len	1.00 ± 0.00	1.07 ± 0.22	1.36 ± 0.37	2.69 ± 0.30	3.88 ± 0.40	4.71 ± 0.35	6.13 ± 0.61
	gini	.864 ± .015	.851 ± .039	.821 ± .038	.780 ± .050	.645 ± .062	.568 ± .063	.632 ± .044
	time	0.7 ± 0.0	0.7 ± 0.1	0.9 ± 0.2	1.8 ± 0.3	4.4 ± 0.4	7.9 ± 0.7	13.8 ± 0.8
sonar	acc	.534 ± .144	.769 ± .115	.813 ± .062	.798 ± .046	.778 ± .057	.788 ± .066	.803 ± .072
	nodes	0.0 ± 0.0	2.0 ± 0.4	7.6 ± 1.4	18.7 ± 2.6	29.5 ± 4.3	39.3 ± 3.3	50.4 ± 6.3
	path len	0.00 ± 0.00	1.65 ± 0.27	3.44 ± 0.30	4.73 ± 0.41	5.34 ± 0.54	5.85 ± 0.48	6.31 ± 0.71
	gini	.000 ± .000	.931 ± .014	.906 ± .020	.844 ± .023	.770 ± .028	.744 ± .032	.730 ± .035
	time	0.3 ± 0.0	0.9 ± 0.1	2.1 ± 0.3	4.3 ± 0.5	6.4 ± 0.8	8.5 ± 1.1	10.5 ± 1.6
pendigits	acc	.094 ± .003	.773 ± .047	.844 ± .024	.865 ± .015	.892 ± .013	.919 ± .010	.930 ± .014
	nodes	0.0 ± 0.0	8.3 ± 0.6	12.8 ± 1.2	30.2 ± 3.7	69.7 ± 10.1	166.7 ± 10.7	372.4 ± 23.3
	path len	0.00 ± 0.00	3.93 ± 0.45	4.57 ± 0.58	5.28 ± 0.34	6.58 ± 0.27	7.93 ± 0.37	9.51 ± 0.18
	gini	.000 ± .000	.743 ± .034	.684 ± .043	.475 ± .090	.317 ± .030	.352 ± .045	.354 ± .031
	time	0.4 ± 0.0	4.2 ± 0.3	5.7 ± 0.5	10.4 ± 1.0	20.2 ± 2.4	51.1 ± 3.4	152.9 ± 8.7
ionosphere	acc	.903 ± .045	.892 ± .057	.923 ± .048	.906 ± .043	.909 ± .079	.900 ± .057	.883 ± .079
	nodes	1.1 ± 0.3	1.8 ± 0.6	3.8 ± 1.1	10.5 ± 2.3	23.3 ± 4.2	40.5 ± 7.4	58.8 ± 6.4
	path len	1.03 ± 0.10	1.42 ± 0.35	2.29 ± 0.48	4.38 ± 0.68	6.01 ± 0.88	7.67 ± 0.88	10.00 ± 2.49
	gini	.894 ± .012	.887 ± .029	.839 ± .032	.705 ± .022	.615 ± .051	.596 ± .051	.594 ± .060
	time	0.7 ± 0.1	0.8 ± 0.1	1.2 ± 0.2	2.6 ± 0.5	5.4 ± 0.9	11.4 ± 1.9	16.2 ± 1.6

D.2.2 Visualization of MNIST and Fashion-MNIST Trees

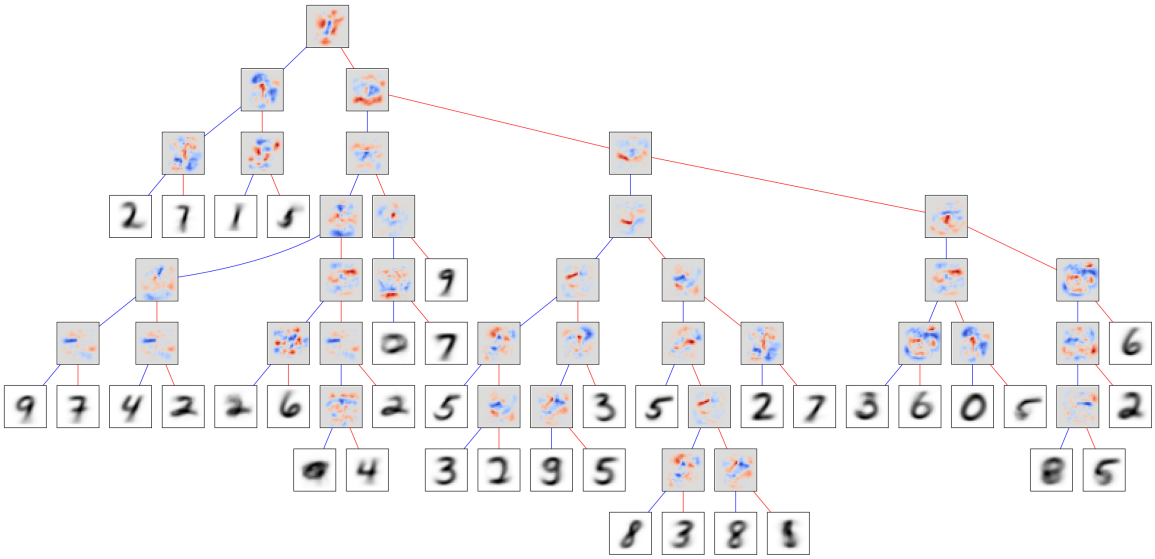
Here we show the full decision trees trained on MNIST and Fashion-MNIST, which were covered in part in Section 4.3.3. For each tree, we also show one randomly selected prediction from the training set for each class. The trees with $\alpha = 10^{-5}$ are not shown because of their large size. While the larger trees may be too small to be legible in print, the image resolutions are high, so zooming in to a digital view of this document will make the details visible.

