Fast Sparse Decision Tree Optimization via Reference Ensembles

Hayden McTavish[1,3*], **Chudi Zhong**[2*], **Reto Achermann**[1], **Ilias Karimalis**[1], **Jacques Chen**[1], **Cynthia Rudin**[2], **Margo Seltzer**[1]

[1] University of British Columbia
[2] Duke University
[3] University of California, San Diego
hmctavish@ucsd.edu, {chudi.zhong, cynthia.rudin}@duke.edu, {achreto, mseltzer}@cs.ubc.ca, iliaskar@students.cs.ubc.ca, jacquesc@student.ubc.ca

Abstract

Sparse decision tree optimization has been one of the most fundamental problems in AI since its inception and is a challenge at the core of interpretable machine learning. Sparse decision tree optimization is computationally hard, and despite steady effort since the 1960's, breakthroughs have been made on the problem only within the past few years, primarily on the problem of finding optimal sparse decision trees. However, current state-of-the-art algorithms often require impractical amounts of computation time and memory to find optimal or near-optimal trees for some real-world datasets, particularly those having several continuous-valued features. Given that the search spaces of these decision tree optimization problems are massive, can we practically hope to find a sparse decision tree that competes in accuracy with a black box machine learning model? We address this problem via smart guessing strategies that can be applied to any optimal branch-and-bound-based decision tree algorithm. The guesses come from knowledge gleaned from black box models. We show that by using these guesses, we can reduce the run time by multiple orders of magnitude while providing bounds on how far the resulting trees can deviate from the black box's accuracy and expressive power. Our approach enables guesses about how to bin continuous features, the size of the tree, and lower bounds on the error for the optimal decision tree. Our experiments show that in many cases we can rapidly construct sparse decision trees that match the accuracy of black box models. To summarize: *when you are having trouble optimizing, just guess.*

1 Introduction

Decision trees are one of the leading forms of interpretable AI models. Since the development of the first decision tree algorithm (Morgan and Sonquist 1963), a huge number of algorithms have been proposed to improve both accuracy and run time. However, major approaches are based on decision tree induction, using heuristic splitting and pruning (e.g., Breiman et al. 1984; Quinlan 1993). Growing a tree in a greedy way, though fast, leads to suboptimal models with no indication of how far away the solution is from optimality. The generated trees are usually much more complicated than they need to be, hindering interpretability. Optimizing sparse decision trees remains one of the most fundamental problems in machine learning (ML).

Full decision tree optimization is NP-hard (Laurent and Rivest 1976), leading to challenges in searching for optimal trees in a reasonable time, even for small datasets. Major advances have been made recently using either mathematical programming solvers (e.g., Verwer and Zhang 2019) or customized branch-and-bound with dynamic programming (Aglin, Nijssen, and Schaus 2020; Lin et al. 2020; Demirović et al. 2020), showing us that there is hope. However, these methods are frequently unable to find the optimal tree within a reasonable amount of time, or even if they do find the optimal solution, it can take a long time to *prove* that the tree is optimal or close-to-optimal.

Ideally, we would like an algorithm that, within a few minutes, produces a sparse decision tree that is as accurate as a black box machine learning model. Also, we wish to have a guarantee that the model will have performance close to that of the black box. We present a practical way to achieve this by introducing a set of smart guessing techniques that speed up sparse decision tree computations for branch-and-bound methods by orders of magnitude.

The key is to guess in a way that prunes the search space without eliminating optimal and near-optimal solutions. We derive those smart guesses from a black box tree-ensemble reference model whose performance we aim to match. Our guesses come in three flavors. The first type of guess reduces the number of thresholds we consider as a possible split on a continuous feature. Here, we use splits generated by black box ensembles. The second type of guess concerns the maximum depth we might need for an optimal tree, where we relate the complexity of a tree ensemble to the depth of an equivalently complex class of individual trees. The third type of guess uses the accuracy of black box models on subsets of the data to guess lower bounds on the loss for each subproblem we encounter. Our guesses are guaranteed to predict as well or better than the black box tree-ensemble reference model: taking the sparsest decision tree that makes the same predictions as the black box, our method will find this tree, an equivalently good tree, or an even better one. Together, these guesses decrease the run time by several orders of magnitude, allowing fast production of sparse and interpretable trees that achieve black box predictive performance.

[*]These authors contributed equally.

2 Related Work

Our work relates to the field of decision tree optimization and thinning tree ensembles.

Decision tree optimization. Optimization techniques have been used for decision trees from the 1990s until the present (Bennett and Blue 1996; Dobkin et al. 1997; Farhangfar, Greiner, and Zinkevich 2008; Nijssen and Fromont 2007, 2010). Recently, many works have directly optimized a performance metric (e.g., accuracy) with soft or hard sparsity constraints on the tree size. Such decision tree optimization problems can be formulated using mixed integer programming (MIP) (Bertsimas and Dunn 2017; Verwer and Zhang 2019; Vilas Boas et al. 2021; Günlük et al. 2021; Rudin and Ertekin 2018; Aghaei, Gómez, and Vayanos 2021). Other approaches use SAT solvers to find optimal decision trees (Narodytska et al. 2018; Hu et al. 2020), though these techniques require data to be perfectly separable, which is not typical for machine learning. Carrizosa, Molero-Río, and Morales (2021) provide an overview of mathematical programming for decision trees.

Another branch of decision tree optimization has produced customized dynamic programming algorithms that incorporate branch-and-bound techniques. Hu, Rudin, and Seltzer (2019); Angelino et al. (2018); Chen and Rudin (2018) use analytical bounds combined with bit-vector based computation to efficiently reduce the search space and construct optimal sparse decision trees. Lin et al. (2020) extend this work to use dynamic programming. Aglin, Nijssen, and Schaus (2020) also use dynamic programming with bounds to find optimal trees of a given depth. Demirović et al. (2020) additionally introduce constraints on both depth and number of nodes to improve the scalability of decision tree optimization.

Our guessing strategies can further improve the scalability of all branch-and-bound based optimal decision tree algorithms without fundamentally changing their respective search strategies. Instead, we reduce time and memory costs by using a subset of thresholds and tighter lower bounds to prune the search space.

"Born-again" trees. Breiman and Shang (1996) proposed to replace a tree ensemble with a newly constructed single tree. The tree ensemble is used to generate additional observations that are then used to find the best split for a tree node in the new tree. Zhou and Hooker (2016) later follow a similar strategy. Other recent work uses black box models to determine splitting and stopping criteria for growing a single tree inductively (Bai et al. 2019) or exploit the class distributions predicted by an ensemble to determine splitting and stopping criteria (Van Assche and Blockeel 2007). Vandewiele et al. (2016) use a genetic approach to construct a large ensemble and combine models from different subsets of this ensemble to get a single model with high accuracy.

