The Search for Optimal Robust Classification Trees

Pushing the limits of exhaustive search

Kutlu Şan Demirören[1]

Supervisor(s): Emir Demirović[1]**, Koos van der Linden**[1]**, Daniël Vos**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 27, 2025

Name of the student: Kutlu Şan Demirören
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Daniël Vos, Jasmijn Baaijens

Abstract

Interpretability distinguishes decision trees from most other machine learning models; what they still have in common is that they are vulnerable to adversarial examples. Various robust decision tree algorithms exist; however, they either do not provide optimal results or are not scalable with data that has continuous features. In this work, we demonstrate RobTree, a scalable optimal robust decision tree algorithm for continuous features. We propose new theorems that reduce the number of thresholds to be considered to half of what was previously considered and give way to pruning techniques. The results of this paper indicate that RobTree vastly outperforms the state-of-the-art in terms of runtime for trees of depth two up to two orders of magnitude.

1 Introduction

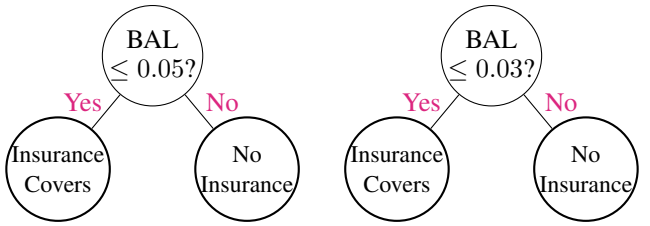
Building classification models is a fundamental task in machine learning, with their applications becoming increasingly widespread. However, these models are not always interpretable by humans. With the integration of machine learning into fields such as finance [10] and law [21], model interpretability becomes a critical concern, as the decisions made can have significant consequences. Decision trees, therefore play a vital role as interpretable machine learning models [11].

(Optimal) Decision trees The problem of finding *optimal* decision trees was shown to be NP-hard [15]. Hence, suboptimal greedy algorithms that optimize a local objective at each step such as CART [3], ID3 [19] and C4.5 [20] were developed.

Optimal decision trees were shown to perform better than greedy decision trees for most use cases [22]. Hence, research has been conducted on finding *optimal* decision trees effectively. For binary features, Mixed-Integer Linear Programming (MILP) and Dynamic Programming (DP) approaches, such as the work by Günlük et al. [12] and MurTree [8] respectively were proposed to explore the solution space effectively.

Moreover, for continuous features Mazumder et al. [17] proposed Quant-BnB, a scalable branch and bound (BnB) algorithm for Optimal Decision Trees. A slightly different approach, ConTree which augments BnB with the use of DP was proposed by Brita et al. [4] and was shown to achieve state-of-the-art scores.

Robust decision trees Decision trees, like most machine learning models, are vulnerable to adversarial examples [18]. Take the example in Figure 1 in which many car accidents involve drivers with blood alcohol levels (BAL) just above the legal limit of 0.05. If the tree in Figure 1a is used to decide the insurance policy, an attacker can slightly reduce BAL values to manipulate the tree into granting insurance coverage. In contrast, the tree in Figure 1b is more robust to such changes. Scenarios such as this motivate the need for *robust* decision trees that resist adversarial inputs.



(a) An optimal decision tree that puts the threshold at the legal limit. It is vulnerable to adversarial attacks

(b) A more *robust* decision tree. It is less vulnerable to perturbations by an adversary as the threshold is farther away.

Figure 1: Two decision trees that can be used to decide the insurance policy on a car crash based the on blood alcohol level(BAL) of the driver. If there are many incidents close to the 0.05 threshold, an adversary can perturb the data slightly to change the outcome.

Greedy algorithms for robustness, such as TREANT [5], GROOT [23], and FPRDT [13] that are suboptimal achieve generally good scores. However, due to their lack of optimality, they can produce arbitrarily bad trees [25], driving the research toward optimal robust decision trees.