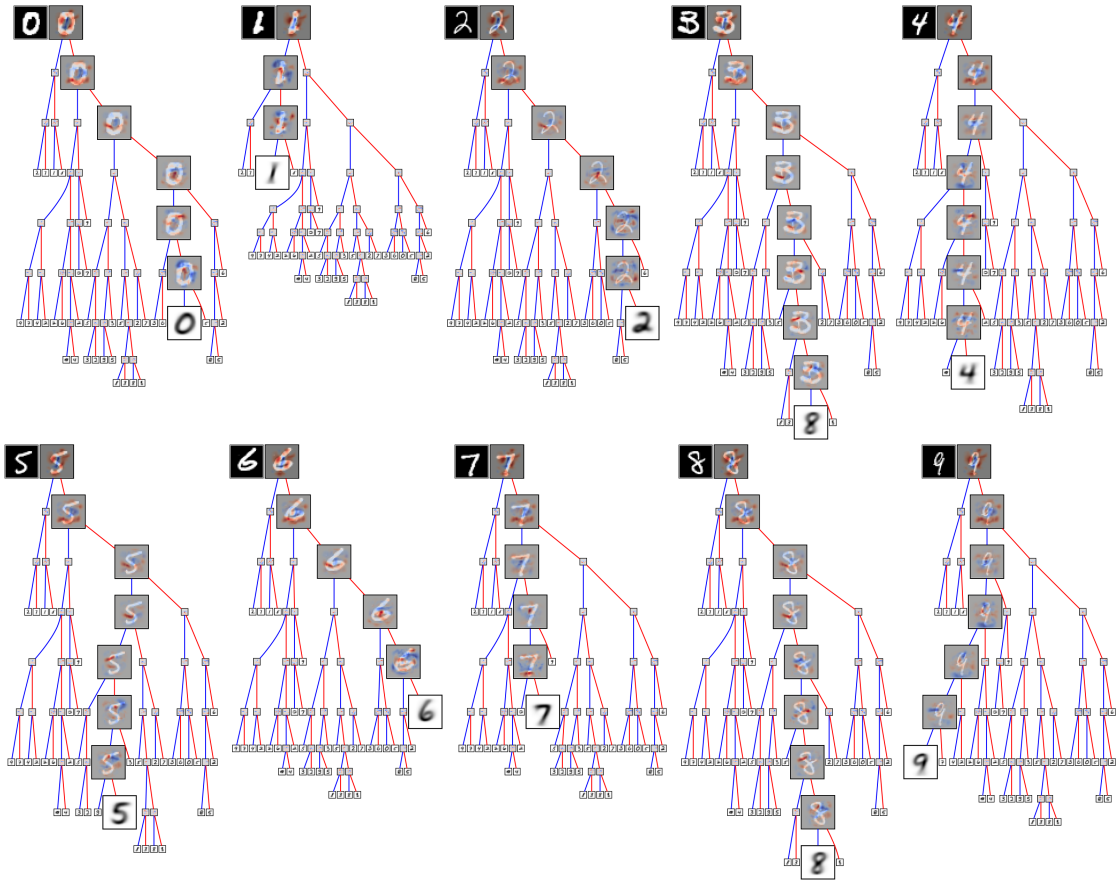
MNIST $\alpha = 10^{-2}$



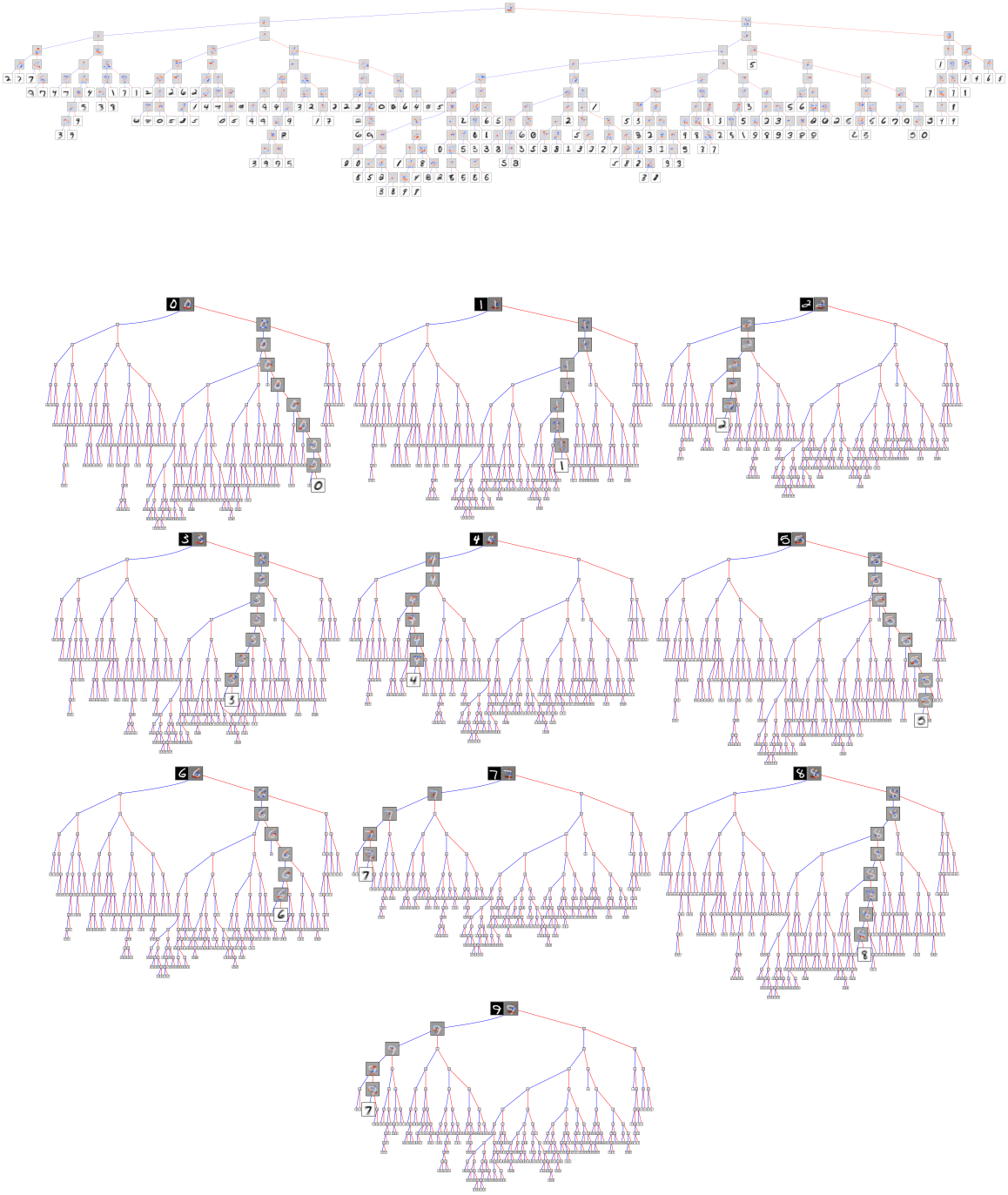


MNIST $\alpha = 10^{-3}$

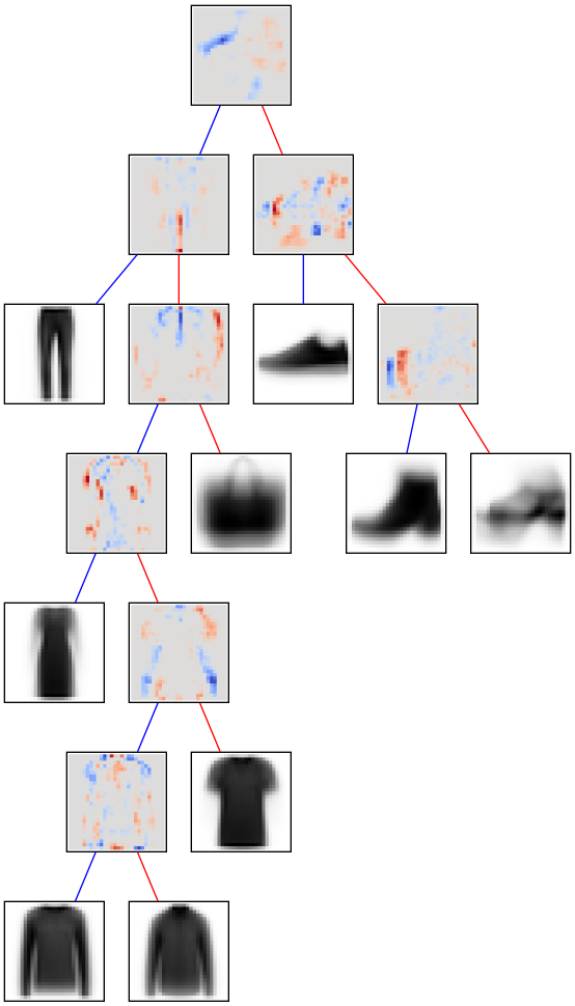


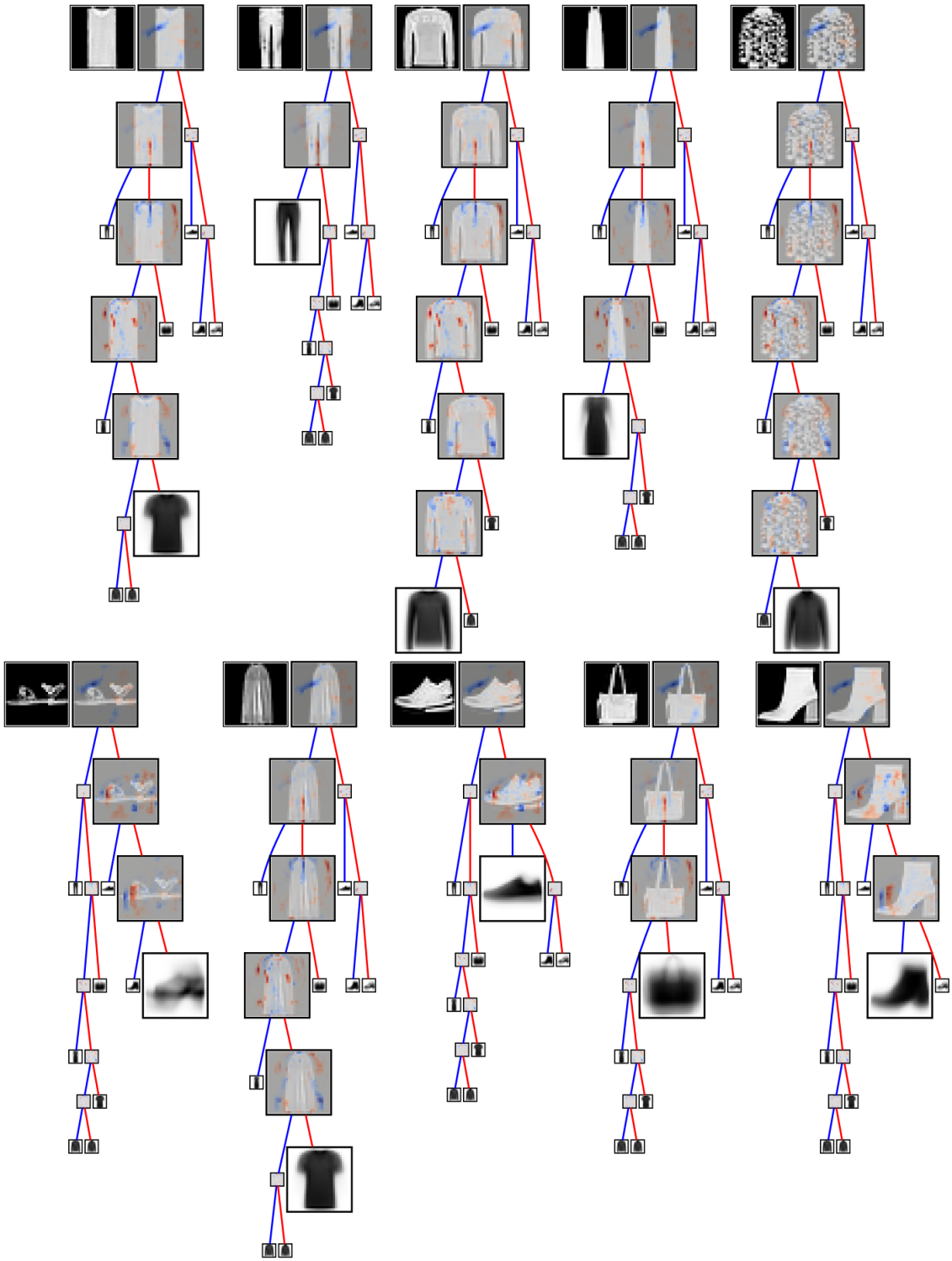


MNIST $\alpha = 10^{-4}$

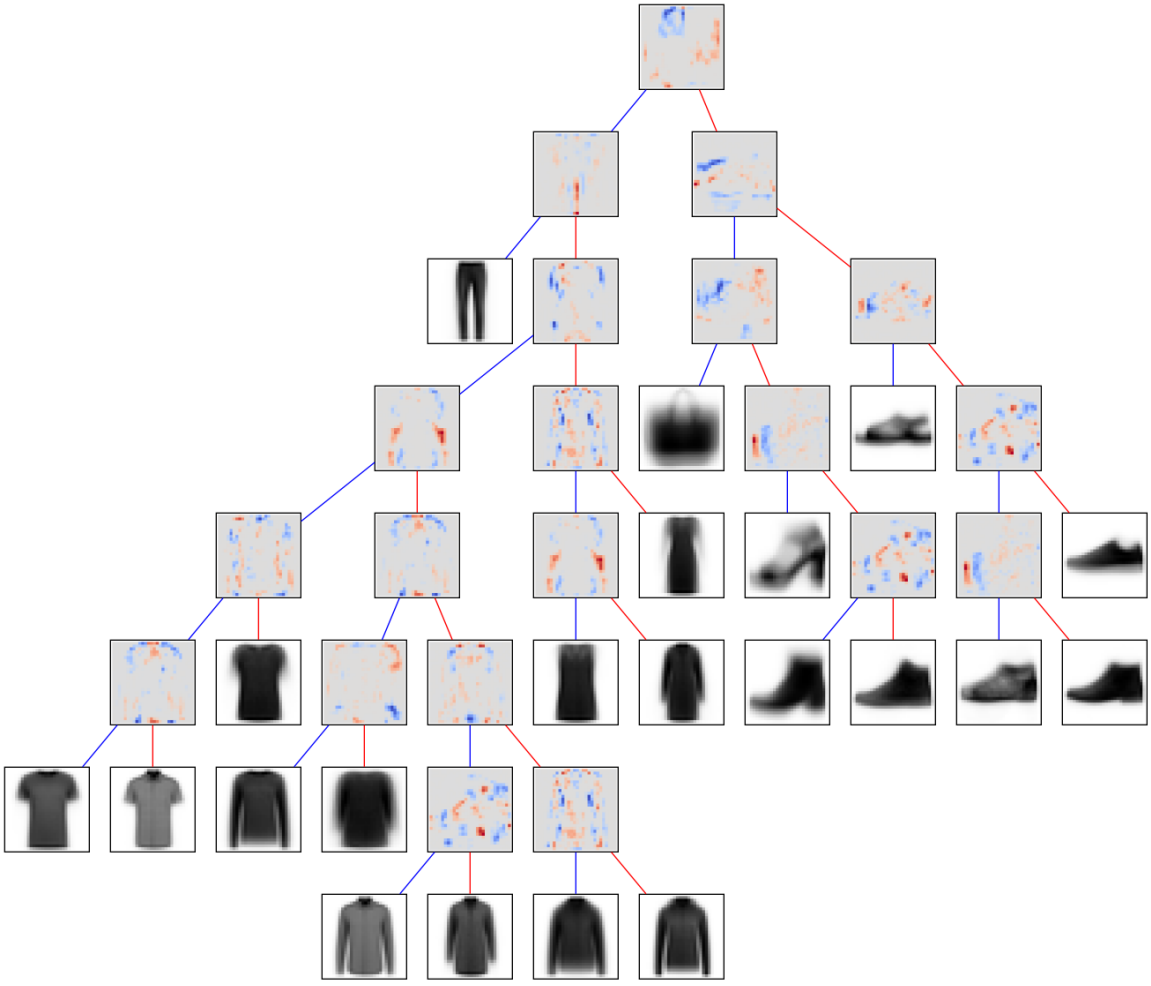


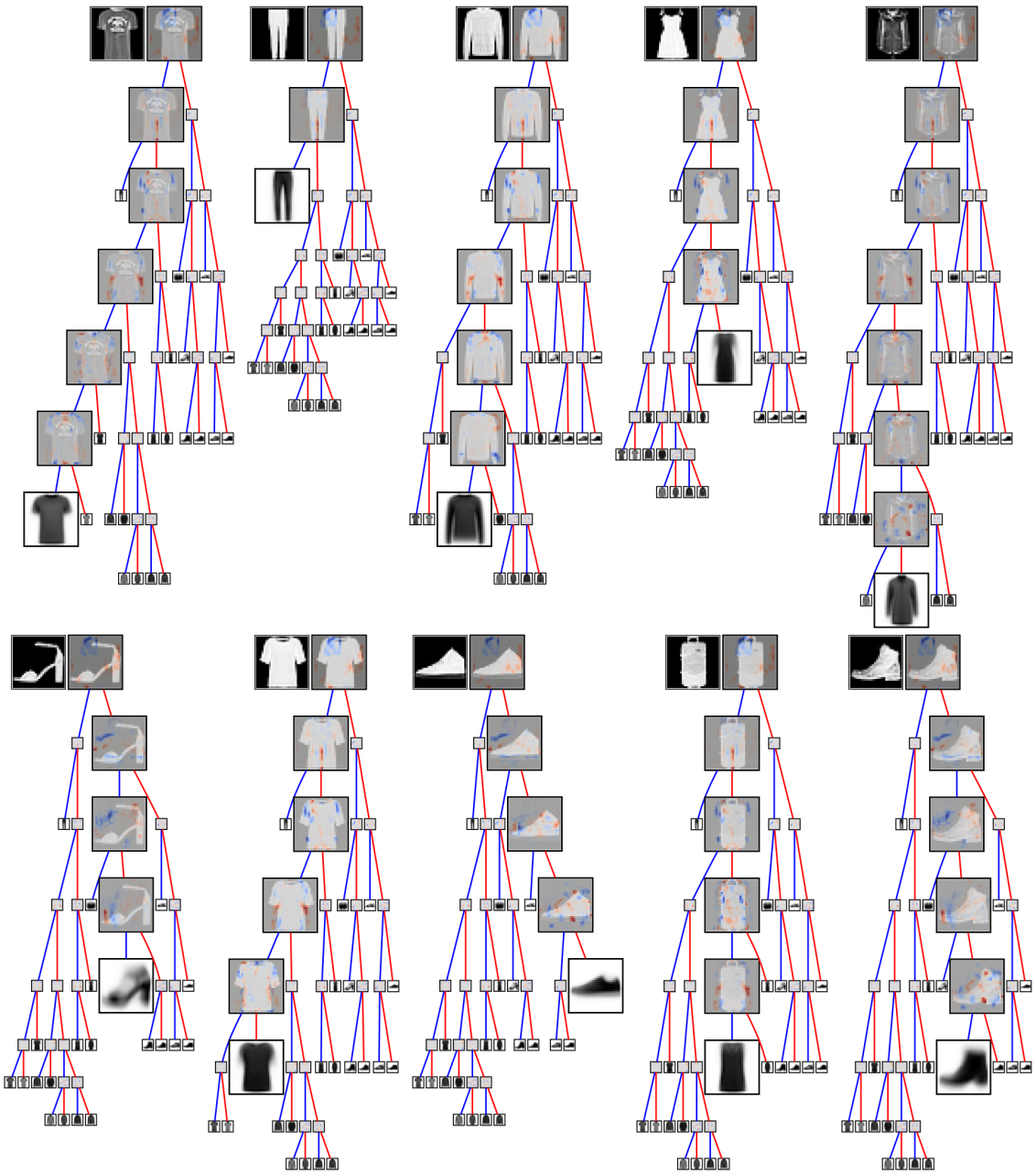
Fashion-MNIST $\alpha = 10^{-2}$



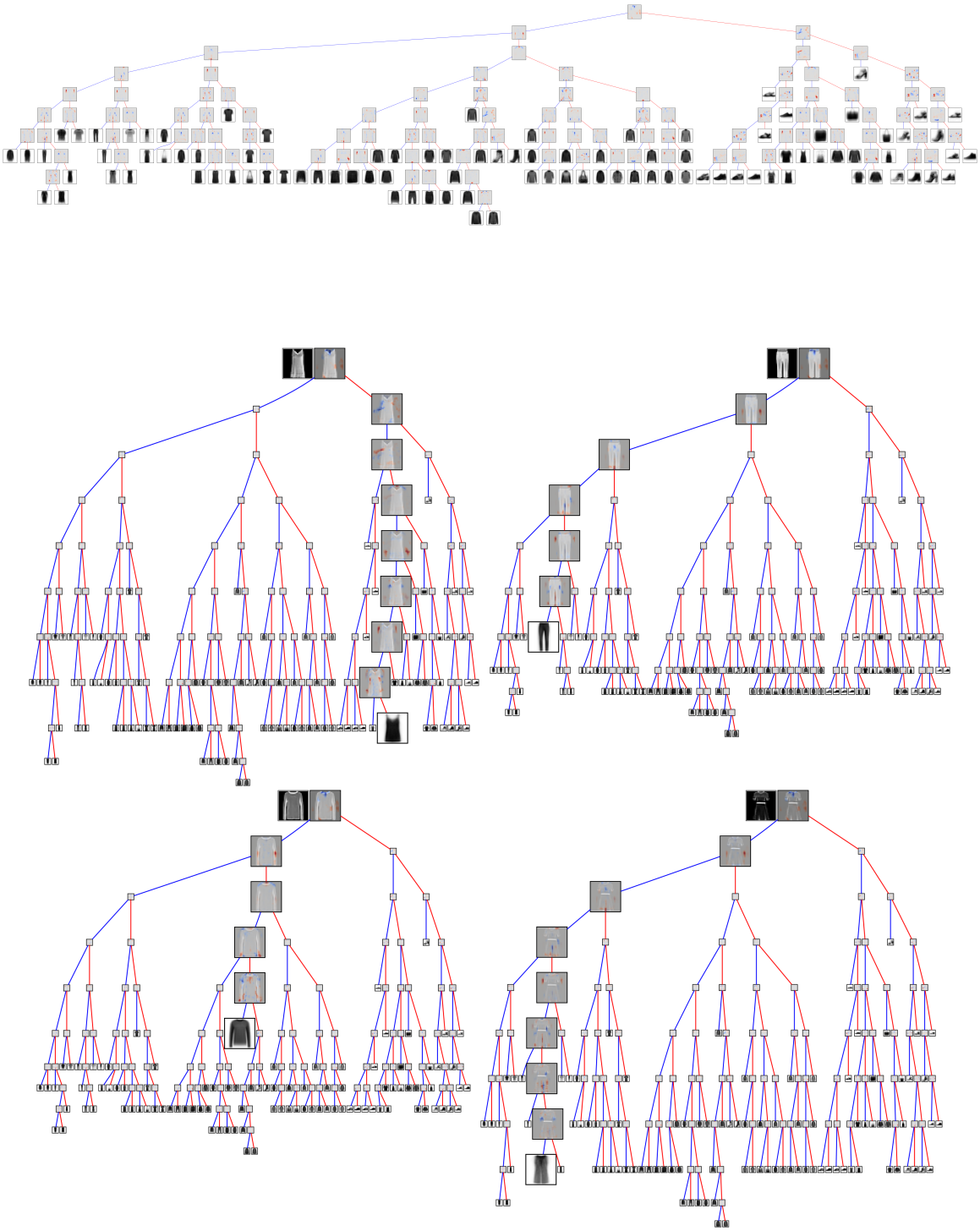


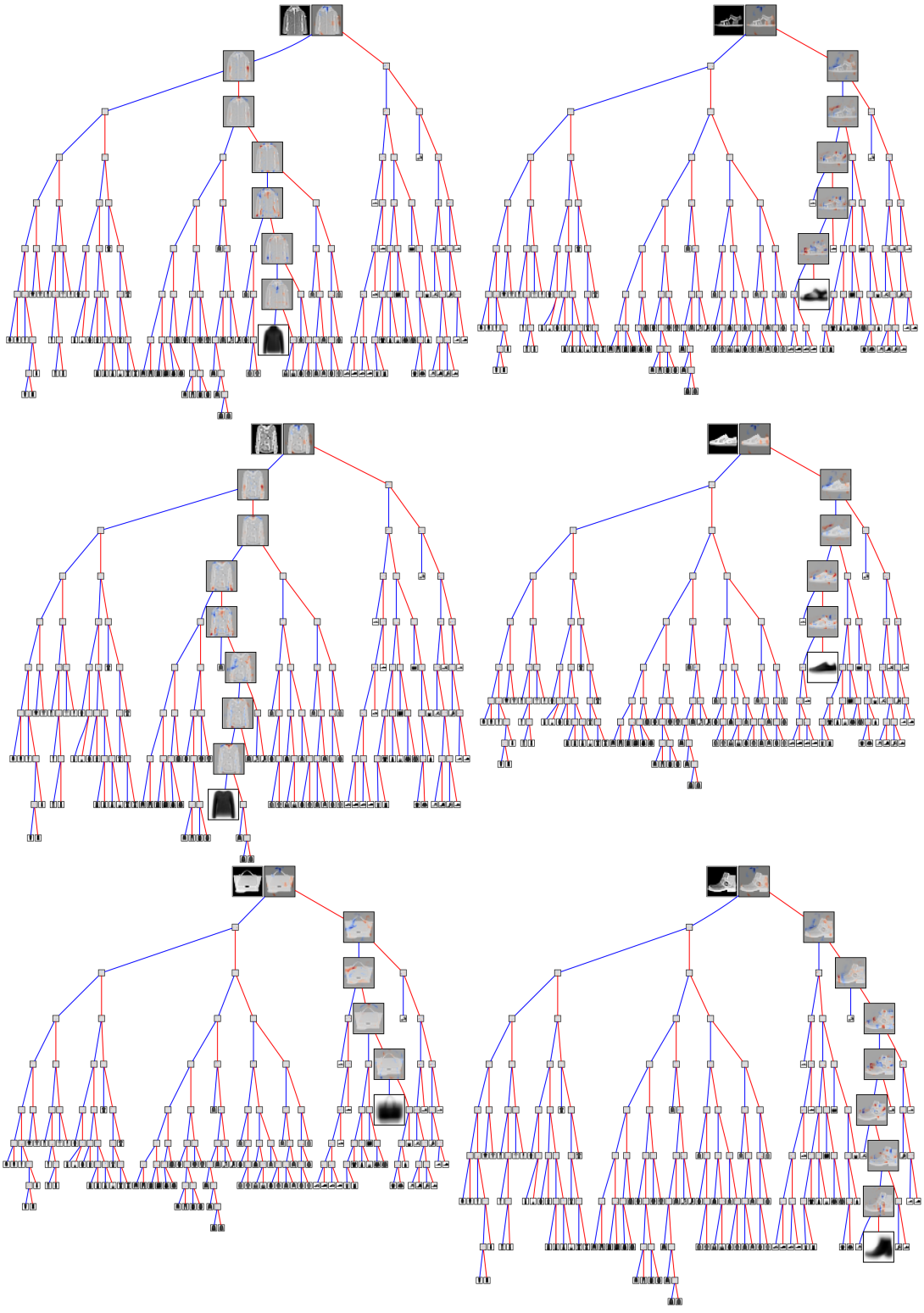
Fashion-MNIST $\alpha = 10^{-3}$





Fashion-MNIST $\alpha = 10^{-4}$





Bibliography

[1] Joaquín Abellán and Serafín Moral. Building classification trees using the total uncertainty criterion. *International Journal of Intelligent Systems*, 18(12):1215–1225, 2003.

[2] Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1991. DOI: 10.24432/C5PC7J.