Another branch of work focuses on the inner structure of the tree ensemble. Akiba, Kaneda, and Almuallim (1998) generate if-then rules from each of the ensemble classifiers and convert the rules into binary vectors. These vectors are then used as training data for learning a new decision tree. Recent work following this line of reasoning extracts, ranks, and prunes conjunctions from the tree ensemble and organizes the set of conjunctions into a rule set (Sirikulviriya and Sinthupinyo 2011), an ordered rule list (Deng 2019), or a single tree (Sagi and Rokach 2020). Hara and Hayashi (2018) adopt a probabilistic model representation of a tree ensemble and use Bayesian model selection for tree pruning.

The work of Vidal and Schiffer (2020) produces a decision tree of minimal complexity that makes identical predictions to the reference model (a tree ensemble). Such a decision tree is called a born-again tree ensemble.

We use a reference model for two purposes: to help reduce the number of possible values one could split on and to determine an accuracy goal for a subset of points. By using a tree ensemble as our reference model, we can guarantee that our solution will have a regularized objective that matches or improves upon that of any born-again tree ensemble.

3 Notation and Objectives

We denote the training dataset as $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i$ are M-vectors of features, and $y_i \in \{0,1\}$ are labels. Let $\mathbf{x}$ be the $N \times M$ covariate matrix and $\mathbf{y}$ be the N-vector of labels, and let x_{ij} denote the j-th feature of $\mathbf{x}_i$. We transform each continuous feature into binary features by creating a split point at the mean value between every ordered pair of unique values present in the training data. Let k_j be the number of unique values realized by feature j, then the total number of features is $\tilde{M} = \sum_{j=1}^M (k_j - 1)$. We denote the binarized covariate matrix as $\tilde{\mathbf{x}}$ where $\tilde{\mathbf{x}}_i \in \{0,1\}^{\tilde{M}}$ are binary features.

Let $\mathcal{L}(t, \tilde{\mathbf{x}}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^N \mathbf{1}[y_i \neq \hat{y}_i^t]$ be the loss of the tree t on the training dataset, given predictions $\{\hat{y}_i^t\}_{i=1}^N$ from tree t. Most optimal decision tree algorithms minimize accuracy constrained by a depth bound, i.e.,

$$\underset{t}{\text{minimize}} \quad \mathcal{L}(t, \tilde{\mathbf{x}}, \mathbf{y}), \text{ s.t. depth}(t) \leq d. \tag{1}$$

Instead, Hu, Rudin, and Seltzer (2019) and Lin et al. (2020) define the objective function $R(t, \tilde{\mathbf{x}}, \mathbf{y})$ as the combination of the misclassification loss and a sparsity penalty on the number of leaves. That is, $R(t, \tilde{\mathbf{x}}, \mathbf{y}) = \mathcal{L}(t, \tilde{\mathbf{x}}, \mathbf{y}) + \lambda H_t$, where H_t is the number of leaves in the tree t and λ is a regularization parameter. They minimize the objective function, i.e.,

$$\underset{t}{\text{minimize}} \quad \mathcal{L}(t, \tilde{\mathbf{x}}, \mathbf{y}) + \lambda H_t. \tag{2}$$

Ideally, we prefer to solve (2) since we do not know the optimal depth in advance, and even at a given depth, we would prefer to minimize the number of leaves. But (1) is much easier to solve. We discuss details in Section 4.2.

4 Methodology

We present three guessing techniques. The first guesses how to transform continuous features into binary features, the second guesses tree depth for sparsity-regularized models, and the third guesses tighter bounds to allow faster time-to-completion. We use a boosted decision tree (Freund and Schapire 1995; Friedman 2001) as our reference model whose performance we want to match or exceed. We refer to Appendix A for proofs of all theorems presented.

4.1 Guessing Thresholds via Column Elimination

Since most state-of-the-art optimal decision tree algorithms require binary inputs, continuous features require preprocessing. We can transform a continuous feature into a set of binary dummy variables. Call each midpoint between two consecutive points a *split point*. Unfortunately, naïvely creating split points at the mean values between each ordered pair of unique values present in the training data can dramatically increase the search space of decision trees (Lin et al. 2020, Theorem H.1), leading to the possibility of encountering either a time or memory limit. Verwer and Zhang (2019) use a technique called bucketization to reduce the number of thresholds considered: instead of including split points between all realized values of each continuous feature, bucketization removes split points between realized feature values for which the labels are identical. This reduces computation, but for our purposes, it is still too conservative, and will lead to slow run times. Instead, we use a subset of thresholds from our reference boosted decision tree model, since we know that with these thresholds, it is possible to produce a model with the accuracy we are trying to match. Because computation of sparse trees has factorial time complexity, each feature we remove reduces run time substantially.

Column Elimination Algorithm: Our algorithm works by iteratively pruning features (removing columns) of least importance until a predefined threshold is reached: 1) Starting with our reference model, extract all thresholds for all features used in all of the trees in the boosting model. 2) Order them by variable importance (we use Gini importance), and remove the least important threshold (among all thresholds and all features). 3) Re-fit the boosted tree with the remaining features. 4) Continue this procedure until the training performance of the remaining tree drops below a predefined threshold. (In our experiments, we stop the procedure when there is any drop in performance at all, but one could use a different threshold such as 1% if desired.) After this procedure, the remaining data matrix is denoted by $\tilde{\mathbf{x}}_T^{\text{reduced}}$.

If, during this procedure, we eliminate too many thresholds, we will not be able to match black box accuracy. Theorem 4.1 bounds the gap between the loss of the black-box tree ensemble and loss of the optimal tree after threshold guessing.

Let T' be the ensemble tree built with $(\tilde{\mathbf{x}}_T^{\text{reduced}}, \mathbf{y})$. We consider t' to be any decision tree that makes the same predictions as T' for every observation in $\tilde{\mathbf{x}}_T^{\text{reduced}}$. We provide Figure 8 in the appendix as an illustration of finding a t' with the minimum number of leaves for a given ensemble. Note that $H_{t'}$ may be relatively small even if T' is a fairly large ensemble, because the predictions are binary (each leaf predicts either 0 or 1) and the outcomes tend to vary smoothly as continuous features change, with not many jumps.

Theorem 4.1. *(Guarantee for model on reduced data). Define the following:*

- *Let T be the ensemble tree built upon $(\mathbf{x}, \mathbf{y})$.*
- *Let T' be the ensemble tree built with $(\tilde{\mathbf{x}}_T^{\text{reduced}}, \mathbf{y})$.*
- *Let t' be any decision tree that makes the same predictions as T' for every observation in $\tilde{\mathbf{x}}_T^{\text{reduced}}$.*
- *Let $t^* \in \arg\min_t \mathcal{L}(t, \tilde{\mathbf{x}}_T^{\text{reduced}}, \mathbf{y}) + \lambda H_t$ be an optimal tree on the reduced dataset.*

Then, $\mathcal{L}(t^*, \tilde{\mathbf{x}}_T^{\text{reduced}}, \mathbf{y}) - \mathcal{L}(T, \mathbf{x}, \mathbf{y}) \leq \lambda(H_{t'} - H_{t^*})$, or equivalently, $R(t^*, \tilde{\mathbf{x}}_T^{\text{reduced}}, \mathbf{y}) \leq \mathcal{L}(T, \mathbf{x}, \mathbf{y}) + \lambda H_{t'}$.

