# Optimal Sparse Decision Trees

by

Xiyang Hu

Department of Statistical Science
Duke University

Date: ______________________________

Approved:

______________________________
Cynthia Rudin, Supervisor

______________________________
Jerome Reiter

______________________________
Peter Hoff

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Statistical Science
in the Graduate School of
Duke University

2019

# ABSTRACT

# Optimal Sparse Decision Trees

by

Xiyang Hu

Department of Statistical Science
Duke University

Date: ______________________

Approved:

______________________
Cynthia Rudin, Supervisor

______________________
Jerome Reiter

______________________
Peter Hoff

An abstract of a thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Statistical Science
in the Graduate School of
Duke University

2019

# Abstract

Decision tree algorithms have been among the most popular algorithms for interpretable (transparent) machine learning since the early 1980's. The problem that has plagued decision tree algorithms since their inception is their lack of optimality, or lack of guarantees of closeness to optimality: decision tree algorithms are often greedy or myopic, and sometimes produce unquestionably suboptimal models. Hardness of decision tree optimization is both a theoretical and practical obstacle, and even careful mathematical programming approaches have not been able to solve these problems efficiently. This work introduces the first practical algorithm for optimal decision trees for binary variables. The algorithm is a co-design of analytical bounds that reduce the search space and modern systems techniques, including data structures and a custom bit-vector library. We highlight possible steps to improving the scalability and speed of future generations of this algorithm based on insights from our theory and experiments.

# Acknowledgements

This work is done in collaboration with Professor Cynthia Rudin (Duke University), and Professor Margo Seltzer (The University of British Columbia). I would like to express my sincerest gratitude to both of them, for their kindness, guidance and support. I was very fortunate to have Cynthia and Margo as my advisors. Their brilliance, diligence, rigor and curiosity have been inspiring me on my road of academic research. I also want to thank Jack Dent for providing ideas and implementation help at the early stage of this research project.

# Contents

**Abstract** **iv**

**Acknowledgements** **v**

**List of Figures** **viii**

**List of Tables** **ix**

**1 Introduction** **1**

**2 Related Work** **3**

**3 Optimal Sparse Decision Trees** **6**

3.1 Objective Function . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.2 Optimization Framework . . . . . . . . . . . . . . . . . . . . . . . . 7

3.3 Hierarchical Objective Lower Bound . . . . . . . . . . . . . . . . . . 9

3.4 Lower Bounds on Leaf Support and Classification Accuracy . . . . . . 11

3.5 Permutation Bound . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

3.6 Similar Support Bound . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.7 Equivalent Points Bound . . . . . . . . . . . . . . . . . . . . . . . . 16

**4 Incremental Computation** **18**

**5 Implementation** **22**

5.1 Data Structure of Leaf and Tree . . . . . . . . . . . . . . . . . . . . 22

5.2 Queue . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

5.3 Symmetry-aware Map . . . . . . . . . . . . . . . . . . . . . . . . . . 23

5.4 Execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

**6 Experiments** **25**

**7 Conclusion and Future Work** **31**

**Bibliography** **32**

# List of Figures

6.1 Training accuracy of OSDT, CART, BinOCT on different data (time limit: 30min). Horizontal lines indicate the accuracy of the best OSDT tree. On most datasets, all trees of BinOCT and CART are below this line. . . . . . 26

6.2 Example OSDT execution traces (COMPAS Dataset, $\lambda = 0.005$). Lines are the objective value and dashes are the lower bound for OSDT. For each scheduling policy, we mark the time to optimum and the optimal objective value using a star. . . . . . 27

6.3 Scalability with respect to number of samples and number of features using (multiples of) the ProPublica data set. ($\lambda = 0.005$). Note that all these executions include the 4 features of the optimal tree (Figure 6.4), and the data size are increased by duplicating the whole data set multiple times. . . . . . 28

6.4 The optimal decision tree generated by OSDT on COMPAS dataset. ($\lambda = 0.005$) . . . . . . 28

6.5 The decision tree generated by BinOCT and OSDT on the Tic-Tac-Toe data. Trees of BinOCT must be complete binary trees, while OSDT can generate trees of any shape. . . . . . 30

6.6 The decision tree generated by BinOCT and OSDT on Monk1 dataset. The tree generated by BinOCT includes useless splits, while OSDT can avoid this problem. . . . . . 30

# List of Tables

2.1 Search spaces of rule lists and decision trees with number of variables $p = 10, 20$ and depth $d = 1, 2, 3, 4, 5$. The search space of the trees explodes in comparison. . . . . . . . . . . . . . . . . . . . . . . . . 4

6.1 The sample size and the number of features of the datasets. . . . . . 25

6.2 Per-bound performance improvement, for the ProPublica data set ($\lambda = 0.005$, cold start, using curiosity). The columns report the total execution time, time to optimum, total number of trees evaluated, number of trees evaluated to optimum, and memory consumption. The first row shows our algorithm with all bounds; subsequent rows show variants that each remove a specific bound (one bound at a time, not cumulative). All rows except the last one represent a complete execution, *i.e.*, until the queue is empty. For the last row ('No equivalent points bound'), the algorithm was terminated after running out of the memory about ∼64GB RAM. . . . . . . . . . . . . . . . . . . . . . . . 29

# Chapter 1

# Introduction

Interpretable machine learning has been growing in importance, as society has started to realize the dangers of using black box models for high stakes decisions: complications with confounding have haunted our medical machine learning models [ZBL$^+$18], bad predictions from black boxes have announced to millions of people that their dangerous levels of air pollution were safe [McG18], high-stakes credit risk decisions are being made without proper justification, and black box risk predictions have been wreaking havoc with the perception of fairness of our criminal justice system [FLB16]. In all of these applications – medical imaging, pollution modeling, recidivism risk, credit scoring – interpretable models have been created that are just as accurate as the best black box models (by the CDC, Arnold Foundation, and others). [ALSA$^+$18, CLR$^+$18, etc.]. However, such interpretable models are not generally easy to construct. If we want people to replace their black box models with interpretable models, the tools to build these interpretable models must first exist.

Decision trees are one of the leading forms of interpretable models. Despite several attempts over the last several decades to improve the optimality of decision tree algorithms, the CART [BFOS84] and C4.5 [Qui93] decision tree algorithms (and other greedy tree-growing variants) have remained as dominant methods in practice for decision trees. CART and C4.5 grow decision trees from the top down without backtracking, which means that if a suboptimal split was introduced near the top of the tree, the algorithm could spend many extra splits trying to undo the mistake it made at the top, leading to less-accurate and less-interpretable trees. Problems with greedy splitting and pruning have been known since the early 1990's, when mathematical programming tools had started to be used for creating optimal binary-split decision trees [Ben92, BB96], in a line of work [NF07, BD17, MGKS18, BCMRM18] until the present [VZ19], however, these techniques use all-purpose tools, and tend not to scale to realistically-sized problems unless simplified to trees of a specific form. Other works [KS06] make overly strong assumptions (e.g., independence of all variables) to ensure optimal trees are produced using greedy algorithms.

Our work takes a different approach to optimal decision trees than mathematical programming, greedy methods, brute force, or strong assumptions. We use a composite of analytical bounds to perform massive pruning of the search space, specialized data structures to store intermediate computations and symmetries, and a bit-vector library to evaluate decision trees more quickly on the data, along with fast search policies, computational reuse and other systems-based techniques in order to achieve optimal trees according to a linear balance between accuracy and number of leaves.

This work builds upon recent work on decision lists [ALSA$^+$18, ALSA$^+$17, YRS17, LSAA$^+$18, LRMM15], which are one-sided trees. Decision lists are much easier to optimize than full-blown binary split decision trees; full trees have an exponentially larger number of symmetries. Also, rules for building rule lists can be pre-mined from the database and used to assemble the rule lists, whereas our optimal trees are constructed without these primitives; this means our optimal trees require dramatically more computation. Further, our trees are not required to be a fixed size or shape, which means the trees are optimally sparse for any given level of accuracy, but it also means the search space of possible trees is truly enormous. Despite the hardness of finding optimal solutions, our algorithm is able to locate optimal trees and prove optimality (or closeness of optimality) in reasonable amounts of time for datasets of the sizes used in the criminal justice system (tens of thousands or millions of observations, tens of features). Our algorithm is the first practical optimal decision tree algorithm to achieve solutions for problems of these sizes.

Because we find provably optimal trees, our experiments show where previous studies have claimed to produce optimal models yet failed; we show specific cases where this happens.

We demonstrate our methods on recidivism and credit risk data-sets. These are two of the high-stakes decisions problems where interpretability is needed most in AI systems.

We provide ablation experiments to show which of our techniques is most influential over reducing computation for various datasets. As a result of this analysis, we are able to pinpoint possible future paths to improvement for scalability and computational speed. We believe these could be valuable for an emergent future class of possible new decision tree algorithms.

# Chapter 2

# Related Work

Optimal decision trees have a quite a long history [Ben92], so we focus on recent techniques and their relevance.

There are efficient algorithms that claim to generate optimal sparse trees, but which do not optimally balance the criteria of optimality and sparsity; instead they pre-specify the topology of the tree (*i.e.*, they know *a priori* exactly what the structure of the splits and leaves are, even though they do not know which variables are split) and only find the optimal tree of the given topology [MGKS18]. While this is an important problem, it is not the one we aim to solve, as we do not claim to know the topology of the optimal tree in advance. The most successful current algorithm of this variety is BinOCT [VZ19], which searches for a full and complete binary tree of a given depth, based on continuous-optimization methods for continuous variables, which will be discussed at length later. Some exploration of learning optimal decision trees is based on boolean satisfiability (SAT) [NIP$^+$18], but again, this work looks only for the optimal tree of a given number of nodes.

The DL8 algorithm [NF07] optimizes a ranking function to find a decision tree under constraints of size, depth, accuracy and leaves. DL8 creates trees from the bottom up, meaning that trees are assembled out of all possible leaves, which are itemsets pre-mined from the data [ALSA$^+$18, similarly to]. DL8 does not have publicly available sourcecode. [NF07] warn of issues of running out of memory when storing all partially-constructed trees.

There are several works that produce oblique trees [BCMRM18], where splits involve several variables; oblique trees are not addressed here, as they can be less interpretable.