[3] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[4] Inês Almeida and Joao Xavier. Djam: Distributed jacobi asynchronous method for learning personal models. *IEEE Signal Processing Letters*, 25(9):1389–1392, 2018.

[5] E. Alpaydin and Fevzi. Alimoglu. Pen-Based Recognition of Handwritten Digits. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C5MG6K.

[6] Ayca Altay and Didem Cinar. *Fuzzy Decision Trees*, pages 221–261. Springer International Publishing, 2016.

[7] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 731–744, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Artur Andrzejak, Felix Langner, and Silvestre Zabala. Interpretable models from distributed data via merging of decision trees. In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 1–9, 2013.

[9] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017.

[10] Stanley Bak. nnenum: Verification of ReLU neural networks with optimized abstraction refinement. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods*, pages 19–36, Cham, 2021. Springer International Publishing.

[11] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. *arXiv preprint arXiv:1706.09773*, 2017.

[12] Nicola Bastianello and Emiliano Dall'Anese. Distributed and inexact proximal gradient method for online convex optimization. In *2021 European Control Conference (ECC)*, pages 2432–2437, 2021.

[13] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[14] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials*, 2023.

[15] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

[16] Michael Sh Birman and Michael Z Solomjak. *Spectral theory of self-adjoint operators in Hilbert space*, volume 5. Springer Science & Business Media, 2012.

[17] Avrim Blum and Shuchi Chawla. Learning from Labeled and Unlabeled Data using Graph Mincuts, 6 2018.

[18] Avrim Blum, John Lafferty, Mugizi Robert Rwebangira, and Rajashekar Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 13, New York, NY, USA, 2004. Association for Computing Machinery.

[19] Marko Bohanec. Car Evaluation. UCI Machine Learning Repository, 1997. DOI: 10.24432/C5JP48.

[20] Ferdinand Bollwein and Stephan Westphal. A branch & bound algorithm to determine optimal bivariate splits for oblique decision tree induction. *Applied Intelligence*, 51(10):7552–7572, Oct 2021.

[21] Valerio Bonsignori, Riccardo Guidotti, and Anna Monreale. Deriving a single interpretable model by merging tree-based classifiers. In Carlos Soares and Luis Torgo, editors, *Discovery Science*, pages 347–357, Cham, 2021. Springer International Publishing.