That is, for any tree t' that matches the predictions of T' for $\tilde{\mathbf{x}}_T^{\text{reduced}}$ (even the smallest such tree), the difference in loss between the black box tree ensemble T and the optimal single tree t^* (based on $\tilde{\mathbf{x}}_T^{\text{reduced}}$) is not worse than the regularization coefficient times the difference between the sizes of two trees: t^* and t'. Equivalently, t^* will never be worse than t' in terms of the regularized objective. If we pick t' to be a born-again tree ensemble for T' (Vidal and Schiffer 2020) (which we can do because born-again tree ensembles make the same predictions as T' for all inputs, and therefore make the same predictions for $\tilde{\mathbf{x}}_T^{\text{reduced}}$), we can show that our thresholding approach guarantees that we will never do worse than the best born-again tree ensemble for our simplified reference model, in terms of the regularized objective. If we pick a t' with the minimal number of leaves, this theorem shows we even match or beat the simplest tree that can exactly replicate the reference models' predictions on the training set.

4.2 Guessing Depth

As discussed in Section 3, there are two different approaches to producing optimal decision trees: one uses a hard depth constraint and the other uses a per-leaf penalty. Algorithms that use only depth constraints tend to run more quickly but can produce needlessly complicated trees, because they do not reward trees for being shallower than the maximum depth. Depth constraints assess trees only by the length of their longest path, not by any other measure of simplicity, such as the length of the average path or the number of different decision paths. In contrast, per-leaf penalty algorithms (Hu, Rudin, and Seltzer 2019; Lin et al. 2020) produce sparse models, but frequently have longer running times, as they search a much larger space, because they do not assume they know the depth of an optimal tree. We show that by adding a depth constraint (a guess on the maximum depth of an optimal tree) to per-leaf penalty algorithms, such as GOSDT (Lin et al. 2020), we can achieve *per-leaf* sparsity regularization at run times comparable to *depth-constrained* algorithms, such as DL8.5 (Aglin, Nijssen, and Schaus 2020). In particular, we combine (1) and (2) to produce a new objective,

$$\underset{t}{\text{minimize}} \quad \mathcal{L}(t, \tilde{\mathbf{x}}, \mathbf{y}) + \lambda H_t \text{ s.t. depth}(t) \leq d, \qquad (3)$$

where we aim to choose d such that it reduces the search space without removing all optimal or near-optimal solutions. Most papers use depth guesses between 2 and 5 (Aglin, Nijssen, and Schaus 2020), which we have done in our experiments. However, Theorem 4.2 provides guidance on other ways to select a depth constraint to train accurate models quickly. Also, Theorem A.1 bounds the gap between the objectives of optimal trees with a relatively smaller depth guess and with no depth guess. This gap depends on the trade-off between sparsity and accuracy.

Interestingly, using a large depth constraint is often less efficient than removing the depth constraint entirely for GOSDT, because when we use a branch-and-bound approach with recursion, the ability to re-use solutions of recurring subproblems diminishes in the presence of a depth constraint.

Theorem 4.2. *(Min depth needed to match complexity of ensemble). Let B be the base hypothesis class (e.g., decision stumps or shallow trees) that has VC dimension at least 3 and let $K \geq 3$ be the number of weak classifiers (members of B) combined in an ensemble model. Let $\mathcal{F}_{ensemble}$ be the set of weighted sums of weak classifiers, i.e., $T \in \mathcal{F}_{ensemble}$ has $T(x) = sign(\sum_{k=1}^{K} w_k h_k(x))$, where $\forall k, w_k \in \mathbb{R}, h_k \in B$. Let $\mathcal{F}_{d,tree}$ be the class of single binary decision trees with depth at most:*

$$d = \lceil \log_2 ((K \cdot VC(B) + K) \cdot (3 \ln(K \cdot VC(B) + K) + 2)) \rceil .$$

It is then true that $VC(\mathcal{F}_{d,tree}) \geq VC(\mathcal{F}_{ensemble})$.

That is, the class of single trees with depth at most d has complexity at least that of the ensemble. The following results are special cases of Theorem 4.2:

- Suppose B is the class of decision trees with depth at most 3. To match or exceed the complexity of an ensemble of $K = 10$ trees from B with a single tree, it is sufficient to use trees of depth 11.
- Suppose B is the class of decision trees with depth at most 3. To match or exceed the complexity of an ensemble of $K = 100$ trees from B with a single tree, it is sufficient to use trees of depth 15.

The bound is conservative, so we might choose a smaller depth than is calculated in the theorem; the theorem provides an upper bound on the depth we need to consider for matching the accuracy of the black box.

4.3 Guessing Tighter Lower Bounds

Branch-and-bound approaches to decision tree optimization, such as GOSDT and DL8.5, are limited by the inefficiency of their lower bound estimates. To remove a potential feature split from the search, the algorithm must prove that the best possible (lowest) objectives on the two resulting subproblems sum to a value that is worse (larger) than optimal. Calculating tight enough bounds to do this is often slow, requiring an algorithm to waste substantial time exploring suboptimal parts of the search space before it can prove the absence of an optimal solution in that space.

We use informed guesses to quickly tighten lower bounds. These guesses are based on a reference model – another classifier that we believe will misclassify a similar set of training points to an optimal tree. Let T be such a reference model and $\hat{y}_i^T$ be the predictions of that reference model on training observation i. Define s_a as the subset of training observations that satisfy a boolean assertion a:

$$\begin{aligned} s_a &:= \{i : a(\tilde{\mathbf{x}}_i) = \text{True}, i \in \{1, ..., N\}\} \\ \tilde{\mathbf{x}}(s_a) &:= \{\tilde{\mathbf{x}}_i : i \in s_a\} \\ \mathbf{y}(s_a) &:= \{y_i : i \in s_a\} . \end{aligned}$$

We can then define our guessed lower bound as the disagreement of the reference and true labels for these observations (plus a penalty reflecting that at least 1 leaf will be used in solving this subproblem):

$$lb_{\text{guess}}(s_a) := \frac{1}{N} \sum_{i \in s_a} \mathbf{1}[y_i \neq \hat{y}_i^T] + \lambda. \tag{4}$$

We use this lower bound to prune the search space. In particular, we consider a subproblem to be solved if we find a subtree that achieves an objective less than or equal to its lb_{guess} (even if we have not proved it is optimal); this is equivalent to assuming the subproblem is solved when we match the reference model's accuracy on that subproblem. We further let the algorithm omit any part of the space whose estimated lower bound is large enough to suggest that it does not contain an optimal solution. That is, for each subproblem, we use the reference model's accuracy as a guide for the best accuracy we hope to obtain.