The most recent mathematical programming algorithms are OCT [BD17] and BinOCT [VZ19]. Example figures from the OCT paper [BD17] show decision trees that are clearly suboptimal. However, as the code was not made public, the work in the OCT paper [BD17] is not easily reproducible, so it is not clear where the problem occurred. We discuss this in Section §6. [VZ19] introduce a much faster mathematical programming formulation, and their experiments indicate that BinOCT outperforms OCT, but since BinOCT is constrained to create complete binary trees of a given depth rather than optimally sparse trees, it sometimes creates unnecessary leaves in order to complete a tree at a given depth, as we show in Section §6. BinOCT solves a dramatically easier problem than the method introduced in this work. As it turns out, the search space of perfect binary trees of a given depth is much smaller than that of binary trees with the same number of leaves. For instance, the number of different unlabeled binary trees with 8 leaves is $Catalan(7) = 429$, but the number

Search Space of CORELS and Decision Trees

| | $p = 10$ | | $p = 20$ | |
|---|---|---|---|---|
| $d$ | Rule Lists | Trees | Rule Lists | Trees |
| 1 | $5.500 \times 10^1$ | $1.000 \times 10^1$ | $2.100 \times 10^2$ | $2.000 \times 10^1$ |
| 2 | $3.025 \times 10^3$ | $1.000 \times 10^3$ | $4.410 \times 10^4$ | $8.000 \times 10^3$ |
| 3 | $1.604 \times 10^5$ | $5.329 \times 10^6$ | $9.173 \times 10^6$ | $9.411 \times 10^8$ |
| 4 | $8.345 \times 10^6$ | $9.338 \times 10^{20}$ | $1.898 \times 10^9$ | $9.204 \times 10^{28}$ |
| 5 | $4.257 \times 10^8$ | Inf | $3.911 \times 10^{11}$ | Inf |

**Table 2.1**: Search spaces of rule lists and decision trees with number of variables $p = 10, 20$ and depth $d = 1, 2, 3, 4, 5$. The search space of the trees explodes in comparison.

of unlabeled perfect binary trees with 8 leaves is only 1. In our setting, we penalize (but do not fix) the number of leaves, which means that our search space contains all trees, though we can bound the maximum number of leaves based on the size of the regularization parameter. Therefore, our search space is much larger than that of BinOCT.

Our work builds upon the CORELS algorithm of [ALSA$^+$18, ALSA$^+$17, LSAA$^+$18] and its predecessors [YRS17, LRMM15] which creates optimal decision lists (rule lists). Applying those ideas to decision trees is nontrivial. The rule list optimization problem is much easier, since the rules are pre-mined (there is no mining of rules in our decision tree optimization). Rule list optimization involves selecting an optimal subset and an optimal permutation of the rules in each subset. Decision tree optimization involves considering every possible split of every possible variable, and every possible shape and size of tree. This is an exponentially harder optimization problem with a huge number of symmetries to consider. In addition, in CORELS, the maximum number of clauses per rule is set to be $c = 2$. For a data set with $p$ features, there would be $D = p + \binom{p}{2}$ rules in total, and the number of distinct rule lists with $d_r$ rules is $P(D, d_r)$, where $P(m, k)$ is the number of $k$-permutations of $m$. Therefore, the search space of CORELS is $\sum_{d_r=1}^{d-1} P(D, d_r)$. But, for a full binary tree with depth $d_t$ and data with $p$ features, the number of distinct trees is:

$$N_{d_t} = \sum_{n_0=1}^{1} \sum_{n_1=1}^{2^{n_0}} \cdots \sum_{n_{d_t-1}=1}^{2^{n_{d_t-2}}} p \times \binom{2^{n_0}}{n_1} (p-1)^{n_1} \times \cdots \times \binom{2^{n_{d_t-2}}}{n_{d_t-1}} (p-(d_t-1))^{n_{d_t-1}}, \tag{2.1}$$

and the search space of decision trees up to depth $d$ is $\sum_{d_t=1}^{d} N_{d_t}$. Table 2.1 shows how the search spaces of rule lists and decision trees grow as the tree depth increases. The search space of the trees is massive compared to that of the rule lists.

Applying techniques from rule lists to decision trees necessitated new designs for the data structures, splitting mechanisms and bounds. An important difference between rule lists and trees is that during the growth of rule lists, we only add one new rule to the list each time, but for the growth of trees, we need to split existing leaves and add a new *pair* of leaves for each. This leads to several bounds that are quite different from those in CORELS, *i.e.*, Theorem 4, Theorem 5 and Corollary 6, which consider a pair of leaves rather than a single leaf. In this paper, we introduce bounds only for the case of one split at a time; however, in our implementation, we can split more than one leaf at a time, and the bounds are adapted accordingly.

# Chapter 3

# Optimal Sparse Decision Trees

We focus on binary classification, and our decision trees are Boolean functions. It is possible to generalize this framework to multiclass settings or weighted data. We denote training data as $\{(x_n, y_n)\}_{n=1}^N$, where $x_n \in \{0,1\}^M$ are binary features and $y_n \in \{0,1\}$ are labels. Let $\mathbf{x} = \{x_n\}_{n=1}^N$ and $\mathbf{y} = \{y_n\}_{n=1}^N$, and let $x_{n,m}$ denote the $m$-th feature of $x_n$.

For a decision tree, what we actually obtain is a set of leaves or conjunction of predicates. These leaves are mutually exclusive, which means their order does not matter in evaluating the accuracy of the tree. A tree can thus be expressed in terms of only its leaves. A leaf set $d = (p_1, p_2, \ldots, p_H)$ of length $H \geq 0$ is an $H$-tuple containing $H$ distinct leaves, and every $p_k$ corresponds to its $\hat{y}_k^{(\text{leaf})}$, for $k = 1, \ldots, H$, where $p_k$ is the classification rule of the path from the root to leaf $k$ and $\hat{y}_k^{(\text{leaf})}$ is the label for leaf $k$. In our setting, $p_k$ is a Boolean assertion, which evaluates to be either true or false for each datum $x_n$, and $\hat{y}_k^{(\text{leaf})}$ is a label prediction for all data in the leaf $k$. For example, $(x_{n,1} = 0) \wedge (x_{n,2} = 1)$ defines $p_k$ and $(\hat{y}_n = 1)$ is the predicted label of datum $x_n$.

We explore the search space by considering which leaves of the tree can be beneficially split. The leaf set $d = (p_1, p_2, \ldots, p_K, p_{K+1}, \ldots, p_H)$ is the $H$-leaf tree, where the first $K$ leaves are restricted not to be split, and the remaining $H-K$ leaves can be split. We alternately represent this leaf set as $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, where $d_{un} = (p_1, \ldots, p_K)$ are the unchanged leaves of $d$, $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})}) \in \{0,1\}^K$ are the predicted labels of leaves $d_{un}$, and $d_{\text{split}} = (p_{K+1}, \ldots, p_H)$ are the leaves we are going to split, $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})}) \in \{0, 1\}^{H-K}$ are the predicted labels of leaves $d_{\text{split}}$. We call $d_{un}$ a $K$-prefix of $d$, which means its leaves are a size-$K$ subset of $(p_1, \ldots, p_K, \ldots, p_H)$. If we have a new prefix $d'_{un}$, which is a supset of $d_{un}$, *i.e.*, $d'_{un} \supseteq d_{un}$, then we say $d'_{un}$ starts with $d_{un}$. We define $\sigma(d_{un})$ to be all $d$'s child trees whose (unchanged) leaves include $d_{un}$:

$$\sigma(d_{un}) = \{(d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H') : d'_{un} \text{ starts with } d_{un}\}. \tag{3.1}$$

If we have two prefix $d_{un} = (p_1, \ldots, p_K)$ and $d'_{un} = (p_1, \ldots, p_K, p_{K+1})$, and $d'_{un}$ starts with $d_{un}$ and contain one more leaf, then we define $d'_{un}$ to be a child of $d_{un}$ and $d_{un}$ to be a parent of $d'_{un}$.

Note that in our setting, two trees with identical leaf sets, but with different assignments to $d_{un}$ and $d_{\text{split}}$, are *different* trees. Further, a child tree can *only* be generated through splitting those leaves of its parent tree within $d_{\text{split}}$.

A tree $d$ classifies datum $x_n$ by providing the label prediction $\hat{y}_k^{(\text{leaf})}$ of the leaf whose $p_k$ is true for $x_n$. If $p_k$ evaluates to true for $x_n$, we say the leaf $k$ of leaf set $d_{un}$

captures $x_n$ . In our setting, all the data captured by a prefix's leaves are also captured by the prefix itself.

Let $\beta$ be a set of leaves. We define $\text{cap}(x_n, \beta) = 1$ if a leaf in $\beta$ captures datum $x_n$, and 0 otherwise. For example, let $d_{un}$ and $d'_{un}$ be prefixes such that $d'_{un}$ starts with $d_{un}$, then $d'_{un}$ captures all the data that $d_{un}$ captures:

$$\{x_n : \text{cap}(x_n, d_{un})\} \subseteq \{x_n : \text{cap}(x_n, d'_{un})\}.$$

Now let $d_{un}$ be a set of leaves, and let $\beta$ be a subset of leaves in $d_{un}$. We denote the normalized support of $\beta$ as $\text{supp}(\beta, \mathbf{x})$, which is the fraction of the dataset captured by $\beta$:

$$\text{supp}(\beta, \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, \beta). \tag{3.2}$$

In our setting, decision trees are empirically set to minimize the regularized misclassification error. Therefore, the combination of training data $(\mathbf{x}, \mathbf{y})$, a leaf set $d_{un} = (p_1, \ldots, p_K)$ and $d_{\text{split}} = (p_{K+1}, \ldots, p_H)$ denotes a tree $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ with prefix $d_{un}$, where $\hat{y}_h^{(\text{leaf})}$, $(1 \leq h \leq H)$, corresponds to the majority label of data captured by leaf $p_h$ within the set of label predictions $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})})$ and $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$. Throughout this paper, we always assume a decision tree assigns labels to its leaves in this way.