[22] Aaron Bostrom and Anthony Bagnall. A shapelet transform for multivariate time series classification. *arXiv preprint arXiv:1712.06428*, 2017.

[23] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[24] Houssem Ben Braiek and Foutse Khomh. Machine learning robustness: A primer. *arXiv preprint arXiv:2404.00897*, 2024.

[25] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.

[26] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[27] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[28] Miguel A. Carreira-Perpinan and Pooya 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*, volume 31. Curran Associates, Inc., 2018.

[29] Mustafa S. Cetin, Abdullah Mueen, and Vince D. Calhoun. Shapelet ensemble for multi-dimensional time series. In Suresh Venkatasubramanian and Jieping Ye, editors, *SIAM International Conference on Data Mining 2015, SDM 2015*, SIAM International Conference on Data Mining 2015, SDM 2015, pages 307–315. Society for Industrial and Applied Mathematics Publications, 2015. Publisher Copyright: Copyright © SIAM.; SIAM International Conference on Data Mining 2015, SDM 2015 ; Conference date: 30-04-2015 Through 02-05-2015.

[30] Apostolos Chalkis, Ioannis Z Emiris, and Vissarion Fisikopoulos. Practical volume estimation by a new annealing schedule for cooling convex bodies. *arXiv preprint arXiv:1905.05494*, 2019.

[31] B. Chandra and P. Paul Varghese. Fuzzifying gini index based decision trees. *Expert Systems with Applications*, 36(4):8549–8559, 2009.

[32] Robin LP Chang and Theodosios Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on systems, Man, and cybernetics*, 7(1):28–35, 1977.

[33] Nontawat Charoenphakdee, Jongyeong Lee, and Masashi Sugiyama. On symmetric losses for learning from corrupted labels. In *International Conference on Machine Learning*, pages 961–970. PMLR, 2019.

[34] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[35] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[36] Yi-lai Chen, Tao Wang, Ben-sheng Wang, and Zhou-jun Li. A survey of fuzzy decision tree classifier. *Fuzzy Information and Engineering*, 1(2):149–159, 2009.

[37] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53:5113–5155, 2020.

[38] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR, 09–15 Jun 2019.

[39] Patrick L. Combettes and Valérie R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

[40] Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. In D. Touretzky, M.C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

[41] K.A. Crockett, Z. Bandar, and A. Al-Attar. Soft decision trees: a new approach using non-linear fuzzification. In *Ninth IEEE International Conference on Fuzzy Systems. FUZZ- IEEE 2000 (Cat. No.00CH37063)*, volume 1, pages 209–215, 2000.

[42] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 241–251, New York, NY, USA, 1992. Association for Computing Machinery.

[43] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[44] Chao Deng and M. Zu Guo. A new co-training-style random forest for computer aided diagnosis. *Journal of Intelligent Information Systems*, 36(3):253–281, Jun 2011.

[45] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[46] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[47] Yashesh Dhebar and Kalyanmoy Deb. Interpretable rule discovery through bilevel optimization of split-rules of nonlinear decision trees for classification problems. *IEEE Transactions on Cybernetics*, 51(11):5573–5584, 2021.

[48] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[49] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[50] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[51] Martin E. Dyer and Alan M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17(5):967–974, 1988.

[52] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 43–65, Cham, 2020. Springer International Publishing.

[53] Ioannis Z. Emiris and Vissarion Fisikopoulos. Practical polytope volume approximation. *ACM Trans. Math. Softw.*, 44(4), June 2018.

[54] Vladimir Estivill-Castro, Eugene Gilmore, and René Hexel. Constructing interpretable decision trees using parallel coordinates. In Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 152–164, Cham, 2020. Springer International Publishing.

[55] Vladimir Estivill-Castro, Eugene Gilmore, and René Hexel. Human-in-the-loop construction of decision tree classifiers with parallel coordinates. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3852–3859, 2020.

[56] Chenglin Fan and Ping Li. Classification acceleration via merging decision trees. In *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, FODS '20, page 13–22, New York, NY, USA, 2020. Association for Computing Machinery.

[57] Feng-Lei Fan, Jinjun Xiong, Mengzhou Li, and Ge Wang. On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(6):741–760, 2021.

[58] Joel Feldman. UBC Math 511, Lecture Notes: The Spectral Theorem for Commuting, Bounded, Normal Operators. URL: https://personal.math.ubc.ca/~feldman/m511/. Retrieved on 10/13/2024.

[59] Alberto Fernández, Salvador García, Julián Luengo, Ester Bernadó-Mansilla, and Francisco Herrera. Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study. *IEEE Transactions on Evolutionary Computation*, 14(6):913–941, 2010.

[60] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: 10.24432/C56C76.

[61] Timo Freiesleben and Thomas Grote. Beyond generalization: a theory of robustness in machine learning. *Synthese*, 202(4):109, Sep 2023.

[62] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[63] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[64] Magzhan Gabidolla and Miguel Á. Carreira-Perpiñán. Optimal interpretable clustering using oblique decision trees. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 400–410, New York, NY, USA, 2022. Association for Computing Machinery.

[65] Magzhan Gabidolla, Arman Zharmagambetov, and Miguel Á. Carreira-Perpiñán. Improved multiclass adaboost using sparse oblique decision trees. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[66] Cunjing Ge and Feifei Ma. A fast and practical method to estimate volumes of convex polytopes. In Jianxin Wang and Chee Yap, editors, *Frontiers in Algorithmics*, pages 52–65, Cham, 2015. Springer International Publishing.

[67] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

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

[69] Mohamed F. Ghalwash and Zoran Obradovic. Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC Bioinformatics*, 13(1):195, Aug 2012.

[70] Aritra Ghosh, Himanshu Kumar, and P. S. Sastry. Robust loss functions under label noise for deep neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[71] Aritra Ghosh, Naresh Manwani, and P. S. Sastry. On the Robustness of Decision Tree Learning Under Label Noise. In Jinho Kim, Kyuseok Shim, Longbing Cao, Jae-Gil Lee, Xuemin Lin, and Yang-Sae Moon, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 685–697, Cham, 2017. Springer International Publishing.

[72] Aritra Ghosh, Naresh Manwani, and P.S. Sastry. Making risk minimization tolerant to label noise. *Neurocomput.*, 160(C):93–107, jul 2015.

[73] Eugene Gilmore, Vladimir Estivill-Castro, and René Hexel. More interpretable decision trees. In Hugo Sanjurjo González, Iker Pastor López, Pablo García Bringas, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems*, pages 280–292, Cham, 2021. Springer International Publishing.

[74] Nicholas Gisolfi. *Model-centric verification of artificial intelligence*. PhD thesis, Carnegie Mellon University, 2021.

[75] Jack Good, Torin Kovach, Kyle Miller, and Artur Dubrawski. Feature learning for interpretable, performant decision trees. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 66571–66582. Curran Associates, Inc., 2023.

[76] Jack H. Good, Nicholas Gisolfi, Kyle Miller, and Artur Dubrawski. Verification of fuzzy decision trees. *IEEE Transactions on Software Engineering*, pages 1–12, 2023.

[77] Jack H. Good, Kyle Miller, and Artur Dubrawski. Kernel density decision trees. In *Proceedings of the AAAI Spring Symposium on AI Engineering*. Curran Associates, Inc., 2022.