Thus, we introduce modifications to a general branch-and-bound decision tree algorithm that allow it to use lower bound guessing informed by a reference model. We focus here on approaches to solve (3), noting that (1) and (2) are special cases of (3). For a subset of observations s_a, let t_a be the subtree used to classify those points, and let H_{t_a} be the number of leaves in that subtree. We can then define:

$$R(t_a, \tilde{\mathbf{x}}(s_a), \mathbf{y}(s_a)) = \tfrac{1}{N} \textstyle\sum_{i \in s_a} \mathbf{1}[y_i \neq \hat{y}_i^{t_a}] + \lambda H_{t_a}.$$

For any dataset partition A, where $a \in A$ corresponds to the data handled by a given subtree of t:

$$R(t, \tilde{\mathbf{x}}, \mathbf{y}) = \textstyle\sum_{a \in A} R(t_a, \tilde{\mathbf{x}}(s_a), \mathbf{y}(s_a)).$$

Given a set of observations s_a and a depth constraint d for which we want to find an optimal tree, consider the partition of s_a resulting from "splitting" on a given boolean feature j in $\tilde{\mathbf{x}}$, i.e., the sets given by:

$$\begin{aligned} s_a \cap s_j &= \{i : a(\tilde{\mathbf{x}}_i) \wedge (\tilde{\mathbf{x}}_{ij} = 1), i \in \{1, ..., N\}\} \\ s_a \cap s_j^c &= \{i : a(\tilde{\mathbf{x}}_i) \wedge (\tilde{\mathbf{x}}_{ij} = 0), i \in \{1, ..., N\}\} . \end{aligned}$$

Let $t^*_{a \cap j, d-1}$ be the optimal solution to the set of observations $s_a \cap s_j$ with depth constraint $d - 1$, and let $t^*_{a \cap j^c, d-1}$ be similarly defined for the set of observations $s_a \cap s_j^c$. To find the optimal tree given the indices s_a and the depth constraint d, one approach is to find the first split by solving

$$\begin{aligned} \underset{j}{\text{minimize}} \, \big(& R(t^*_{a \cap j, d-1}, \tilde{\mathbf{x}}(s_a \cap s_j), \mathbf{y}(s_a \cap s_j)) \\ & + R(t^*_{a \cap j^c, d-1}, \tilde{\mathbf{x}}(s_a \cap s_j^c), \mathbf{y}(s_a \cap s_j^c)) \big) . \end{aligned} \tag{5}$$

This leads to recursive subproblems representing different sets of observations and depth constraints. One can solve for all splits recursively according to the above equation. Our modification to use lower bound guessing informed by a reference model applies to all branch-and-bound optimization techniques that have this structure.

Define majority(s_a) as the majority label class in s_a, and let $ub(s_a)$ be the objective of a single leaf predicting majority(s_a) for each point in s_a:

$$ub(s_a) = \tfrac{1}{N} \textstyle\sum_{i \in s_a} \mathbf{1}[y_i \neq \text{majority}(s_a)] + \lambda,$$

where (as usual) λ is a fixed per-leaf penalty for the model. A recursive branch-and-bound algorithm using lower bound guessing finds a solution for observations s_a with depth constraint d with the following modifications to its ordinary approach (see Appendix G for further details). Note that the subproblem is identified by (s_a, d).

Lower-bound Guessing for Branch-and-Bound Search

1. *(Use guess to initialize lower bound.)* If $ub(s_a) \leq lb_{\text{guess}}(s_a) + \lambda$ or $d = 0$, we are done with the subproblem. We consider it solved with objective $ub(s_a)$, corresponding to a single leaf predicting majority(s_a). Otherwise, set the current lower bound $lb_{\text{curr}}(s_a, d) = lb_{\text{guess}}(s_a)$ and go to Step 2.
2. Search the space of possible trees for subproblem (s_a, d) by exploring the possible features on which the subproblem could be split to form two new subproblems, with depth constraint $d - 1$, and solving those recursively. While searching:

   (a) *(Additional termination condition if we match black box performance.)* If we find a subtree t for subproblem (s_a, d) with $R(t, \tilde{\mathbf{x}}(s_a), \mathbf{y}(s_a)) \leq lb_{\text{current}}(s_a, d)$, we are done with the subproblem. We consider it solved with objective $R(t, \tilde{\mathbf{x}}(s_a), \mathbf{y}(s_a))$, corresponding to subtree t.

   (b) *(Modified lower bound update.)* At any time, we can opt to update the lower bound for the subproblem (with d' as shorthand for $d - 1$):

   $$lb_{\text{spl}} \leftarrow \min_{j \in \text{features}} (lb_{\text{curr}}(s_a \cap s_j, d') + lb_{\text{curr}}(s_a \cap s_j^c, d'))$$

   (This corresponds to splitting on the best feature.)

   $$lb_{\text{curr}}(s_a, d) \leftarrow \max(lb_{\text{curr}}(s_a, d), \min(ub(s_a), lb_{\text{spl}}))$$

   (Here, $\min(ub(s_a), lb_{\text{spl}})$ is the better of not splitting on any feature and splitting on the best feature. The max ensures that our lower bounds never decrease. The lb_{curr} term comes from either our initial lower bound guess or the lower bound we have so far.) If the lower bound increases to match $ub(s_a)$, we are done with the subproblem. We consider it solved with objective $ub(s_a)$, corresponding to a single leaf predicting majority(s_a). If the lower bound increases but does not match $ub(s_a)$, we apply Case 2a with the new, increased $lb_{\text{curr}}(s_a, d)$.

This approach is provably close to optimal when the reference model makes errors similar to an optimal tree. Let $s_{T,\text{incorrect}}$ be the set of observations incorrectly classified by the reference model T, i.e., $s_{T,\text{incorrect}} = \{i | y_i \neq \hat{y}_i^T\}$.

Let t_{guess} be a tree returned from our lower-bound guessing algorithm. The following theorem holds:

Theorem 4.3. *(Guarantee on guessed model performance). Let $R(t_{guess}, \tilde{\mathbf{x}}, \mathbf{y})$ denote the objective of t_{guess} on the full binarized dataset $(\tilde{\mathbf{x}}, \mathbf{y})$ for some per-leaf penalty λ (and with t_{guess} subject to depth constraint d). Then for any decision tree t that satisfies the same depth constraint d, we have:*

$$R(t_{guess}, \tilde{\mathbf{x}}, \mathbf{y}) \leq \frac{1}{N}\left(|s_{T,\text{incorrect}}| + \sum_{i \in s_{T,\text{correct}}} \mathbf{1}[y_i \neq \hat{y}_i^t]\right) + \lambda H_t.$$

That is, the objective of the guessing model is no worse than the union of errors made by the reference model and tree t.