Optimal robust decision trees Vos et al. [25] proposed ROCT, a MILP approach for constructing optimal robust decision trees, achieving state-of-the-art results. To the best of our knowledge, ROCT is the only optimal algorithm designed to ensure robustness against a deterministic adversary. However, ROCT has limited scalability as its runtime increases significantly with the dataset size, starting at trees of depth two. Hence, there is no scalable algorithm for constructing optimal robust decision trees that handle continuous features.

RobTree This work seeks to answer the question: *To what extent are search-based methods more scalable than MILP methods for finding optimal robust decision trees?* To address this gap in the literature, we propose RobTree, a scalable BnB algorithm for finding optimal robust classification trees with continuous features. Moreover, we also provide theorems regarding optimal robust decision trees that give way to various pruning methods during search. We show that for trees of depth two, RobTree solves instances up to two orders of magnitude faster than ROCT, achieving state-of-the-art performance. While our work is limited to depth two, it can be extended to trees of depth three and further.

In short, our contributions with this paper are several theorems that can be used in future work for speeding up the search for optimal robust trees. Moreover, we also provide RobTree, an adaptation of the Quant-BnB algorithm that achieves state-of-the-art results in a scalable manner. We also provide a novel search algorithm that is used as a subroutine in the RobTree algorithm.

2 Related Works

This section details the literature that supports the design of the RobTree algorithm, along with insight into the algorithms with which it is compared against.

2.1 (Optimal) Decision Trees

Greedy decision trees Early research on decision trees led to the development of greedy algorithms such as CART [3] and ID3 [19] which optimize local objectives with heuristics such as information gain and Gini impurity. While the aforementioned greedy algorithms still remain relevant, Van der Linden et al. [22] have shown that optimal trees often outperform greedy trees as the quality of greedy trees diminish with more data.

Optimal decision trees Search-based methods have achieved state-of-the-art results in learning optimal decision trees. For binary features the DL8.5 algorithm employs a combination of BnB techniques and caching, demonstrating their effectiveness in this context [1]. Building on the idea of using BnB, MurTree leverages the separable nature of the problem to apply DP along with novel pruning techniques to enhance the search process [8]. Unfortunately, due to the nature of our problem, the left and right subtrees of a node are not always independent of each other. As a result, memoization and DP approaches cannot be applied.

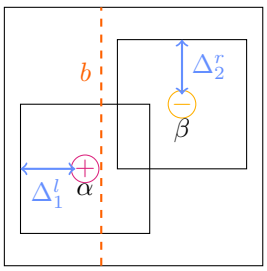
Continuous features Although continuous features can be binarized to fit the aforementioned models, the increase in the size of the problem due to binarization often results in a substantial increase in runtime and memory usage, as demonstrated by Mazumder et al. [17]. Hence, Mazumder et al. [17] proposed Quant-BnB, a BnB algorithm that constructs optimal decision trees for continuous features in a scalable manner. Unlike previous search algorithms, Quant-BnB does not rely on caching or DP. Instead, it leverages novel lower and upper bounds based on quantiles to prune substantial portions of the search space, which makes it appealing for our use case. The work of Mazumder et al. [17] is limited by the algorithm being only well-suited for shallow trees of maximum depth three.

ConTree further illustrated the effectiveness of branch and bound techniques by combining it with Dynamic Programming while still working with continuous features [4]. Outperforming Quant-BnB, ConTree's small runtime can be attributed to its specialized depth two solver and novel lower bounds it introduced. Demirović et al. [9] showed that search combined with aggressive pruning is also effective in black-box environments, utilizing a new trace-based pruning approach and elimination of symmetries.

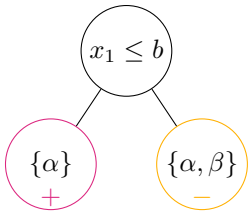
In this paper, we primarily use ideas from the work of Mazumder et al. [17] and Brita et al. [9] as they provide state-of-the-art results in optimal decision trees with continuous features.