## 3.1 Objective Function

For a tree $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, we define its objective function as a combination of the misclassification error and a sparsity penalty on the number of leaves:

$$R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda H(d). \tag{3.3}$$

$H(d)$ is the number of leaves in the tree $d$. $R(d, \mathbf{x}, \mathbf{y})$ is a regularized empirical risk. The loss $\ell(d, \mathbf{x}, \mathbf{y})$, is the misclassification error of $d$, *i.e.*, the fraction of training data with incorrectly predicted labels. $\lambda H(d)$ is a regularization term which penalizes bigger trees. The regularization parameter $\lambda \geq 0$ is a small user-defined constant. It imposes a penalty whereby misclassifying a fraction of $\lambda$ of the training data suffers the same as adding one to the number of leaves.

## 3.2 Optimization Framework

We minimize the objective function based on a branch-and-bound framework. We propose a series of specialized useful bounds that work together to eliminate a large part of the search space. These bounds are discussed in detail in the following paragraphs.

Some of our bounds could be adapted directly from CORELS [ALSA+18], namely these two:

- **(Hierarchical objective lower bound)** Lower bounds of a parent tree also hold for every child tree of that parent. (§3.3, Theorem 1).
- **(Equivalent points bound)** For a given dataset, if there are multiple samples with exactly the same features but different labels, then no matter how we build our classifier, we would always make mistakes. And the lower bound of number of mistakes is the number of such samples with minority labels. (§3.7, Theorem 9).

Some of our bounds adapt from CORELS [ALSA+17] with minor changes:

- **(Objective lower bound with one-step lookahead)** With respect to the number of leaves, if a tree copy does not achieve enough accuracy, we can prune all child trees of it. (§3.3, Lemma 2).
- **(Lower bound on leaf support)** For an optimal tree, the support of each pair of leaves in the tree must be above $2\lambda$. (§3.4, Theorem 3).

Some of our bounds are distinct from CORELS because they are only relevant to trees and not to lists:

- **(Lower bound on incremental classification accuracy)** Each split must result in sufficient reduction of the loss. The loss reduction should be larger than the increase of the regularization term. Otherwise, we need to further split at least one of the child leaves (§3.4, Theorem 4).
- **(Leaf permutation bound)** We only need to consider the optimal permutation of leaves in a tree; we do not need to consider all other permutations (explained in §3.5, Corollary 6).
- **(Leaf accurate support bound)** Each leaf in an optimal decision tree must have misclassification error that is sufficiently small. Specifically, for each leaf in an optimal decision tree, the number of correctly classified samples must be above a threshold. (§3.4, Theorem 5).
- **(Similar support bound)** Given two trees are exactly the same but one internal node split by different similar enough features, if we know one of them and all its child trees cannot be the optimal tree, we can also prune the other one and all its child trees (§3.6, Theorem 7).

## 3.3 Hierarchical Objective Lower Bound

The loss can be decomposed into two parts corresponding to the unchanged leaves and the leaves to be split:

$$\ell(d, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \ell_q(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y}),$$

where $d_{un} = (p_1, \ldots, p_K)$, $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})})$, $d_{\text{split}} = (p_{K+1}, \ldots, p_H)$ and $\delta_{\text{split}} = (\hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$;

$$\ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$$

is the proportion of data in the unchanged leaves that are misclassified, and

$$\ell_p(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{H} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$$

is the proportion of data in the leaves we are going to split that are misclassified. We define a lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ on the objective by leaving out the latter loss,

$$b(d_{un}, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H \leq R(d, \mathbf{x}, \mathbf{y}), \tag{3.4}$$

where the leaves $d_{un}$ are kept and the leaves $d_{\text{split}}$ are going to be split. Here, $b(d_{un}, \mathbf{x}, \mathbf{y})$ gives a lower bound on the objective of *any* child tree of $d$.

**Theorem 1** (Hierarchical objective lower bound)**.** *Define $b(d_{un}, \mathbf{x}, \mathbf{y}) = \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H$, as in* (3.4)*. Define $\sigma(d_{un})$ to be the set of all $d$'s child trees whose unchanged leaves contains $d_{un}$, as in* (3.1)*. Let $d = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ be a tree with unchanged leaves $d_{un}$, and let $d' = (d'_{un}, \delta'_{un}, d'_{split}, \delta'_{split}, K', H') \in \sigma(d_{un})$ be any child tree such that its unchanged leaves $d'_{un}$ contain $d_{un}$ and $K' \geq K, H' \geq H$, then $b(d_{un}, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y})$.*

*Proof.* Let $d_{un} = (p_1, \ldots, p_K)$ and $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})})$; let $d'_{un} = (p_1, \ldots, p_K, p_{K+1}, \ldots, p_{K'})$ and $\delta'_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_K^{(\text{leaf})}, \hat{y}_{K+1}^{(\text{leaf})}, \ldots, \hat{y}_{K'}^{(\text{leaf})})$. Note that the mistakes made by $d'_{un}$ are no less than those of $d_{un}$:

$$\begin{aligned}
\ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] \\
&= \frac{1}{N} \sum_{n=1}^{N} \left( \sum_{k=1}^{K} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] + \sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] \right) \\
&= \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] \geq \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}),
\end{aligned} \tag{3.5}$$

**Algorithm 1** Branch-and-bound for learning optimal decision trees.

```
Input: Objective function $R(d, \mathbf{x}, \mathbf{y})$, objective lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$, set of features $S = \{s_m\}_{m=1}^M$, training data $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^N$, initial best known tree $d^0$ with objective $R^0 = R(d^0, \mathbf{x}, \mathbf{y})$; $d^0$ could be obtained as output from another (approximate) algorithm, otherwise, $(d^0, R^0) = (\text{null}, 1)$ provides reasonable default values
Output: Provably optimal decision tree $d^*$ with minimum objective $R^*$

$(d^c, R^c) \leftarrow (d^0, R^0)$                                   ▷ Initialize best tree and objective
$Q \leftarrow \text{queue}(\,[\,((), (), (), \delta_{\text{split}}, 0, 0)\,]\,)$            ▷ Initialize queue with empty tree
while $Q$ not empty do                                                ▷ Stop when queue is empty
    $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H) \leftarrow Q.\text{pop}(\,)$    ▷ Remove tree $d$ from the queue
    if $b(d_{un}, \mathbf{x}, \mathbf{y}) < R^c$ then                               ▷ Bound: Apply Theorem 1
        $R \leftarrow R(d, \mathbf{x}, \mathbf{y})$                                 ▷ Compute objective of tree $d$
        if $R < R^c$ then                                             ▷ Update best tree and objective
            $(d^c, R^c) \leftarrow (d, R)$
        end if
        for every possible combination of features to split $d_{\text{split}}$ do
                                                                      ▷ Branch: Enqueue $d_{un}$'s children
            split $d_{\text{split}}$ and get new leaves $d_{\text{new}}$
            for each possible subset $d'_{\text{split}}$ of $d_{\text{new}}$ do
                $d'_{un} = d_{un} \cup (d_{\text{new}} \setminus d'_{\text{split}})$
                $Q.\text{push}(\,(d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')\,)$
            end for
        end for
    end if
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$                                   ▷ Identify provably optimal solution
```

It follows that

$$\begin{aligned} b(d_{un}, \mathbf{x}, \mathbf{y}) &= \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H \\ &\leq \ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) + \lambda H' = b(d'_{un}, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y}). \end{aligned} \tag{3.6}$$

□

Consider a sequence of trees, where each tree is the parent of the following tree. In this case, the lower bounds of these trees increase monotonically, which is amenable to branch-and-bound. We illustrate our framework in Algorithm 1.

According to Theorem 1, we can hierarchically prune the search space. During the execution of the algorithm, we cache the current best (smallest) objective $R^c$, which is dynamic and monotonically decreasing. In this process, when we generate a tree whose unchanged leaves $d_{un}$ corresponding to lower bound $b(d_{un}, \mathbf{x}, \mathbf{y}) \geq R^c$, according to Theorem 1, we do not need to consider *any* child tree $d' \in \sigma(d_{un})$ of this tree whose $d'_{un}$ contains $d_{un}$. This is because we have $b(d'_{un}, \mathbf{x}, \mathbf{y}) \geq b(d_{un}, \mathbf{x}, \mathbf{y})$, which leads to $R(d', \mathbf{x}, \mathbf{y}) \geq b(d'_{un}, \mathbf{x}, \mathbf{y}) \geq b(d_{un}, \mathbf{x}, \mathbf{y}) \geq R^c$. Therefore, any child of such a tree cannot be the optimal tree, and we can prune the corresponding search space.

Based on Theorem 1, we describe a consequence in Lemma 2.

**Lemma 2** (Objective lower bound with one-step lookahead). *Let $d$ be a $H$-leaf tree with a $K$-leaf prefix and let $R^c$ be the current best objective. If $b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c$, then for any child tree $d' \in \sigma(d_{un})$, its prefix $d'_{un}$ starts with $d_{un}$ and $K' > K, H' > H$, and it follows that $R(d', \mathbf{x}, \mathbf{y}) \geq R^c$.*

*Proof.*

$$\begin{aligned}
R(d'_{un}, \mathbf{x}, y) \geq b(d'_{un}, \mathbf{x}, \mathbf{y}) &= \ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) + \lambda H' \\
&= \ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) + \lambda H + \lambda(H' - H) \\
&\geq \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H + \lambda(H' - H) \\
&= b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda(H' - H) \\
&\geq b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c. \qquad (3.7)
\end{aligned}$$

□

Therefore, even though we might have a tree with unchanged leaves $d_{un}$ whose lower bound $b(d_{un}, \mathbf{x}, \mathbf{y}) \leq R^c$, if $b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c$, we can still prune all its child trees with a larger number of leaves and unchanged leaves $d'_{un} \supseteq d_{un}$.

## 3.4 Lower Bounds on Leaf Support and Classification Accuracy

We provide three lower bounds on the fraction of correctly classified data and the normalized support of leaves in any optimal tree. All of them are relevant with $\lambda$.