The most significant consequence of this theorem is that

$$R(t_{\text{guess}}, \tilde{\mathbf{x}}, \mathbf{y}) - R(t^*, \tilde{\mathbf{x}}, \mathbf{y}) \leq \frac{1}{N}\left(|s_{T,\text{incorrect}}| - \sum_{i \in s_{T,\text{incorrect}}} \mathbf{1}[y_i \neq \hat{y}_i^{t^*}]\right)$$

where we selected t^*, an optimal tree for the given λ and depth constraint, as t, and we subtracted $R(t^*, \tilde{\mathbf{x}}, \mathbf{y})$ from both sides. This means that if the data points misclassified by the black box are a subset of the data points misclassified by the optimal decision tree, we are guaranteed not to lose any optimality. However much optimality we lose is a function of how many data points misclassified by the black box could have been correctly classified by the optimal tree.

We also prove that adding lower bound guessing after using threshold guessing does not change the worst case performance from Theorem 4.1.

Corollary 4.3.1. *Let T, T' and t' be defined as in Theorem 4.1. Let t_{guess} be the tree obtained using lower-bound guessing with T' as the reference model, on $\tilde{\mathbf{x}}_T^{reduced}, \mathbf{y}$, with depth constraint matching or exceeding the depth of t'. Then $\mathcal{L}(t_{guess}, \tilde{\mathbf{x}}_T^{reduced}, \mathbf{y}) - \mathcal{L}(T, \mathbf{x}, \mathbf{y}) \leq \lambda(H_{t'} - H_{t_{guess}})$, or equivalently, $R(t_{guess}, \tilde{\mathbf{x}}_T^{reduced}, \mathbf{y}) \leq \mathcal{L}(T, \mathbf{x}, \mathbf{y}) + \lambda H_{t'}$.*

The difference between this corollary and Theorem 4.1 is that it uses t_{guess} rather than t^*, without any weakening of the bound. Because of this corollary, our experimental results focus on lower bound guessing in combination with threshold guessing, using the same reference model, rather than using lower bound guessing on its own.

5 Experiments

Our evaluation addresses the following questions:

1. How accurate are interpretable models with guessing relative to black box models? (§5.1)
2. How well does each guessing method perform? (§5.2)
3. What happens if guesses are wrong? (§5.3)
4. What do sparse accurate trees look like? (§5.4)

We use seven datasets: one simulated 2D spiral pattern dataset, the Fair Isaac (FICO) credit risk dataset (FICO et al. 2018) for the Explainable ML Challenge, three recidivism datasets (COMPAS, Larson et al. 2016, Broward, Wang et al. 2022, Netherlands, Tollenaar and Van der Heijden 2013), and two coupon datasets (Takeaway and Restaurant), which were collected on Amazon Mechanical Turk via a survey (Wang et al. 2017). Table 1 summarizes all the datasets.

Unless stated otherwise, all plots show the median value across 5 folds with error bars corresponding to the first and third quartile. We use GBDT as the reference model for guessing and run it using scikit-learn (Pedregosa et al. 2011). We configure GBDT with default parameters, but select dataset-specific values for the depth and number of weak classifiers: (n_est, max_depth) $= (20, 3)$ for COMPAS and Spiral, $(40, 1)$ for Broward and FICO, $(50, 2)$ for Takeaway and Restaurant, and $(30, 2)$ for Netherlands. Appendix C presents details about our hyper-parameter selection process, and Appendix B presents the experimental setup and more details about the datasets.

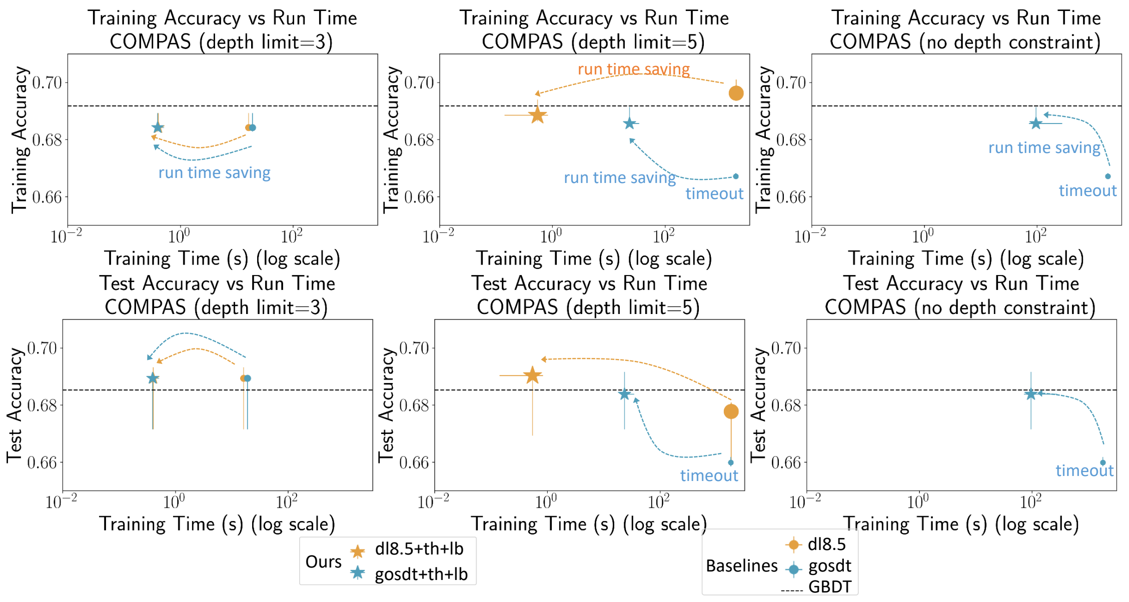


Figure 1: (Train-Time Savings from Guessing). Training and test accuracy versus run time for GOSDT (blue) and DL8.5 (gold) on the COMPAS data set with regularization 0.001 and different depth constraints. The black line shows the accuracy of a GBDT model (100 max-depth 3 weak classifiers). Circles show baseline performance (no guessing), stars show performance with all three guessing techniques, and marker size indicates the number of leaves. The displayed confidence bands come from 5-fold cross-validation. DL8.5 requires a depth constraint, so it does not appear in the right-most plots.

Dataset	samples	features	binary features
Spiral	100	2	180
COMPAS	6907	7	134
Broward	1954	38	588
Netherlands	20000	9	53890
FICO	10459	23	1917
Takeaway	2280	21	87
Restaurant	2653	21	87

Table 1: Datasets

5.1 Performance With and Without Guessing

Our first experiments support our claim that using guessing strategies enables optimal decision tree algorithms to quickly find sparse trees whose accuracy is competitive with a GBDT that was trained using 100 max-depth 3 weak classifiers.

Figure 1 shows **training and testing results** on the COMPAS data set. (Results for the other data sets are in Appendix C.1.) The difference between the stars and the circles shows that *our guessing strategies dramatically improve both DL8.5 and GOSDT run times, typically by 1-2 orders of magnitude.* Furthermore, *the trees we created have test accuracy that sometimes beats the black box*, because the sparsity of our trees acts as regularization.