2.2 (Optimal) Robust Decision Trees

Greedy robust trees Over the years, many greedy algorithms were proposed for constructing robust decision trees. For instance, Chen et al. [6] introduced a greedy algorithm that models the effect of adversarial perturbations by considering the maximum possible loss caused by an attacker. Since their splitting criterion has a computational complexity of $\mathcal{O}(n)$ in the size of the dataset, they propose a heuristic to approximate the split efficiently. Building on this idea,



(a) An illustration of the example with two data points. The horizontal axis represents the value of the first feature. The vertical axis is the second feature.



(b) A corresponding optimal depth one decision tree. In each leaf node, the set of data points that can reach that leaf node is given. Both leaf nodes have the classification of the negative class.

Figure 2: Visualization of the example in section 3.

TREANT was later introduced to enable more flexible modeling of the adversary by allowing the range of perturbation to be specified individually for each feature [5]. Given that both TREANT and the method by Chen et al. [6] have a time complexity of $\mathcal{O}(n^2)$, fast greedy algorithms such as GROOT [23] and FPRDT [13] that run in $\mathcal{O}(n \log n)$ time were proposed to improve scalability. Despite their practical success, these greedy algorithms can produce arbitrarily poor results [25]. Vos et al. [24] have shown an algorithm for the labeling of the leaf nodes in a given decision tree, which can improve greedy tree performance.

Optimal robust decision trees MILP has been a common approach for constructing optimal robust decision trees. As Justin et al. [16] and Blanco et al. [2] have used MILP approaches to construct trees robust against noise and random perturbations in the data. In a setting against an adversary, ROCT was proposed as an algorithm that constructs robust optimal decision trees that can be implemented as either a MILP or a MaxSAT formulation [25]. However, both formulations suffer from scalability issues, limiting ROCT's applicability to trees of depth two on datasets containing only a few thousand samples and tens of features.

In this work, we compare directly with ROCT in terms of runtime. We also compare with GROOT to serve as a baseline on the adversarial accuracy. Both algorithms achieve state-of-the-art results.

3 Preliminaries

This section aims to detail the terminology and theory required to understand the rest of the paper. To give an idea of the problem, we start with an example.

Example. Consider two data points $\alpha = (0.45, 0.5)$ and $\beta = (0.75, 0.7)$ that are made up of two features. Let α and β be of classification of positive (colored pink) and negative (colored yellow), respectively. Now consider an adversary that can perturb each feature positively and negatively by at most 0.1– we denote this as $\Delta_l^f = \Delta_r^f = 0.1 \; \forall f \in \mathcal{F}$. Hence, the region of space to which the adversary can move a data point is akin to a box, as illustrated in Figure 2a. For a tree of depth

one, if we choose a split at a threshold b at the root, then data point β will always end up in the right leaf as shown in Figure 2b. However, due to the adversary, α can end up in both the left and right leaf; hence it "co-exists" on the two leaves. We consider a data point correctly classified if it is correctly classified in all of the leaves it "co-exists" in. So even if α is correctly classified if it ends up in the left node, we consider it not correctly classified by the tree in Figure 2b as an adversary can perturb it in a way that it is misclassified. β, on the other hand, is correctly classified.

3.1 Terminology

We take close inspiration from the notation used by Vos et al. [25] and Mazumder et al. [17]. The distribution of the data we classify using our decision tree is denoted by D.