**Theorem 3** (Lower bound on leaf support). *Let $d^* = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ be any optimal tree with objective $R^*$, i.e., $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. For each child leaf pair $p_k, p_{k+1}$ of a split, the sum of normalized supports of $p_k, p_{k+1}$ should be no less than twice the regularization parameter, i.e., $2\lambda$,*

$$2\lambda \leq \operatorname{supp}(p_k, \mathbf{x}) + \operatorname{supp}(p_{k+1}, \mathbf{x}). \qquad (3.8)$$

*Proof.* Let $d^* = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ be an optimal tree with leaves $(p_1, \ldots, p_H)$ and labels $(\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$. Consider the tree $d = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ derived from $d^*$ by deleting a pair of leaves $p_i \to \hat{y}_i^{(\text{leaf})}, p_{i+1} \to \hat{y}_{i+1}^{(\text{leaf})}$ and adding the their parent leaf $p_j \to \hat{y}_j^{(\text{leaf})}$, therefore $d'_{un} = (p_1, \ldots, p_{i-1}, p_{i+2}, \ldots, p_H, p_j)$ and $\delta'_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_{i-1}^{(\text{leaf})}, \hat{y}_{i+2}^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})}, \hat{y}_j^{(\text{leaf})})$.

When $d$ misclassified half of the data captured by $p_i, p_{i+1}$, while $d^*$ correctly classified them all, the difference between $d$ and $d^*$ would maximize, which provides an upper bound:

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda(H-1) &\leq \ell(d^*, \mathbf{x}, \mathbf{y}) + \text{supp}(p_i, \mathbf{x}) + \text{supp}(p_{i+1}, \mathbf{x}) \\ &\quad - \frac{1}{2}[\text{supp}(p_i, \mathbf{x}) + \text{supp}(p_{i+1}, \mathbf{x})] + \lambda(H-1) \\ &= R(d^*, \mathbf{x}, \mathbf{y}) + \frac{1}{2}[\text{supp}(p_i, \mathbf{x}) + \text{supp}(p_{i+1}, \mathbf{x})] - \lambda \\ &= R^* + \frac{1}{2}[\text{supp}(p_i, \mathbf{x}) + \text{supp}(p_{i+1}, \mathbf{x})] - \lambda \qquad (3.9) \end{aligned}$$

where $\text{supp}(p_i, \mathbf{x}), \text{supp}(p_i, \mathbf{x})$ is the normalized support of $p_i, p_{i+1}$, defined in (3.2), and the regularization 'bonus' comes from the fact that $d^*$ have one more leaf than $d$.

Because $d^*$ is the optimal tree, we have $R^* \leq R(d, \mathbf{x}, \mathbf{y})$, which combined with (3.9) leads to (3.8). Therefore, for each child leaf pair $p_k, p_{k+1}$ of a split, the sum of normalized supports of $p_k, p_{k+1}$ should be no less than twice the regularization parameter, *i.e.*, $2\lambda$. $\square$

Therefore, for a tree $d$, if any of its leaf pairs captures less than a fraction $2\lambda$ of samples, it is unlikely to be the optimal tree, even if $b(d_{un}, \mathbf{x}, \mathbf{y}) < R^*$. None of its child trees would be an optimal tree. Thus, after evaluating $d$, we can prune tree $d$. Theorem 3 only depends on the amount of data captured by the leaves. In Theorem 4, we provide a tighter upper bound on $R(d, \mathbf{x}, \mathbf{y})$ in (3.9) by also leveraging the classification accuracy.

**Theorem 4** (Lower bound on incremental classification accuracy)**.** *Let $d^* = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ be any optimal tree with objective $R^*$, i.e., $d^* \in \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. Let $d^*$ have leaves $d_{un} = (p_1, \ldots, p_H)$ and labels $\delta_{un} = (\hat{y}_1^{(leaf)}, \ldots, \hat{y}_H^{(leaf)})$. For each leaf pair $p_k, p_{k+1}$ and the corresponding labels $\hat{y}_k^{(leaf)}, \hat{y}_{k+1}^{(leaf)}$ in $d^*$ and the their parent node (the leaf in the parent tree) $p_j$ and its label $\hat{y}_j^{(leaf)}$, define $a_k$ to be the incremental classification accuracy of splitting $p_j$ to get $p_k, p_{k+1}$:*

$$\begin{aligned} a_k \equiv \frac{1}{N} \sum_{n=1}^{N} \{ &\text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(leaf)} = y_n] \\ &+ \text{cap}(x_n, p_{k+1}) \wedge \mathbb{1}[\hat{y}_{k+1}^{(leaf)} = y_n] \\ &- \text{cap}(x_n, p_j) \wedge \mathbb{1}[\hat{y}_j^{(leaf)} = y_n] \}. \qquad (3.10) \end{aligned}$$

*In this case, $\lambda$ provides a lower bound, $\lambda \leq a_k$.*

*Proof.* Let $d = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ be the tree derived from $d^*$ by deleting a pair of leaves $p_i \to \hat{y}_i^{(\text{leaf})}, p_{i+1} \to \hat{y}_{i+1}^{(\text{leaf})}$ and adding the their parent leaf $p_j \to \hat{y}_j^{(\text{leaf})}$. The discrepancy between $d^*$ and $d$ is the discrepancy between $p_i, p_{i+1}$ and $p_j$:

$$\ell(d, \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y}) = a_i,$$

where $a_i$ is defined in (3.10). Therefore,

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda(K-1) &= \ell(d^*, \mathbf{x}, \mathbf{y}) + a_i + \lambda(K-1) \\ &= R(d^*, \mathbf{x}, \mathbf{y}) + a_i - \lambda \\ &= R^* + a_i - \lambda. \end{aligned} \tag{3.11}$$

This combined with $R^* \leq R(d, \mathbf{x}, \mathbf{y})$ leads to $\lambda \leq a_i$. □

Thus, when we split a leaf of the parent tree, if the incremental fraction of data that are correctly classified after this split is less than a fraction $\lambda$, we need to further split at least one of the two child leaves to search for the optimal tree. We can actually view Theorem 3 as a sub-condition of Theorem 4, and in our implementation, we are able to leverage both of them. This is because Theorem 3 provides a much cheaper check in terms of the computation, which we can make use of to prune the search space before we calculate the bound of Theorem 4. In our algorithm, we apply Theorem 3 when we split the leaves. We only need to split the leaves whose normalized supports are no less than $2\lambda$. And we apply Theorem 4 when we construct the trees. For every new split we do, we need to check the incremental accuracy corresponding to this split. If it is less than $\lambda$, we further split at least one of the two child leaves.

Both Theorem 3 and Theorem 4 are bounds for pair of leaves. We further give a bound on a single leaf's classification accuracy in Theorem 5.

**Theorem 5** (Lower bound on classification accuracy)**.** *Let $d^* = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ be any optimal tree with objective $R^*$, i.e., $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. For each leaf $(p_k, \hat{y}_k^{(leaf)})$ in $d^*$, the fraction of correctly classified data in leaf $k$ should be no less than $\lambda$,*

$$\lambda \leq \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(leaf)} = y_n]. \tag{3.12}$$

*Proof.* Let $d = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ be the tree derived from $d^*$ by deleting a pair of leaves $p_i \to \hat{y}_i^{(\text{leaf})}, p_{i+1} \to \hat{y}_{i+1}^{(\text{leaf})}$ and adding the their parent leaf $p_j \to \hat{y}_j^{(\text{leaf})}$. The discrepancy between $d^*$ and $d$ is the discrepancy between $p_i, p_{i+1}$ and $p_j$:

$$\ell(d, \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y}) = a_i,$$

where we defined $a_i$ in (3.10). According to Theorem 4,

$$\begin{aligned}
\lambda \leq & a_i, \\
\lambda \leq & \frac{1}{N}\sum_{n=1}^{N}\{\text{cap}(x_n, p_i) \wedge \mathbb{1}[\hat{y}_i^{(\text{leaf})} = y_n] + \text{cap}(x_n, p_{i+1}) \wedge \mathbb{1}[\hat{y}_{i+1}^{(\text{leaf})} = y_n] \\
& - \text{cap}(x_n, p_j) \wedge \mathbb{1}[\hat{y}_j^{(\text{leaf})} = y_n]\}.
\end{aligned} \tag{3.13}$$

For any leaf $j$ and its two child leaves $i, i+1$, we always have

$$\begin{aligned}
\sum_{n=1}^{N} \text{cap}(x_n, p_i) \wedge \mathbb{1}[\hat{y}_i^{(\text{leaf})} = y_n] &\leq \sum_{n=1}^{N} \text{cap}(x_n, p_j) \wedge \mathbb{1}[\hat{y}_j^{(\text{leaf})} = y_n], \\
\sum_{n=1}^{N} \text{cap}(x_n, p_{i+1}) \wedge \mathbb{1}[\hat{y}_{i+1}^{(\text{leaf})} = y_n] &\leq \sum_{n=1}^{N} \text{cap}(x_n, p_j) \wedge \mathbb{1}[\hat{y}_j^{(\text{leaf})} = y_n]
\end{aligned} \tag{3.14}$$

which indicates that $a_i \leq \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p_i) \wedge \mathbb{1}[\hat{y}_i^{(\text{leaf})} = y_n]$ and $a_i \leq \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p_{i+1}) \wedge \mathbb{1}[\hat{y}_{i+1}^{(\text{leaf})} = y_n]$. Therefore,

$$\begin{aligned}
\lambda &\leq \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p_i) \wedge \mathbb{1}[\hat{y}_i^{(\text{leaf})} = y_n], \\
\lambda &\leq \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p_{i+1}) \wedge \mathbb{1}[\hat{y}_{i+1}^{(\text{leaf})} = y_n].
\end{aligned} \tag{3.15}$$

□

Thus, in a leaf we are thinking about extending, if a feature does not have the minimum fraction of correctly classified data for points that go to that leaf, then we can exclude that feature further down the tree extending that leaf. The feature will be excluded anywhere below the leaf, not just in the split immediately being considered on the current leaf.

## 3.5 Permutation Bound

If two trees are composed of the same leaves, *i.e.*, they contain the same conjunctions of features up to a permutation, then they classify all the data in the same way and their child trees are also permutations of each other. Therefore, if we already have all children from one permutation of a tree, then there is no benefit to considering child trees generated from a different permutation.