Figure 2 shows the **accuracy-sparsity tradeoff** for different decision tree models (with GBDT accuracy indicated by the black line). The *batree* results are for the Born-Again tree ensembles of Vidal and Schiffer (2020). Results for the remaining datasets (which are similar) appear in Appendix C.2. The guessed models are both sparse and accurate, especially with respect to CART and batree.

We also compare GOSDT with all three guessing strategies to Interpretable AI's Optimal Decision Trees package, a proprietary commercial adaptation of Optimal Classification Tree (OCT) (Bertsimas and Dunn 2017) in Appendix H. The results show the run time of the two methods is comparable despite the fact that GOSDT provides guarantees on accuracy while the adaptation of OCT does not. GOSDT with all three guesses tends to find sparser trees with comparable accuracy.

5.2 Efficacy of Different Guessing Techniques

Comparing the three graphs in Figure 1 shows how guessing different depths affects run time performance. For the next three experiments, we fix the maximum depth to 5 and examine the impact of the other guessing strategies.

Guessing Thresholds: We found that guessing thresholds has the most dramatic impact, so we quantify that impact first. We compare the training time and accuracy with and without threshold guessing for GOSDT and DL8.5 on seven datasets. We limit run time to 30 minutes and run on a machine with 125GB memory. Experiments that time out are shown with orange bars with hatches; Experiments that run out of memory are in gray with hatches. If the experiment timed out, we analyze the best model found at the time of the timeout; if we run out of memory, we are unable to obtain a model. Figure 3 shows that the *training time after guessing thresholds (blue bars) is orders of magnitude faster than without thresholds guessing (orange bars)*. Orange bars are bounded by the time limit (1800 seconds) set for the experiments, but in reality, the training time is much longer. Moreover, threshold guessing leads to little change in both training and test accuracy

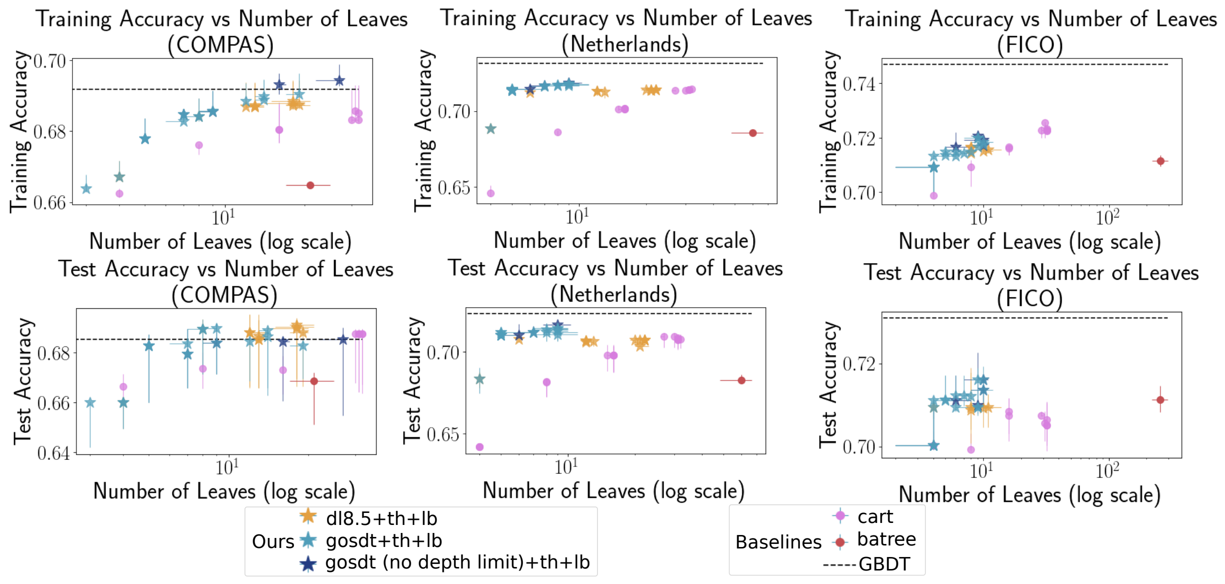


Figure 2: (Sparsity vs. Accuracy). DL8.5 and GOSDT use guessed thresholds and guessed lower bounds. CART is trained on the original dataset with no guess. This figure shows that our guessed trees define a frontier; we achieve the highest accuracy for almost every level of sparsity. The baseline methods (CART, batree) do not achieve results on the frontier.

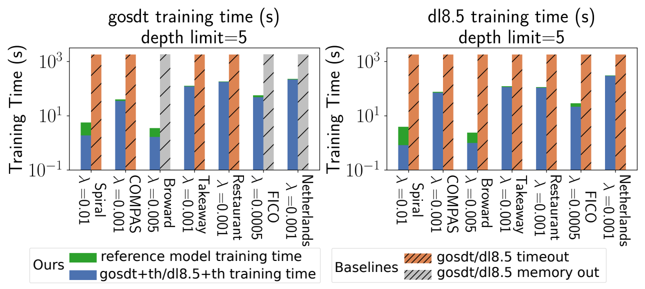


Figure 3: (Value of Threshold Guessing). Training time (logscale) for GOSDT and DL8.5 with depth limit 5. All baselines (hatched) timed out (orange) or hit the memory limit (gray). Green parts indicate threshold guessing times.

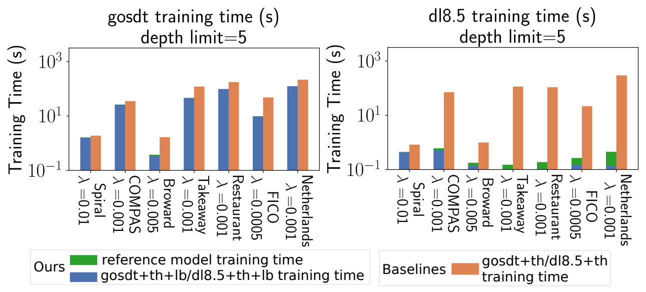


Figure 4: (Value of Lower Bound Guessing). Training time (logscale) for GOSDT and DL8.5 with and without lower-bound guessing, using depth limit 5 and threshold-guessing.

(see Appendix C.3).

Guessing Lower Bounds: Next, we add lower-bound guesses to the threshold guesses to produce Figure 4. The results are qualitatively similar to those in Figure 3: using lower bound guesses produces an additional run time benefit. Appendix C.4 shows more results and verifies that using lower-bound guesses often leads to a solution with the same training error as without lower-bound guesses or leads to finding a simpler solution with test error that is still comparable to the solution found without lower-bound guessing.

Value of Depth Constraints: We examine depth constraints' effect on GOSDT in Figure 5. We run GOSDT without a depth constraint, which produces an optimal tree for each of five folds using per-leaf regularization, and compare to GOSDT with a depth constraint. Above a certain threshold, depth constraints do not reduce training accuracy (since the constraint does not remove all optimal trees from the search space). Some depth constraints slightly improve test accuracy (see depths 4 and 5), since constraining GOSDT to find shallower trees can prevent or reduce overfitting.