- Let $\mathcal{F}$ be the set of features that the data points have. Then f refers to a feature of each data point. For a data point x, x^f would refer to a specific feature of x. For a collection of data points D, D^f is a set that contains all values for feature f in D.
- Since we are assuming a box-shaped attack model, we define a range that the attacker can move the value of each feature f in. Δ_f^l and Δ_f^r denote the maximum amount by which an attacker can shift feature f negatively and positively, respectively, for each data point.
- $S(x)$ denotes the set of all possible perturbations that an attacker can apply to a sample x.
- $\mathcal{T}$ is used to refer to a decision tree that maps an input x to a leaf node $t = \mathcal{T}(x)$ which assigns it the label c_t. $\mathcal{T}_L$ is the set of all leaf nodes of the decision tree. Furthermore, $\mathcal{T}_L^{S(x)}$ refers to all leaf nodes in which an attacker can place a data point x.
- The split of a node is a boundary b on a feature f such that it directs the points $x_i^f \leq b$ to the left subtree and the rest to the right.
- We define $\phi = (f_0, [a, b], F_1, F_2)$. This represents a candidate tree of depth two that splits at a feature f_0 at a threshold between $[a, b]$. Moreover, the left and right subtrees of the root can split from features in the sets F_1 and F_2, respectively. We say that a tree $\mathcal{T}$ "can be made from ϕ" if it fits the description of ϕ.
- We say that a sequence of points $(t_0, \ldots, t_s)$ are almost s-equi-spaced in $[a, b]$ if $t_0 = a, t_s = b$ and $t_j = \lfloor a + (j/s)(b - a) \rfloor$ for $1 \leq j \leq s - 1$.
- We define $u(f)$ to be the number of unique values for a feature f. Hence we let $w_1^f < \ldots < w_{u(f)}^f$ denote the distinct values in f, sorted. Moreover, we define $D_{[a,b]}^f$ to be the data points that have values inside the range $[w_a^f, w_b^f]$ for a feature f and $w_0^f = -\infty$.
- We use $\nu_1, \ldots, \nu_{n'}$ to refer to the order in which our exhaustive search algorithm assigns values to the non-leaf nodes. ν_1 is always the root node.

3.2 Problem Definition

More formally, the objective of this paper is to solve the same problem as ROCT, which is the following min-max problem:

$$\min_{\theta} \mathrm{E}_{(x,y)\sim D} \left(\max_{\delta \in S(x)} L(\theta, x + \delta, y) \right) \tag{1}$$

That is, find the optimal model θ that minimizes the maximum loss that can be incurred due to an attacker. For a given decision tree $\mathcal{T}$ that is created by a model θ, we define the following to be our loss function on a dataset D.

$$\mathcal{L}(D, \mathcal{T}, S) = \sum_{(x,y)\sim D} \left[\bigvee_{t \in \mathcal{T}_L^{S(x)}} c_t \neq y \right] \tag{2}$$

Vos et al. [25] have shown that for 0-1 loss, the objective in Equation 1 can be rewritten as the minimization problem:

$$\min_{\theta} \mathcal{L}(D, \mathcal{T}, S) \tag{3}$$

The goal of this paper is to create an exhaustive search algorithm that minimizes the same objective function as ROCT (the one given in Equation 3) but in a more scalable manner.

4 Theoretical Contributions

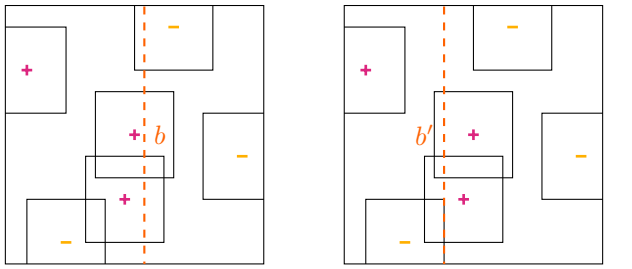
Before diving into the algorithm, we demonstrate the theorems that are used in the algorithm for pruning and reduction of the search space.

Threshold Consideration We show that the number of thresholds considered can be halved compared to those considered by the current state-of-the-art, ROCT. To find the optimal split for a feature f, ROCT selects thresholds from the set $\{x_0^f - \Delta_f^l, x_0^f + \Delta_f^r, \ldots, x_n^f + \Delta_f^r\}$, i.e., both outer bounds of each data point's perturbation range. However, Vos et al. [25] did not prove that this yields optimal splits. In contrast, we show that not only is it optimal, but selecting only the *right* bound per data point is also optimal, a fact which was seemingly not known in the design of ROCT. We use an exchange argument to show that we can convert any optimal robust decision tree into a tree that only considers the right outer bounds without loss of optimality.