**Corollary 6** (Leaf Permutation bound). *Let $\pi$ be any permutation of $\{1, \ldots, H\}$, Let $d = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ and $D = (D_{un}, \Delta_{un}, D_{split}, \Delta_{split}, K, H)$ denote trees* *with leaves* $(p_1, \ldots, p_H)$ *and* $D_{un} = (p_{\pi(1)}, \ldots, p_{\pi(H)})$*, respectively, i.e., the leaves in* $D$ *correspond to a permutation of the leaves in* $d$*. Then the objective lower bounds of* $d$ *and* $D$ *are the same and their child trees correspond to permutations of each other.*

*Proof.* This is intuitive because we can view a tree as a set of leaves and a leaf as a set of clauses, where the order of the leaves does not matter. □

Therefore, if two trees have the same leaves, up to a permutation, according to Corollary 6, either of them can be pruned. We call this symmetry-aware pruning.

## 3.6 Similar Support Bound

Here, we present the similar support bound to deal with highly similar features. With the similar support bound, given two trees are exactly the same but one internal node split by different similar enough features, if we know one of them and all its child trees cannot be the optimal tree, we can also prune the other one and all its child trees.

**Theorem 7** (Similar support bound)**.** *Define* $d = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ *and* $D = (D_{un}, \Delta_{un}, D_{split}, \Delta_{split}, K, H)$ *to be two trees which are exactly the same but one internal node split by different features. Let* $f_1, f_2$ *be the features used to split that node in* $d$ *and* $D$ *respectively. Let* $t_1, t_2$ *be the left subtree and the right subtree under the node* $f_1$ *in* $d$*, and let* $T_1, T_2$ *be the left subtree and the right subtree under the node* $f_2$ *in* $D$*. Denote the normalized support of data captured by only one of* $t_1$ *and* $T_1$ *as* $\omega$*, i.e.,*

$$\omega \equiv \frac{1}{N} \sum_{n=1}^{N} [\neg \operatorname{cap}(x_n, t_1) \wedge \operatorname{cap}(x_n, T_1) + \operatorname{cap}(x_n, t_1) \wedge \neg \operatorname{cap}(x_n, T_1)]. \tag{3.16}$$

*The difference between the two trees' objectives is bounded by* $\omega$ *as the following:*

$$\omega \geq R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \geq -\omega, \tag{3.17}$$

*where* $R(d, \mathbf{x}, \mathbf{y})$ *is objective of* $d$ *and* $R(D, \mathbf{x}, \mathbf{y})$ *is the objective of* $D$*. If* $f_1, f_2$ *are highly similar features such that we cannot split using* $f_2(f_1)$ *under node split by* $f_1(f_2)$ *according to Theorem 5, then we can guarantee that:*

$$\{\textit{all possible } t_1's\} = \{\textit{all possible } T_1's\}, \textit{and } \{\textit{all possible } t_2's\} = \{\textit{all possible } T_2's\}$$

*Therefore, we have*

$$\omega \geq \min_{d^\dagger \in \sigma(d_{un})} R(d^\dagger, \mathbf{x}, \mathbf{y}) - \min_{D^\dagger \in \sigma(D_{un})} R(D^\dagger, \mathbf{x}, \mathbf{y}) \geq -\omega. \tag{3.18}$$

*Proof.* The difference between the objectives of $d$ and $D$ maximizes when one of them correctly classifies all the data corresponding to $\omega$ but the other misclassifies all of them. Therefore,

$$\omega \geq R(d, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) \geq -\omega$$

$\square$

With the similar support bound, for two trees $d$ and $D$ as above, if we already know $d$ and all its child tree cannot achieve a better tree than the current optimal one and $\min_{d^\dagger \in \sigma(d_{un})} R(d^\dagger, \mathbf{x}, \mathbf{y}) \geq R^c + \omega$, we can also prune the $D$ and all its child trees, because

$$\min_{D^\dagger \in \sigma(D_{un})} R(D^\dagger, \mathbf{x}, \mathbf{y}) \geq \min_{d^\dagger \in \sigma(d_{un})} R(d^\dagger, \mathbf{x}, \mathbf{y}) - \omega \geq R^c. \tag{3.19}$$

## 3.7 Equivalent Points Bound

Equivalent Points Bound applies when multiple observations captured by a leaf in $d_{\text{split}}$ have exactly the same features but different labels: then no tree, including those that extend $d_{\text{split}}$, can correctly classify all these observations. In this case, the number of misclassifications made by any tree is no less than the number of observations with the minority label.

For a data set $\{(x_n, y_n)\}_{n=1}^N$ and also a set of features $\{s_m\}_{m=1}^M$, we define a set of samples to be equivalent if they have exactly the same feature values, *i.e.*, $x_i \neq x_j$ are equivalent if

$$\frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[\text{cap}(x_i, s_m) = \text{cap}(x_j, s_m)] = 1.$$

Note that a data set consists of multiple sets of equivalent points; let $\{e_u\}_{u=1}^U$ enumerate these sets. For each observation $x_i$, it belongs to a equivalent points set $e_u$. We denote the fraction of data with the minority label in set $e_u$ as $\theta(e_u)$, *e.g.*, let $e_u = \{x_n : \forall m \in [M], \mathbb{1}[\text{cap}(x_n, s_m) = \text{cap}(x_i, s_m)]\}$, and let $q_u$ be the minority class label among points in $e_u$, then

$$\theta(e_u) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[x_n \in e_u]\, \mathbb{1}[y_n = q_u]. \tag{3.20}$$

We can combine the equivalent points bound with other bounds to get a tighter lower bound on the objective function. As the experimental results demonstrate in §6, there is sometimes a substantial reduction of the search space after incorporating the equivalent points bound. We propose a general equivalent points bound in Proposition 8. We incorporate it into our framework by proposing the specific equivalent points bound in Theorem 9.

**Proposition 8** (General equivalent points bound)**.** *Let* $d = (d_{un}, \delta_{un}, d_{split}, \delta_{split}, K, H)$ *be a tree, then*

$$R(d, \mathbf{x}, \mathbf{y}) \geq \sum_{u=1}^{U} \theta(e_u) + \lambda H.$$

*Proof.* This bound adapts directly form [ALSA+18], where the proof could be found. □

Recall that in our lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ in (3.4), we leave out the misclassification errors of leaves we are going to split $\ell_0(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y})$ from the objective $R(d, \mathbf{x}, \mathbf{y})$. Incorporating the equivalent points bound in Theorem 9, we obtain a tighter bound on our objective because we now have a tighter lower bound on the misclassification errors of leaves we are going to split, $0 \leq b_0(d_{\text{split}}, \mathbf{x}, \mathbf{y}) \leq \ell_0(d_{\text{split}}, \delta_{\text{split}}, \mathbf{x}, \mathbf{y})$.

**Theorem 9** (Equivalent points bound)**.** *Let $d$ be a tree with leaves $d_{un}, d_{split}$ and lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$, then for any tree $d' \in \sigma(d)$ whose prefix $d'_{un} \supseteq d_{un}$,*

$$R(d', \mathbf{x}, \mathbf{y}) \geq b(d_{un}, \mathbf{x}, \mathbf{y}) + b_0(d_{split}, \mathbf{x}, \mathbf{y}), \quad \textit{where} \tag{3.21}$$

$$b_0(d_{split}, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \text{cap}(x_n, d_{split}) \wedge \mathbb{1}[x_n \in e_u] \, \mathbb{1}[y_n = q_u]. \tag{3.22}$$

*Proof.* This bound adapts directly form [ALSA+18], where the proof could be found. □

# Chapter 4

# Incremental Computation

Much of our implementation effort revolves around exploiting incremental computation. During the execution of Algorithm 1, for each tree $d$, we compute the lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ of the tree based on its unchanged leaves $d_{un}$ and the corresponding objective $R(d, \mathbf{x}, \mathbf{y})$ of the tree. Given the hierarchical nature of the parent-children relationship, we *incrementally* compute the objective function and the lower bound throughout the brand-and-bound execution of the algorithm. Together, these ideas save >97% execution time. We provide the details of the incremental computation in this section. And in the next section §5, we further provide the data structures we implement in practice to cache those values for the incremental computation.

For the hierarchical branch-and-bound execution, we start from an empty tree $d = ((), (), (), \delta_{\text{split}}, 0, 0)$, which is the first tree evaluated in the algorithm. For the empty tree, it does not contain any unchanged leaves and has a length of zero. Thus, its lower bound is $b((), \mathbf{x}, \mathbf{y}) = 0$. Since all the data are captured by the empty leaf, the objective of the empty tree is the proportion of minority class data, *i.e.*, $R(d, \mathbf{x}, \mathbf{y}) = \ell_q((), \delta_{\text{split}}, \mathbf{x}, \mathbf{y})$.

Based on the initial objective and the lower bound we get from the first tree, *i.e.*, the empty tree, we further derive their incremental expressions. Let $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H)$ and $d' = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$ be decision trees such that $d'$ is a child of $d$ and $d_{un} = (p_1, \ldots, p_K) \subset d'_{un} = (p_1, \ldots, p_K, \ldots, p_{K'})$. Let $\delta_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_H^{(\text{leaf})})$ and $\delta'_{un} = (\hat{y}_1^{(\text{leaf})}, \ldots, \hat{y}_{H'}^{(\text{leaf})})$ be the corresponding labels. It follows from the hierarchical structure of Algorithm 1 that if we are evaluating the child $d'$, both the objective and the lower bound of its parent $d$ have already been calculated. Thus, in the evaluation of $d'$, we reuse these computations in our algorithm. Now, based on the objective lower bound of its parent $d$, we calculate the objective lower bound of $d'$ incrementally as the following:

$$\begin{aligned} b(d'_{un}, \mathbf{x}, \mathbf{y}) &= \ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) + \lambda(H') \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] + \lambda H' \\ &= \ell_p(d_{un}, \delta_{un}, \mathbf{x}, \mathbf{y}) + \lambda H' + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n] \\ &= b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda(H' - H) + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]. \end{aligned} \tag{4.1}$$

Therefore, when computing $b(d'_{un}, \mathbf{x}, \mathbf{y})$, we can reuse the quantity of the lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ if we cache it in a data structure. Moreover, in practice, we cache the misclassification error of each leaf, because we can view leaves as set of clauses which we can reuse in different trees. In the same way, based on the lower bound, we express the objective of $d'$ incrementally:

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell_p(d'_{un}, \delta'_{un}, \mathbf{x}, \mathbf{y}) + \ell_0(d'_{\text{split}}, \delta'_{\text{split}}, \mathbf{x}, \mathbf{y}) + \lambda H' \\ &= b(d'_{un}, \mathbf{x}, \mathbf{y}) + \ell_0(d'_{\text{split}}, \delta'_{\text{split}}, \mathbf{x}, \mathbf{y}) \\ &= b(d'_{un}, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K'+1}^{H'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]. \end{aligned} \tag{4.2}$$

The incremental computation in (4.2) reuses $b(d'_{un}, \mathbf{x}, \mathbf{y})$, which we have already computed in (4.1). Note that we could alternatively compute $R(d', \mathbf{x}, \mathbf{y})$ incrementally from $R(d, \mathbf{x}, \mathbf{y})$. However, doing so is a somewhat more complex, because we have to subtract the misclassification error of those split leaves of $d$ from $R(d, \mathbf{x}, \mathbf{y})$ and then add that of those new leaves of $d'$ to $R(d, \mathbf{x}, \mathbf{y})$.