Usually, depth constraints allow GOSDT to run several orders of magnitude faster. For large depth constraints, however, the run time can be worse than that with no constraints. Using depth guesses reduces GOSDT's ability to share information between subproblems. When the constraint prunes enough of the search space, the search space reduction dominates; if the constraint does not prune enough, the inability to share subproblems dominates.

5.3 When Guesses Are Wrong

Although rare in our experiments, it is possible to lose accuracy if the reference model is too simple, i.e., only a few thresholds are in the reduced dataset or too many samples are misclassified by the reference model. This happens when we choose a reference model that has low accuracy to begin

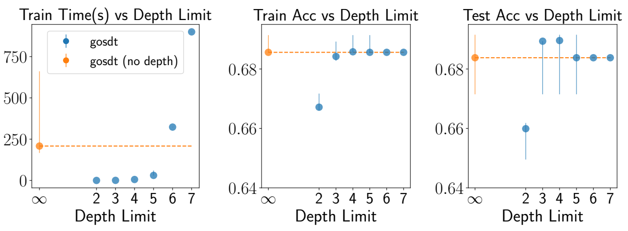


Figure 5: (Value of Depth Constraints). Performance across depth constraints (∞ means no constraint) for the COMPAS dataset with threshold guessing. Regularization 0.001.

with. Thus, we should check the reference model's performance before using it and improve it before use if necessary. Appendix D shows that if one uses a poor reference model, we do lose performance. Appendix C shows that for a wide range of reasonable configurations, we do not suffer dramatic performance costs as a result of guessing.

5.4 Trees

We qualitatively compare trees produced with our guessing technique to several baselines in Figure 6. We observe that with all three guessing techniques, the resulting trees are not only more accurate overall, but they are also sparser than the trees produced by the baselines. Figure 7 shows example GOSDT trees trained using all three guesses on the COMPAS dataset. Appendix C.5 shows more output trees and Appendix F compares decision sets transformed from our trees and trained by Dash, Gunluk, and Wei (2018) on the FICO dataset.

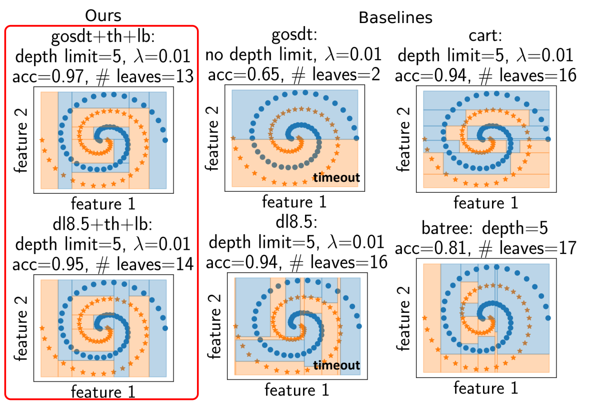
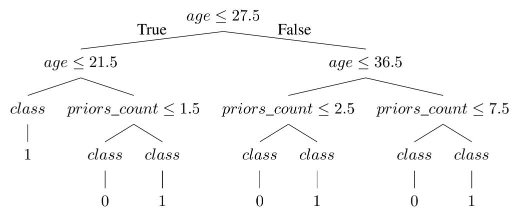
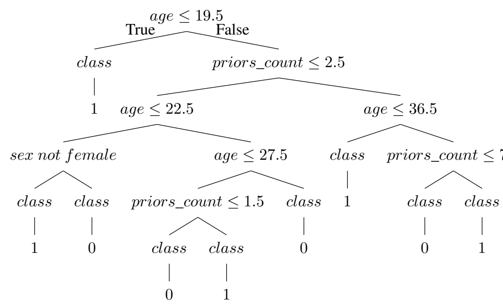


Figure 6: (Tree Visualization). Trees for GOSDT and DL8.5. Dots indicate the data points and the shaded areas are the corresponding classification by the tree. GOSDT trees were trained with and without all three guesses. DL8.5 trees were trained with and without threshold and lower bound guesses at depth 5. CART was trained with depth 5 and batree was trained using random forests with 10 max-depth 3 weak classifiers as the reference model. Training time is 1.606 seconds for GOSDT with all three guesses and 0.430 seconds for DL8.5 with all three guesses (left column). Both GOSDT without guesses and DL8.5 with only depth guesses timed out (middle column).



(a) GOSDT+th+lb (depth limit 3): training time=0.513 (sec), training accuracy=0.683, test accuracy=0.689, #leaves=7 on fold 2.



(b) GOSDT+th+lb (depth limit 5):training time=34.804 (sec), training accuracy=0.685, test accuracy=0.696, # leaves=9 on fold 2.

Figure 7: (Example Trees). GOSDT trees on the COMPAS dataset with guessed thresholds and guessed lower bounds at depth limit 3 and 5. The reference model was trained using 20 max-depth 3 weak classifiers. $\lambda = 0.001$.

6 Conclusion

We introduce smart guessing strategies to find sparse decision trees that compete in accuracy with a black box machine learning model while reducing training time by orders of magnitude relative to training a fully optimized tree. Our guessing strategies can be applied to several existing optimal decision tree algorithms with only minor modifications. With these guessing strategies, powerful decision tree algorithms can be used on much larger datasets.

Code Availability

Implementations of GOSDT and DL8.5 with the guessing strategies discussed in this paper are available at https://github.com/ubc-systopia/gosdt-guesses and https://github.com/ubc-systopia/pydl8.5-lbguess. Our experiment code is available at https://github.com/ubc-systopia/tree-benchmark.

Acknowledgements

We acknowledge the following grant support: NIDA DA054994-01, DOE DE-SC0021358-01, NSF DGE-2022040, NSF FAIN-1934964. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Nous remercions le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) de son soutien.

References

Aghaei, S.; Gómez, A.; and Vayanos, P. 2021. Strong Optimal Classification Trees. arXiv preprint arXiv:2103.15965.

Aglin, G.; Nijssen, S.; and Schaus, P. 2020. Learning optimal decision trees using caching branch-and-bound search. In AAAI Conference on Artificial Intelligence, volume 34, 3146–3153.

Akiba, Y.; Kaneda, S.; and Almuallim, H. 1998. Turning majority voting classifiers into a single decision tree. In Proceedings Tenth IEEE International Conference on Tools with Artificial Intelligence, 224–230. IEEE.

Angelino, E.; Larus-Stone, N.; Alabi, D.; Seltzer, M.; and Rudin, C. 2018. Learning Certifiably Optimal Rule Lists for Categorical Data. Journal of Machine Learning Research, 18(234): 1–78.

Bai, J.; Li, Y.; Li, J.; Jiang, Y.; and Xia, S. 2019. Rectified decision trees: Towards interpretability, compression and empirical soundness. arXiv preprint arXiv:1903.05965.