[78] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[79] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[80] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 392–401, New York, NY, USA, 2014. Association for Computing Machinery.

[81] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 507–520. Curran Associates, Inc., 2022.

[82] Bogdan Gulowaty and Michał Woźniak. Extracting interpretable decision tree ensemble from random forest. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[83] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[84] Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 2019. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

[85] Venkatesan Guruswami and Prasad Raghavendra. Hardness of learning halfspaces with noise. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 543–552, 2006.

[86] Suryabhan Singh Hada, Miguel Á Carreira-Perpiñán, and Arman Zharmagambetov. Sparse oblique decision trees: A tool to understand and manipulate neural net features. *Data Mining and Knowledge Discovery*, pages 1–40, 2023.

[87] Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán. Sparse oblique decision trees: A tool to interpret natural language processing datasets. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[88] Bo Han, Quanming Yao, Tongliang Liu, Gang Niu, Ivor W. Tsang, James T. Kwok, and Masashi Sugiyama. A Survey of Label-noise Representation Learning: Past, Present and Future, February 2021. arXiv:2011.04406 [cs].

[89] Satoshi Hara and Yuichi Yoshida. Average sensitivity of decision tree learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[90] S. Hawking and L. Mlodinow. *The Grand Design*. Random House Publishing Group, 2010.

[91] Sudath R. Heiyanthuduwage, Irfan Altas, Michael Bewong, MD Zahidul Islam, and Oscar B. Deho. Decision trees in federated learning: Current state and future opportunities. *IEEE Access*, pages 1–1, 2024.

[92] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, Jul 2014.

[93] Miklós Horváth, Mark Müller, Marc Fischer, and Martin Vechev. (de-) randomized smoothing for decision stump ensembles. *Advances in Neural Information Processing Systems*, 35:3066–3081, 2022.

[94] Niall Hurley and Scott Rickard. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741, 2009.

[95] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Jul 2019.

[96] Sarah Itani, Fabian Lecron, and Philippe Fortemps. A one-class classification decision tree based on kernel density estimation. *Applied Soft Computing*, 91:106250, 2020.

[97] Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. Heart Disease. UCI Machine Learning Repository, 1988. DOI: 10.24432/C52P4X.

[98] Alex Kantchelian, J. D. Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2387–2396, New York, New York, USA, 20–22 Jun 2016. PMLR.

[99] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pages 5132–5143. PMLR, 2020.

[100] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: an efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[101] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*, Jul 2021.

[102] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 443–452, Cham, 2019. Springer International Publishing.

[103] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[104] M. Koklu and I.A. Ozkan. Dry Bean Dataset. UCI Machine Learning Repository, 2020.

[105] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, 2015.

[106] Thomas S. Kuhn. Objectivity, value judgment, and theory choice. *The Essential Tension: Selected Studies in Scientific Tradition and Change*, pages 320–339, 1977.

[107] Saloni Kwatra and Vicenç Torra. A survey on tree aggregation. In *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2021.

[108] Christian Leistner, Amir Saffari, Jakob Santner, and Horst Bischof. Semi-supervised random forests. In *2009 IEEE 12th International Conference on Computer Vision*, pages 506–513, 2009.

[109] Jurica Levatić, Michelangelo Ceci, Dragi Kocev, and Sašo Džeroski. Semi-supervised classification trees. *Journal of Intelligent Information Systems*, 49(3):461–486, Dec 2017.

[110] Alexander Levine, Aounon Kumar, Thomas Goldstein, and Soheil Feizi. Tight second-order certificates for randomized smoothing. *arXiv preprint arXiv:2010.10549*, 2020.

[111] Guozhong Li, Byron Choi, Jianliang Xu, Sourav S Bhowmick, Kwok-Pan Chun, and Grace Lai-Hung Wong. Shapenet: A shapelet-neural network approach for multivariate time series classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):8375–8383, May 2021.

[112] Ming Li and Zhi-Hua Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(6):1088–1098, 2007.

[113] Ruey-Hsia Li and Geneva G. Belford. Instability of decision tree classification algorithms. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 570–575, New York, NY, USA, 2002. Association for Computing Machinery.

[114] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[115] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[116] Elliott H Lieb and Michael Loss. *Analysis*, volume 14. American Mathematical Soc., 2001.

[117] Han Liu, Alexander Gegov, and Mihaela Cocea. *Rule based systems for big data: a machine learning approach*, volume 13. Springer, 2015.

[118] Xiao Liu, Mingli Song, Dacheng Tao, Zicheng Liu, Luming Zhang, Chun Chen, and Jiajun Bu. Semi-supervised node splitting for random forest construction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[119] Xiao Liu, Mingli Song, Dacheng Tao, Zicheng Liu, Luming Zhang, Chun Chen, and Jiajun Bu. Random forest construction with robust semisupervised node splitting. *IEEE Transactions on Image Processing*, 24(1):471–483, 2015.

[120] Zhenyu Liu, Tao Wen, Wei Sun, and Qilong Zhang. Semi-supervised self-training feature weighted clustering decision tree and random forest. *IEEE Access*, 8:128337–128348, 2020.

[121] Haoran Luo, Fan Cheng, Heng Yu, and Yuqi Yi. Sdtr: Soft decision tree regressor for tabular data. *IEEE Access*, 9:55999–56011, 2021.

[122] Carlos J. Mantas and Joaquín Abellán. Credal-c4.5: Decision tree based on imprecise probabilities to classify noisy data. *Expert Systems with Applications*, 41(10):4625–4637, 2014.

[123] Naresh Manwani and P. S. Sastry. Noise tolerance under risk minimization. *IEEE Transactions on Cybernetics*, 43(3):1146–1151, 2013.

[124] Ričards Marcinkevičs and Julia E. Vogt. Interpretable and explainable machine learning: A methods-centric overview with concrete examples. *WIREs Data Mining and Knowledge Discovery*, 13(3):e1493, 2023.

[125] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[126] Roberto Medico, Joeri Ruyssinck, Dirk Deschrijver, and Tom Dhaene. Learning multivariate shapelets with multi-layer neural networks for interpretable time-series classification. *Advances in Data Analysis and Classification*, 15(4):911–936, Dec 2021.

[127] Qiguang Miao, Ying Cao, Ge Xia, Maoguo Gong, Jiachen Liu, and Jianfeng Song. RBoost: Label Noise-Robust Boosting Algorithm Based on a Nonconvex Loss Function and the Numerically Stable Base Learners. *IEEE Transactions on Neural Networks and Learning Systems*, 27(11):2216–2228, November 2016. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[128] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3578–3586. PMLR, 10–15 Jul 2018.

[129] S. Mitra, K.M. Konwar, and S.K. Pal. Fuzzy decision tree, linguistic rules and fuzzy knowledge-based network: generation and evaluation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(4):328–339, 2002.

[130] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

[131] Meike Nauta, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. Pipnet: Patch-based intuitive prototypes for interpretable image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2744–2753, June 2023.

[132] Meike Nauta, Ron Van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14933–14943, 2021.

[133] Duy T Nguyen, Kathryn E Kasmarik, and Hussein A Abbass. Towards interpretable anns: An exact transformation to multi-class multivariate decision trees. *arXiv preprint arXiv:2003.04675*, 2020.

[134] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. In *Proceedings of the 35th Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*, December 2021.