In Algorithm 2, we present our algorithm incrementally, with both the objective value (4.2) and the lower bound (4.1) computed incrementally. We also present how other bounds are incorporate into our algorithm. In addition, compared with Algorithm 1, in Algorithm 2 we apply the hierarchical objective lower bound from Theorem 1 to child trees instead of the parent tree. This could shrink the size of the queue since we do not need to push those trees which do not satisfy Theorem 1 into the queue. And therefore, this could further save the execution time especially when we use priority queue to order entries as described in section §5.

**Algorithm 2** Incremental branch-and-bound for learning optimal decision trees.

```
**Input:** Objective function $R(d, \mathbf{x}, \mathbf{y})$, objective lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$, set of features $S = \{s_m\}_{m=1}^{M}$, training data $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^{N}$, regularization parameter $\lambda$
**Output:** Provably optimal decision tree $d^*$ with minimum objective $R^*$

$d^c \leftarrow ((), (), (), \delta_{\text{split}}, 0, 0)$    ▷ For simplicity, cold start with an empty tree
$R^c \leftarrow R(d^c, \mathbf{x}, \mathbf{y})$    ▷ Initialize current best objective
$Q \leftarrow \text{queue}(\,[d^c]\,)$    ▷ Initialize queue with the empty tree
while $Q$ not empty do    ▷ Optimization complete when the queue is empty
    $d = (d_{un}, \delta_{un}, d_{\text{split}}, \delta_{\text{split}}, K, H) \leftarrow Q.\text{pop}(\,)$    ▷ Remove tree $d$ from the queue
    for every possible combination of features to split $d_{\text{split}}$ do
        split $d_{\text{split}}$ and get new leaves $d_{new}$
        CheckLeafSupportAndAccuracyBound($d_{\text{new}}$)
        for each possible subset $d'_{\text{split}}$ of $d_{new}$ do
            $d'_{un} = d_{un} \cup (d_{\text{new}} \setminus d'_{\text{split}})$
            $d' = (d'_{un}, \delta'_{un}, d'_{\text{split}}, \delta'_{\text{split}}, K', H')$    ▷ Branch: Generate child tree $d'$
            $\Delta_{un} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=K+1}^{K'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$
            $b(d'_{un}, \mathbf{x}, \mathbf{y}) \leftarrow b(d_{un}, \mathbf{x}, \mathbf{y}) + \lambda(H' - H) + \Delta_{un}$
            $\Delta_R = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=K'+1}^{H'} \text{cap}(x_n, p_k) \wedge \mathbb{1}[\hat{y}_k^{(\text{leaf})} \neq y_n]$
            $R(d', \mathbf{x}, \mathbf{y}) \leftarrow b(d'_{un}, \mathbf{x}, \mathbf{y}) + \Delta_R$
            if $R(d', \mathbf{x}, \mathbf{y}) < R^c$ then
                $(d^c, R^c) \leftarrow (d', R(d', \mathbf{x}, \mathbf{y}))$    ▷ Update $d^c$ and $R^c$
            end if
            if $b(d'_{un}, \mathbf{x}, \mathbf{y}) + \sum_{l \in d'_{\text{split}}} b_0(l, \mathbf{x}, \mathbf{y}) + \lambda \times L(d'_{\text{split}}) < R^c$ then
                    ▷ Bound: Apply bound from Theorem 1, combined with
                    Equivalent points bound (Theorem 9)
                    and Lookahead bound (Lemma 2)
                $Q.\text{push}(d')$    ▷ Add $d'$ to the queue
            end if
        end for
    end for
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$    ▷ Identify provably optimal tree and objective
```

**Algorithm 3** Function for checking lower bounds on leaf support and classification accuracy (Section 3.4)

```
$d_{\text{new}}$ is the set of new leaves we get from spliting $d_{\text{split}}$

function CHECKLEAFSUPPORTANDACCURACYBOUND($d_{\text{new}}$)
    for each new leaf $l$ in $d_{\text{new}}$ do
        if The misclassified data in leaf $l$ is less than a fraction of $\lambda$ then
            We no longer split this leaf in any tree    ▷ Lower bound on leaf support (Theorem3)
        end if
        if Leaf $l$ does not correctly classify at least $\lambda$ samples then
            We exclude the corresponding feature we just used to split the old leaf and get the leaf $l$ further down the tree extending that leaf    ▷ Lower bound on classification accuracy (Theorem5)
        end if
    end for
end function
```

# Chapter 5

# Implementation

We implement a series of data structures designed to support this incremental computation.

## 5.1 Data Structure of Leaf and Tree

First, we store bounds and intermediate results for the trees and the leaves to support incremental computation of the lower bound and the objective. As a global statistic, we maintain the best (minimum) observed value of the objective function and the corresponding tree. As each leaf in a tree represents a set of clauses, for each leaf we store, in an efficient bit-vector representation, the set of samples captured by that clause and the accuracy of those samples with respect to the prediction of the leaf. From the values in the leaves, we can efficiently compute both the value of the objective for an entire tree and leaf values for children formed from splitting the leaf.

Specifically, For the data structure of leaf $l$, we store:

- A set of clauses in the leaf.
- A binary vector of length $N$ (number of data) indicating whether or not each point is captured by the leaf.
- Number of points captured by the leaf.
- A binary vector of length $M$ (number of features) indicating whether or not each feature is dead. In a leaf, a feature is dead if Theorem 5 does not hold. And for any new leaf which contain all these clauses, the dead feature would still keep dead.
- The lower bound on the leaf misclassification error $b_0(l, \mathbf{x}, \mathbf{y})$, which is defined in (3.22)
- The label of the leaf.
- The loss of the leaf.
- A binary scalar indicating whether or not this leaf is dead. A leaf is dead if Theorem 3 does not hold and we would never further split a dead leaf.

And for the tree structure, in addition to cache the tree's leaves, we store additional information of it. For the data structure of tree, we store:

- A set of leaves in the tree.
- The objective.
- The lower bound of the objective.
- A binary vector of length $n_l$ (number of leaves) indicating whether or not each point is going to be split, *i.e.*, split leaves $d_{\text{split}}$ and unchanged leaves $d_{un}$. The unchanged leaves of a tree would still be unchanged leaves in its child trees.

## 5.2 Queue

Second, we use a priority queue to order the exploration of the search space. The queue serves as a worklist with each entry in the queue corresponding to a tree. When an entry is removed from the queue, we use it to generate child trees, incrementally computing the information for the child trees. The ordering of the worklist represents a scheduling policy. We evaluated both structural orderings, e.g., breadth first search and depth first search, and metric-based orderings, e.g., objective function, the lower bound. Each metric produces a different scheduling policy. We achieve the best performance in runtime and memory consumption using the curiosity metric from CORELS [ALSA+18], which is the objective lower bound, divided by the normalized support of its unchanged leaves. For example, relative to using the objective, curiosity reduces runtime by a factor of two and memory consumption by a factor of four.

## 5.3 Symmetry-aware Map

Third, to support symmetry-aware pruning from Corollary 6, we introduce two symmetry aware maps – a LeafCache and a TreeCache. A leaf is a set of clauses, each of which corresponds to an attribute and the value (0,1) of that attribute. As the leaves of a decision tree are mutually exclusive, the data captured by each leaf is insensitive to the order of the leaf's clauses. We encode leaves in a canonical order (sorted by attribute indexes) and use that canonical order as the key into the LeafCache. Each entry in the LeafCache represents all permutations of a set of clauses. We use a Python dictionary to map these keys to the leaf and its cached values. Before creating a leaf object, we first check whether or not we have already created an identical one. If we have not, then we create the leaf and insert it into the map. Otherwise, the permutation already exists, so instead of creating it, we retrieve it from the map and use it as the leaf of the tree we are constructing. Next, we implement the permutation bound (Corollary 6) using the TreeCache. The TreeCache contains encodings of all the trees we have evaluated. Like we did for the clauses in the LeafCache, we introduce a canonical order over the leaves in a tree and use that as the key to the

TreeCache. If our algorithm produces a tree that is simply a permutation of a tree we have already evaluated, then we need not evaluate it again. We determine that this is the case by looking up the tree under consideration in the TreeCache. If it's in the cache, we do nothing; if it is not, then we compute the bounds for the tree and insert it into the cache.

## 5.4 Execution

Now, we illustrate how these data structures support execution of our algorithm. We initialize the algorithm with the current best objective $R^c$ and tree $d^c$. For unexplored trees in the queue, the scheduling policy selects the next tree $d$ to split; we keep removing elements from the queue until the queue is empty. Then, for every very possible combination of features to split $d_{\text{split}}$, we construct a new tree $d'$ with incremental calculation of the lower bound $b(d'_{un}, \mathbf{x}, \mathbf{y})$ and the objective $R(d', \mathbf{x}, \mathbf{y})$. If we achieve a better objective $R(d', \mathbf{x}, \mathbf{y})$, *i.e.*, less than the current best objective $R^c$, we update $R^c$ and $d^c$. If the lower bound of the new tree $d'$, combined with the equivalent points bound (Theorem 9) and the lookahead bound (Theorem 2), is less than the current best objective, then we push it into the queue. Otherwise, according to the hierarchical lower bound (Theorem 1), no child of $d'$ could possibly have an objective better than $R^c$, which means we do not push $d'$ queue. When there are no more trees to explore, *i.e.*, the queue is empty, we have finished the search of the whole space and output the (provably) optimal tree.