Bennett, K. P.; and Blue, J. A. 1996. Optimal decision trees. Technical report, Rensselaer Polytechnic Institute.

Bertsimas, D.; and Dunn, J. 2017. Optimal classification trees. Machine Learning, 106(7): 1039–1082.

Breiman, L.; Friedman, J.; Stone, C. J.; and Olshen, R. A. 1984. Classification and Regression Trees. CRC press.

Breiman, L.; and Shang, N. 1996. Born again trees. Technical report, University of California, Berkeley.

Carrizosa, E.; Molero-Río, C.; and Morales, D. R. 2021. Mathematical optimization in classification and regression trees. Top, 29(1): 5–33.

Caruana, R.; Lou, Y.; Gehrke, J.; Koch, P.; Sturm, M.; and Elhadad, N. 2015. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1721–1730.

Chen, C.; and Rudin, C. 2018. An optimization approach to learning falling rule lists. In International Conference on Artificial Intelligence and Statistics (AISTATS).

Dash, S.; Gunluk, O.; and Wei, D. 2018. Boolean Decision Rules via Column Generation. In Advances in Neural Information Processing Systems, volume 31, 4655–4665.

Demirović, E.; Lukina, A.; Hebrard, E.; Chan, J.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Stuckey, P. J. 2020. MurTree: Optimal Classification Trees via Dynamic Programming and Search. arXiv preprint arXiv:2007.12652.

Deng, H. 2019. Interpreting tree ensembles with intrees. International Journal of Data Science and Analytics, 7(4): 277–287.

Dobkin, D.; Fulton, T.; Gunopulos, D.; Kasif, S.; and Salzberg, S. 1997. Induction of shallow decision trees. IEEE Trans. on Pattern Analysis and Machine Intelligence.

Farhangfar, A.; Greiner, R.; and Zinkevich, M. 2008. A fast way to produce near-optimal fixed-depth decision trees. In Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM-2008).

FICO; Google; Imperial College London; MIT; University of Oxford; UC Irvine; and UC Berkeley. 2018. Explainable Machine Learning Challenge. https://community.fico.com/s/explainable-machine-learning-challenge.

Freund, Y.; and Schapire, R. E. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In Conference on Computational Learning Theory, 23–37. Springer.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. Annals of Statistics, 1189–1232.

Günlük, O.; Kalagnanam, J.; Li, M.; Menickelly, M.; and Scheinberg, K. 2021. Optimal decision trees for categorical data via integer programming. Journal of Global Optimization, 1–28.

Hara, S.; and Hayashi, K. 2018. Making tree ensembles interpretable: A bayesian model selection approach. In International Conference on Artificial Intelligence and Statistics (AISTATS), 77–85.

Hu, H.; Siala, M.; Hébrard, E.; and Huguet, M.-J. 2020. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI).

Hu, X.; Rudin, C.; and Seltzer, M. 2019. Optimal sparse decision trees. In Advances in Neural Information Processing Systems, 7267–7275.

Larson, J.; Mattu, S.; Kirchner, L.; and Angwin, J. 2016. How We Analyzed the COMPAS Recidivism Algorithm. ProPublica.

Laurent, H.; and Rivest, R. L. 1976. Constructing optimal binary decision trees is NP-complete. Information Processing Letters, 5(1): 15–17.

Lin, J.; Zhong, C.; Hu, D.; Rudin, C.; and Seltzer, M. 2020. Generalized and scalable optimal sparse decision trees. In International Conference on Machine Learning (ICML), 6150–6160.

Lou, Y.; Caruana, R.; and Gehrke, J. 2012. Intelligible models for classification and regression. In 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 150–158.

Morgan, J. N.; and Sonquist, J. A. 1963. Problems in the analysis of survey data, and a proposal. Journal of the American Statistical Association, 58(302): 415–434.

Narodytska, N.; Ignatiev, A.; Pereira, F.; Marques-Silva, J.; and RAS, I. 2018. Learning Optimal Decision Trees with SAT. In 27th International Joint Conference on Artificial Intelligence (IJCAI), 1362–1368.

Nijssen, S.; and Fromont, E. 2007. Mining optimal decision trees from itemset lattices. In 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 530–539.

Nijssen, S.; and Fromont, E. 2010. Optimal constraint-based decision tree induction from itemset lattices. Data Mining and Knowledge Discovery, 21(1): 9–51.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12: 2825–2830.

Quinlan, J. R. 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann.

Rudin, C.; and Ertekin, S. 2018. Learning Customized and Optimized Lists of Rules with Mathematical Programming. Mathematical Programming C (Computation), 10: 659–702.

Sagi, O.; and Rokach, L. 2020. Explainable decision forest: Transforming a decision forest into an interpretable tree. Information Fusion, 61: 124–138.

Shalev-Shwartz, S.; and Ben-David, S. 2014. Understanding machine learning: From theory to algorithms. Cambridge University Press.

Sirikulviriya, N.; and Sinthupinyo, S. 2011. Integration of rules from a random forest. In International Conference on Information and Electronics Engineering, volume 6, 194–198.

Tollenaar, N.; and Van der Heijden, P. 2013. Which method predicts recidivism best?: a comparison of statistical, machine learning and data mining predictive models. Journal of the Royal Statistical Society: Series A (Statistics in Society), 176(2): 565–584.

Van Assche, A.; and Blockeel, H. 2007. Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In European Conference on Machine Learning, 418–429. Springer.

Vandewiele, G.; Janssens, O.; Ongenae, F.; De Turck, F.; and Van Hoecke, S. 2016. GENESIM: genetic extraction of a single, interpretable model. In Advances in Neural Information Processing Systems, Workshop on Interpretable Machine Learning in Complex Systems, 1–6.

Verwer, S.; and Zhang, Y. 2019. Learning optimal classification trees using a binary linear program formulation. In AAAI Conference on Artificial Intelligence, volume 33, 1625–1632.

Vidal, T.; and Schiffer, M. 2020. Born-again tree ensembles. In International Conference on Machine Learning (ICML), 9743–9753.

Vilas Boas, M. G.; Santos, H. G.; Merschmann, L. H. d. C.; and Vanden Berghe, G. 2021. Optimal decision trees for the algorithm selection problem: integer programming based approaches. International Transactions in Operational Research, 28(5): 2759–2781.

Wang, C.; Han, B.; Patel, B.; Mohideen, F.; and Rudin, C. 2022. In Pursuit of Interpretable, Fair and Accurate Machine Learning for Criminal Recidivism Prediction. Journal of Quantitative Criminology.

Wang, T.; Rudin, C.; Doshi-Velez, F.; Liu, Y.; Klampfl, E.; and MacNeille, P. 2017. A Bayesian Framework for Learning Rule Sets for Interpretable Classification. Journal of Machine Learning Research, 18(70): 1–37.

Zhou, Y.; and Hooker, G. 2016. Interpreting models via single tree approximation. arXiv preprint arXiv:1610.09036.