[135] Sebastian Nowozin. Improved information gain estimates for decision tree induction. *arXiv preprint arXiv:1206.4620*, 2012.

[136] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[137] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2233–2241, 2017.

[138] 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 E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[139] 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 E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[140] Leonardo Pellegrina and Fabio Vandin. Scalable rule lists learning with sampling. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2352–2363, 2024.

[141] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 243–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[142] Thanawin Rakthanmanon and Eamonn Keogh. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *proceedings of the 2013 SIAM International Conference on Data Mining*, pages 668–676. SIAM, 2013.

[143] Michael Reed and Barry Simon. *I: Functional analysis*, volume 1. Academic press, 1981.

[144] Lior Rokach. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016.

[145] Dawid Rymarczyk, Łukasz Struski, Michał Górszczak, Koryna Lewandowska, Jacek Tabor, and Bartosz Zieliński. Interpretable image classification with differentiable prototypes assignment. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 351–368, Cham, 2022. Springer Nature Switzerland.

[146] Dawid Rymarczyk, Łukasz Struski, Jacek Tabor, and Bartosz Zieliński. Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1420–1430, 2021.

[147] Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 464–483, 2023.

[148] Omer Sagi and Lior Rokach. Approximating xgboost with an interpretable decision tree. *Information Sciences*, 572:522–542, 2021.

[149] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[150] Stephan R Sain. *Adaptive kernel density estimation*. Rice University, 1994.

[151] Zohaib Salahuddin, Henry C. Woodruff, Avishek Chatterjee, and Philippe Lambin. Transparency of deep neural networks for medical image analysis: A review of interpretability methods. *Computers in Biology and Medicine*, 140:105111, 2022.

[152] M. Sato and H. Tsukimoto. Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 1870–1875 vol.3, 2001.

[153] Terry Sejnowski and R. Gorman. Connectionist Bench (Sonar, Mines vs. Rocks). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5T01Q.

[154] V. Sigillito, S. Wing, L. Hutton, and K. Baker. Ionosphere. UCI Machine Learning Repository, 1989. DOI: https://doi.org/10.24432/C5W01B.

[155] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In Silvia Chiappa and Roberto Calandra, editors, *International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1855–1865. PMLR, 26–28 Aug 2020.

[156] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 1998.

[157] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[158] Suryabhan Singh Hada and Miguel Á. Carreira-Perpiñán. Interpretable image classification using sparse oblique decision trees. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2759–2763, 2022.

[159] Padhraic Smyth, Alexander Gray, and Usama M Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Machine Learning Proceedings 1995*, pages 506–514. Elsevier, 1995.

[160] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8135–8153, 2023.

[161] Zenon A. Sosnowski and Lukasz Gadomer. Fuzzy trees and forests—review. *WIREs Data Mining and Knowledge Discovery*, 9(6):e1316, 2019.

[162] Pedro Strecht, João Mendes-Moreira, and Carlos Soares. Merging decision trees: a case study in predicting student performance. In *Advanced Data Mining and Applications: 10th International Conference, ADMA 2014, Guilin, China, December 19-21, 2014. Proceedings 10*, pages 535–548. Springer, 2014.

[163] Alberto Suarez and James F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, Dec 1999.

[164] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4289–4301, 2022.

[165] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[166] Lukasz Sztukiewicz, Jack Henry Good, and Artur Dubrawski. Exploring loss design techniques for decision tree robustness to label noise. In *The Second Tiny Papers Track at ICLR 2024*, 2024.

[167] Jafar Tanha, Maarten van Someren, and Hamideh Afsarmanesh. Semi-supervised self-training for decision tree classifiers. *International Journal of Machine Learning and Cybernetics*, 8(1):355–370, Feb 2017.

[168] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6166–6175. PMLR, 09–15 Jun 2019.

[169] John Törnblom and Simin Nadjm-Tehrani. An abstraction-refinement approach to formal verification of tree ensembles. In Alexander Romanovsky, Elena Troubitsyna, Ilir Gashi, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 301–313, Cham, 2019. Springer International Publishing.

[170] Hoang Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 18–42. Springer, 2020.

[171] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 670–686, Cham, 2019. Springer International Publishing.

[172] Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter. *Formal Aspects of Computing*, 33(4):519–545, Aug 2021.

[173] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, Feb 2015.

[174] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, Feb 2020.

[175] Gilles Vandewiele, Olivier Janssens, Femke Ongenae, Filip De Turck, and Sofie Van Hoecke. Genesim: genetic extraction of a single, interpretable model. *arXiv preprint arXiv:1611.05722*, 2016.

[176] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*, pages 509–517. PMLR, 2017.

[177] Peter Walley. Inferences from multinomial data: learning about a bag of marbles. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):3–34, 1996.

[178] Jiaqi Wang, Huafeng Liu, Xinyue Wang, and Liping Jing. Interpretable image recognition by constructing transparent embedding space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 895–904, October 2021.

[179] Li-Min Wang, Xiao-Lin Li, Chun-Hong Cao, and Sen-Miao Yuan. Combining decision tree and naive bayes for classification. *Knowledge-Based Systems*, 19(7):511–515, 2006. Creative Systems.

[180] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[181] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, pages 1599–1614, 2018.

[182] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29909–29921. Curran Associates, Inc., 2021.

[183] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[184] Y Wang. Robustness and reliability of machine learning systems: a comprehensive review. *Eng Open*, 1(2):90–95, 2023.

[185] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2017.

[186] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics*, 14(2):513–535, 2023.

[187] William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: 10.24432/C5DW2B.

[188] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[189] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[190] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1129–1141. Curran Associates, Inc., 2020.

[191] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

[192] Mengqi Xue, Qihan Huang, Haofei Zhang, Lechao Cheng, Jie Song, Minghui Wu, and Mingli Song. Protopformer: Concentrating on prototypical parts in vision transformers for interpretable image recognition. *arXiv preprint arXiv:2208.10431*, 2022.

[193] Bin-Bin Yang, Wei Gao, and Ming Li. On the Robust Splitting Criterion of Random Forest. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1420–1425, November 2019. ISSN: 2374-8486.

[194] Greg Yang, Tony Duan, J. Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[195] Linxiao Yang, Jingbang Yang, and Liang Sun. Efficient decision rule list learning via unified sequence submodular optimization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3758–3769, 2024.

[196] Lexiang Ye and Eamonn Keogh. Time series shapelets: A new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 947–956, New York, NY, USA, 2009. Association for Computing Machinery.

[197] Olcay Taner Yıldız and Ethem Alpaydın. Regularizing soft decision trees. In Erol Gelenbe and Ricardo Lent, editors, *Information Sciences and Systems 2013*, pages 15–21, Cham, 2013. Springer International Publishing.

[198] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[199] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[200] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021.

[201] Arman Zharmagambetov and Miguel Carreira-Perpinan. Smaller, more accurate regression forests using tree alternating optimization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11398–11408. PMLR, 13–18 Jul 2020.

[202] Arman Zharmagambetov and Miguel A. Carreira-Perpinan. Semi-supervised learning with decision trees: Graph laplacian tree alternating optimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 2392–2405. Curran Associates, Inc., 2022.

[203] Xu Zhou, Pak Lun Kevin Ding, and Baoxin Li. Improving robustness of random forest under label noise. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 950–958, 2019.

[204] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation, 2002.

[205] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, Nov 2004.