# Chapter 6

# Experiments

We address the following questions through experimental analysis:

(1) *Do existing methods achieve optimal solutions, and if not, how far are they from optimal?*

(2) *How fast does our algorithm converge given the hardness of the problem it is solving?*

(3) *How much does each of the bounds contribute to the performance of our algorithm?*

(4) *What do optimal trees look like?*

The results of per-bound performance and memory improvement (Table 6.2) were run on a *m5a.4xlarge* instance of AWS's Elastic Compute Cloud (EC2). The instance has 16 2.5GHz virtual GPU cores and 64 GB of RAM. All other results were run on a personal laptop with a 2.4GHz i5-8259U processor and 16GB of RAM.

We used 7 datasets: Five of them are from the UCI Machine Learning Repository [DKT17], (tic-tac-toc, car evaluation, monk1, monk2, monk3). The other two datasets are the ProPublica recidivism data set [LMKA16] and the Fair Isaac (FICO) credit risk datasets [FGL+18]. We predict which individuals are arrested within two years of release ($N = 7,215$) on the recidivism data set and whether an individual will default on a loan for the FICO dataset. Table 6.1 shows the sample size and the number of features of these datasets .

*Accuracy and optimality:* We tested the accuracy of our algorithm against baseline methods CART and BinOCT [VZ19]. BinOCT is the most recent publically available method for learning optimal classification trees and was shown to outperform other previous methods. As far as we know, there is no public code for most of the other

Datasets used in the experiment

| Dataset | Sample size | Number of features |
|---|---|---|
| tic-tac-toc | 958 | 18 |
| car evaluation | 1728 | 15 |
| monk1 | 124 | 11 |
| monk2 | 169 | 11 |
| monk3 | 122 | 11 |
| FICO | 10,459 | 17 |
| ProPublica | 6,907 | 12 |

**Table 6.1**: The sample size and the number of features of the datasets.

relevant baselines, including [BD17, MGKS18, BCMRM18]. One of these methods, OCT [BD17], reports that CART often dominates their performance (see Fig. 4 and Fig. 5 in their paper). Our models can never be worse than CART's models, because in our implementation, we use the objective value of CART's solution as a warm start to the objective value of the current best.

Figure 6.1 shows the training accuracy on each dataset. The time limits for both BinOCT and our algorithm are set to be 30 minutes.

*Main results:* We observe that (i) We can now evaluate how close to optimal other methods are (and they are often close to optimal or optimal). (ii) Sometimes, the baselines are *not* optimal. Recall that BinOCT searches only for the optimal tree *given the topology of the complete binary tree of a certain depth.* This restriction on the topology massively reduces the search space so that BinOCT runs quickly, but in exchange, it misses optimal sparse solutions that our method finds. (iii) Our method is *fast.* Our method runs on only one thread (we have not yet parallelized it) whereas BinOCT is highly optimized; it makes use of all eight threads in our experiments. Even with the $\approx$ 8x speedup of BinOCT, our method is still competitive.

*Convergence:* Figure 6.2 illustrates the behavior of OSDT for the ProPublica COMPAS dataset with $\lambda = 0.005$, for two different scheduling policies (curiosity and lower bound). The charts in the figure show how the current best objective value $R^c$

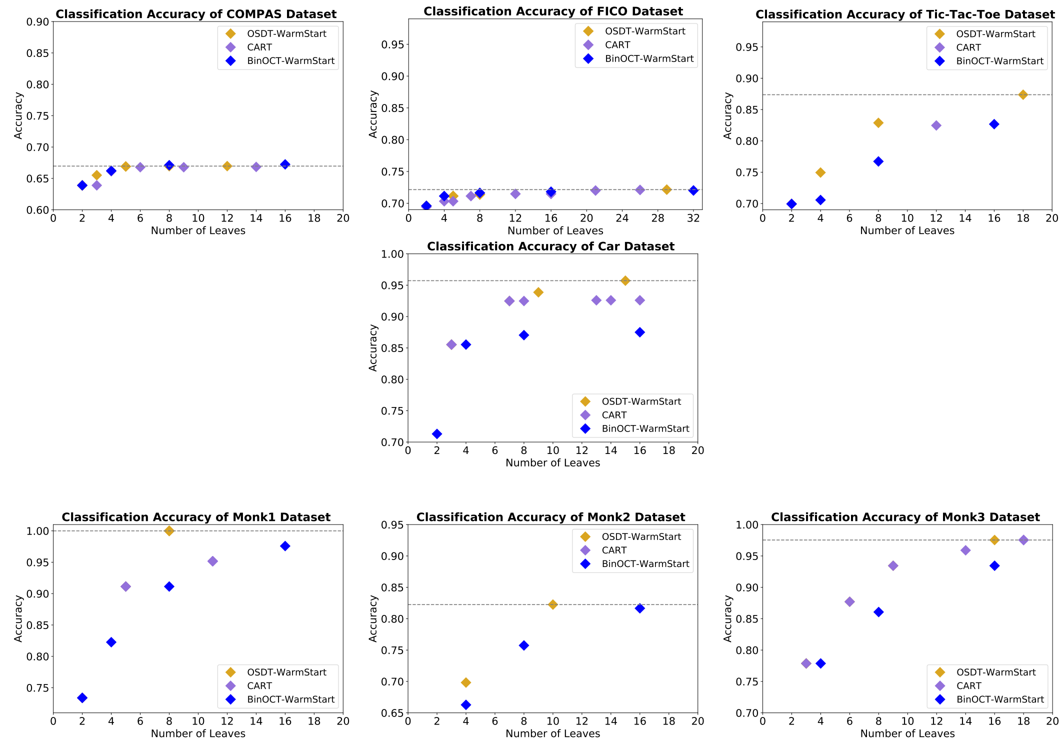


**Figure 6.1**: Training accuracy of OSDT, CART, BinOCT on different data (time limit: 30min). Horizontal lines indicate the accuracy of the best OSDT tree. On most datasets, all trees of BinOCT and CART are below this line.

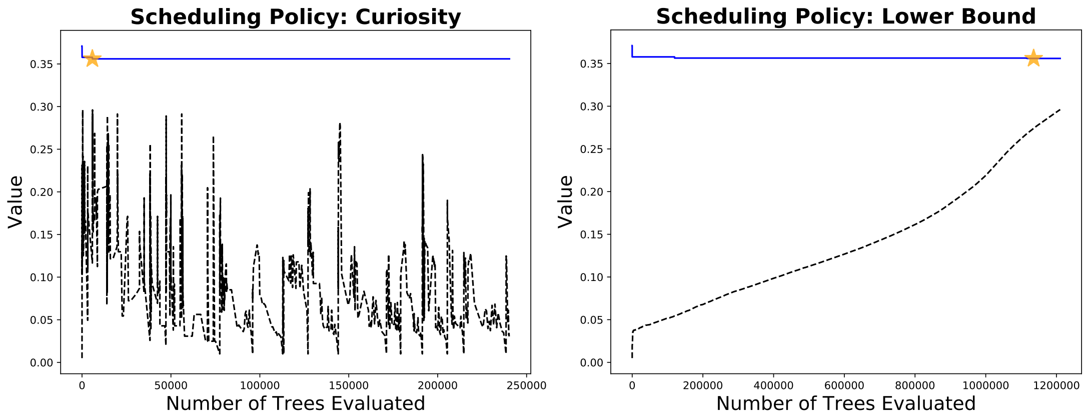


**Figure 6.2**: Example OSDT execution traces (COMPAS Dataset, $\lambda = 0.005$). Lines are the objective value and dashes are the lower bound for OSDT. For each scheduling policy, we mark the time to optimum and the optimal objective value using a star.

and the lower bound $b(d_{un}, \mathbf{x}, \mathbf{y})$ vary as the algorithm progresses. When we schedule using the lower bound, the lower bounds of evaluated trees increase monotonically, and OSDT certifies optimality only when the value of the lower bound becomes large enough that we can prune the remaining search space or when the queue is empty, whichever is reached earlier. Using the curiosity, OSDT finds the optimal tree much more quickly than when using the lower bound.

*Scalability:* Figure 6.3 shows the scalability of OSDT with respect to the number of samples and the number of features. Running time grows *linearly* in the number of samples ($n$), allowing us to scale to millions of samples. With a small number of features (i.e., 4-8) runtime grows exponentially, however, as we add more features, the growth rate slows down, because we are able to prune the search space more quickly as the number of features grows. For example, it takes us half the runtime to find the optimal tree using four features, but only one quarter of the run time to find the optimal tree with twelve features.

*Algorithm optimizations:* Next, we evaluate how much each of our bounds contributes to OSDT's performance and what effect the scheduling metric has on execution. Table 6.2 provides experimental statistics of total execution time, time to optimum, total number of trees evaluated, number of trees evaluated to optimum, and memory consumption on the recidivism data set. The first row is the full OSDT implementation, and the others are variants each of which removes a specific bound. While all the optimizations reduce the search space, the lookahead and equivalent points bounds are, by far, the most significant, reducing time to optimum by at least two orders of magnitude and reducing memory consumption by more than one order of magnitude. In our experiment, although the scheduling policy has a smaller effect, it is still significant – curiosity is a factor of two faster than the objective function and consumes 25% of the memory consumed when using the objective function. All other scheduling policies, *i.e.*, the lower bound and the entropy, are significantly worse.

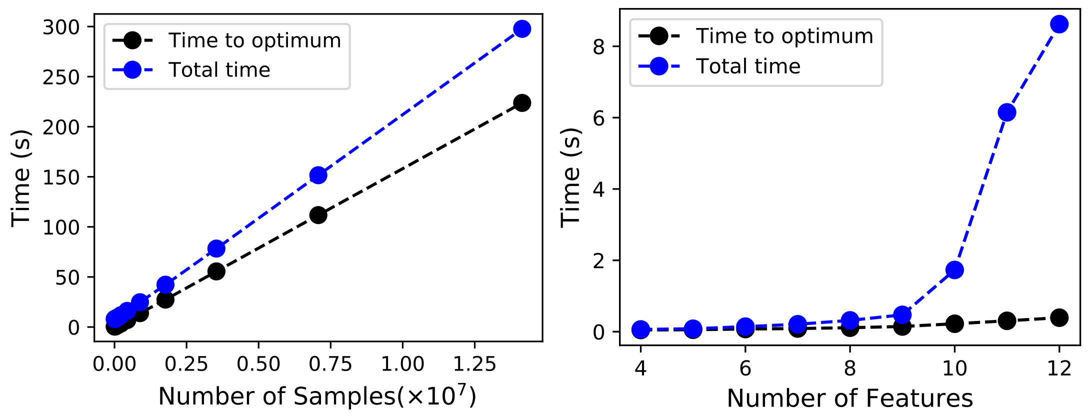


**Figure 6.3**: Scalability with respect to number of samples and number of features using (multiples of) the ProPublica data set. ($\lambda = 0.005$). Note that all these executions include the 4 features of the optimal tree (Figure 6.4), and the data size are increased by duplicating the whole data set multiple times.

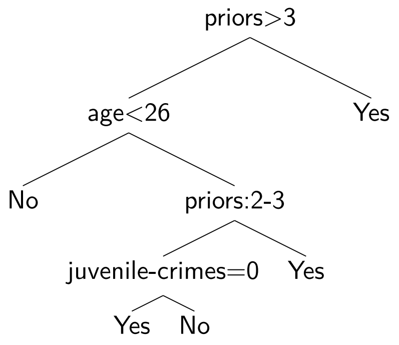


**Figure 6.4**: The optimal decision tree generated by OSDT on COMPAS dataset. ($\lambda = 0.005$)

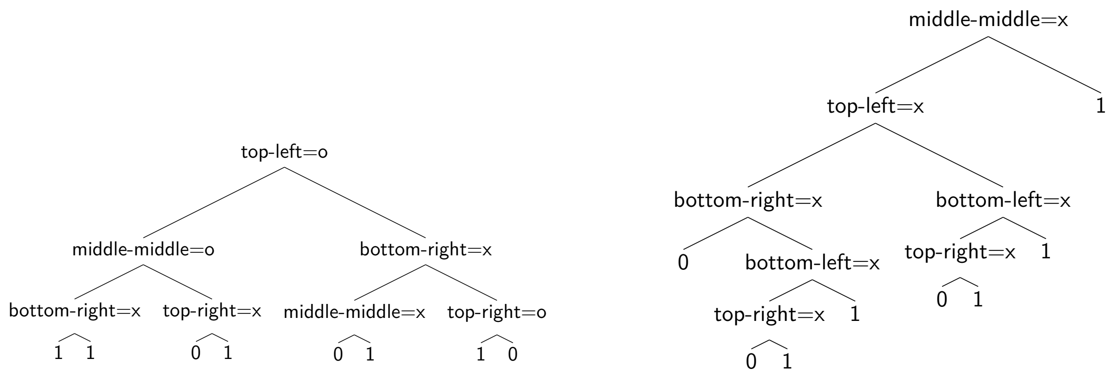
*Trees:* Finally, we provide illustrations of the trees produced by OSDT and the baseline methods in Figures 6.4, 6.5 and 6.6. OSDT genereates trees of any shape, and our objective penalizes trees with more leaves, thus it never introduces splits that produce a pair of leaves with the same label. However, the trees generated by BinOCT are always complete binary trees of given depth. Limiting the tree shape prevents BinOCT from finding the globally optimal tree in most cases. In fact, BinOCT sometimes produces useless splits, which leads to a trees with more leaves than necessary to achieve the same accuracy.

Per-bound performance improvement (ProPublica data set)

| Algorithm variant | Total time (s) | Slow-down | Time to optimum (s) |
|---|---|---|---|
| All bounds | 14.75 | — | 1.09 |
| No support bound | 16.69 | 1.13× | 1.18 |
| No incremental accuracy bound | 29.42 | 1.99× | 1.27 |
| No accuracy bound | 31.72 | 2.15× | 1.44 |
| No lookahead bound | 31479 | 2134× | 186 |
| No equivalent points bound | >8226 | >557× | — |

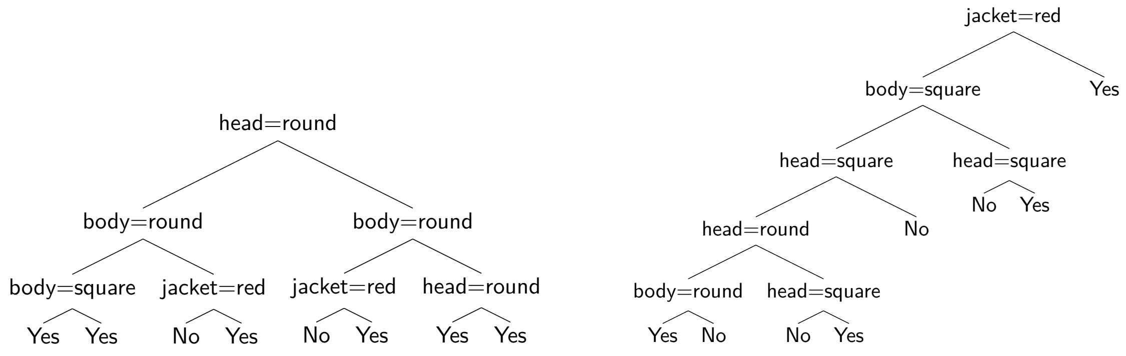
| Algorithm variant | Total #trees evaluated | #trees to optimum | Mem (GB) |
|---|---|---|---|
| All bounds | 241306 | 18195 | .08 |
| No support bound | 278868 | 21232 | .08 |
| No incremental accuracy bound | 548618 | 24682 | .08 |
| No accuracy bound | 482013 | 23075 | .09 |
| No lookahead bound | 283712499 | 3078445 | 10 |
| No equivalent points bound | >41000000 | — | >64 |

**Table 6.2**: Per-bound performance improvement, for the ProPublica data set ($\lambda = 0.005$, cold start, using curiosity). The columns report the total execution time, time to optimum, total number of trees evaluated, number of trees evaluated to optimum, and memory consumption. The first row shows our algorithm with all bounds; subsequent rows show variants that each remove a specific bound (one bound at a time, not cumulative). All rows except the last one represent a complete execution, *i.e.*, until the queue is empty. For the last row ('No equivalent points bound'), the algorithm was terminated after running out of the memory about ~64GB RAM.



(a) BinOCT (accuracy: 76.722%) (b) OSDT (accuracy: 82.881%)

**Figure 6.5**: The decision tree generated by BinOCT and OSDT on the Tic-Tac-Toe data. Trees of BinOCT must be complete binary trees, while OSDT can generate trees of any shape.



(a) BinOCT (accuracy: 91.129%) (b) OSDT (accuracy: 100%)

**Figure 6.6**: The decision tree generated by BinOCT and OSDT on Monk1 dataset. The tree generated by BinOCT includes useless splits, while OSDT can avoid this problem.

# Chapter 7

# Conclusion and Future Work

Our work shows the possibility of optimal (or provably near-optimal) decision trees for reasonably sized datasets. We have reason to believe this basic framework can be extended to much larger datasets with some more work. In Theorem 7, we identified a key mechanism for scaling these algorithms up, which is the possibility of a bound stating that highly correlated features can substitute for each other, leading to similar model accuracies. This bound would work ideally when we can evaluate the tree and all its child trees before we evaluate its similar tree and all its child trees. However, in our queue, two similar trees would be pushed out at almost the same time. We have tried to put this bound into OSDT, but the amount of time spent evaluating the bound increased computation rather than reduce it. Further work needs to be done to determine when the bound should be computed in order to be effective, but if it can be done, the algorithm could potentially scale to much larger scales.

# Bibliography

[ALSA$^+$17] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists for categorical data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[ALSA$^+$18] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78, 2018.

[BB96] K. P. Bennett and J. A. Blue. Optimal decision trees. Technical report, R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute, 1996.

[BCMRM18] Rafael Blanquero, Emilio Carrizosa, Cristina Molero-Rıo, and Dolores Romero Morales. Optimal randomized classification trees. August 2018.

[BD17] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[Ben92] Kristin Bennett. Decision tree construction via linear programming. In *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference, Utica, Illinois*, 1992.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[CLR$^+$18] Chaofan Chen, Kancheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. An interpretable model with globally consistent explanations for credit risk. In *NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy*, 2018.

[DKT17] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[FGL$^+$18] FICO, Google, Imperial College London, MIT, University of Oxford, UC Irvine, and UC Berkeley. Fico explainable machine learning challenge, 2018.

[FLB16] Anthony W. Flores, Christopher T. Lowenkamp, and Kristin Bechtel. False positives, false negatives, and false analyses: A rejoinder to "Machine bias: There's software used across the country to predict future criminals". *Federal probation*, 80(2), September 2016.

[KS06] Adam R. Klivans and Rocco A. Servedio. Toward attribute efficient learning of decision lists and parities. *Journal of Machine Learning Research*, 7:587–602, 2006.

[LMKA16] J. Larson, S. Mattu, L. Kirchner, and J. Angwin. How we analyzed the COMPAS recidivism algorithm. *ProPublica*, 2016.

[LRMM15] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[LSAA$^+$18] N. Larus-Stone, E. Angelino, D. Alabi, M. Seltzer, V. Kaxiras, A. Saligrama, and C. Rudin. Systems optimizations for learning certifiably optimal rule lists. In *SysML Conference*, 2018.

[McG18] Michael McGough. How bad is sacramento's air, exactly? google results appear at odds with reality, some say. *Sacramento Bee*, 2018.

[MGKS18] Matt Menickelly, Oktay Günlük, Jayant Kalagnanam, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Preprint at arXiv:1612.03225*, January 2018.

[NF07] Siegfried Nijssen and Elisa Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539. ACM, 2007.

[NIP$^+$18] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, Joao Marques-Silva, and ISDCT SB RAS. Learning optimal decision trees with sat. In *IJCAI*, pages 1362–1368, 2018.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[VZ19] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *33rd AAAI Conference on Artificial Intelligence*, 2019.

[YRS17] H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. In *International Conference on Machine Learning (ICML)*, 2017.

[ZBL$^+$18] John R. Zech, Marcus A Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric K. Oermann. Confounding variables can degrade generalization performance of radiological deep models. *arXiv:1807.00431*, 2018.