Theorem 1. *Let $x_0, \ldots, x_n$ be a collection of data points. Then for a feature f the split threshold can be chosen exclusively from the set $F_f = \{-\infty, x_0^f + \Delta_f^r, \ldots, x_n^f + \Delta_f^r\}$ without loss of optimality.*

Proof. Let $\mathcal{T}^*$ be a robust optimal decision tree. We demonstrate that any decision threshold for a feature f in $\mathcal{T}^*$ can be changed to a threshold in F_f without loss of optimality. Take an arbitrary decision node a from $\mathcal{T}^*$ that splits on a feature f with a decision threshold b. Let $D' = \{x_0, \ldots, x_{n'}\}$ be the data points that can reach a through some perturbation in $S(x_i)$. We can then divide the data points into two sets that denote the points to the left and right of b, $A = \{x_i \in D' \mid x_i^f \leq b\}$ and $B = \{x_i \in D' \mid x_i^f > b\}$ respectively.

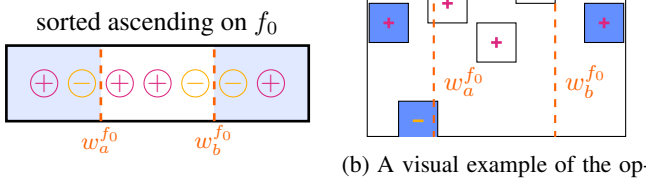
Case 1: If $|B| = 0$ then the threshold b can be changed to $b' = \max(\{-\infty\} \cup A^f) + \Delta_f^r$ without loss of optimality.



(a) A visualisation of the data points D' for an arbitrary decision node splitting a feature at threshold b.

(b) Moving any threshold b to a threshold $b' \in F_f$. Optimality is not lost as $\mathcal{C}' \subseteq \mathcal{C}$ and all other points are unaffected.

Figure 3: Visualization of case 3 of Theorem 1.



(a) A visualization of the operation in Theorem 3 inspired by Mazumder et al. [17]. The points in the range $[w_a^{f_0}, w_b^{f_0}]$ are "dropped".

(b) A visual example of the operation in an adversarial setting. The adversary does not affect which points are "dropped". In this case, the points without fill are dropped.

Figure 4: The intuition behind Theorem 3.

Case 2: If $|A| = 0$ similarly, the threshold b can be changed to $b' = -\infty$. This does not lose optimality as the data points that could have reached both the left and right subtrees now can only reach the right subtree.

Case 3: If $|A|, |B| > 0$ consider the set $\mathcal{C} = \{ x_i^f \in D' \mid x_i^f - \Delta_f^l \leq b, x_i^f + \Delta_f^r > b\}$ (i.e the data points that can be directed to both sub trees of a by an attacker). Now, b can be changed to be the outer boundary of the rightmost data point that is in A but not $\mathcal{C}$. More formally, to $b' = \max(\{-\infty\} \cup (A \setminus \mathcal{C})^f) + \Delta_f^r$. This does not lose optimality as by the choice of b' the set $\mathcal{C}' = \{x_i \in D' \mid x_i^f - \Delta_f^l \leq b', x_i^f + \Delta_f^r > b'\}$ is a subset of the set $\mathcal{C}$. This is because moving the threshold to the left in this manner cannot add a new element to $\mathcal{C}$ from A or B as visualized in Figure 3. Moreover, the data points in $\mathcal{C} \setminus \mathcal{C}'$ are now directed to the right subtree rather than both. The data points that can "co-exist" in both subtrees are now reduced, which cannot lead to more misclassifications. Since all other data points outside of $\mathcal{C}$ are unaffected, the tree is still optimal.

Since a was arbitrarily chosen, we can do a finite amount of these changes in any optimal tree $\mathcal{T}^*$ without losing optimality and end up using only the thresholds of the set F_f for each feature f.

□

Similarity Lowerbound The following bound is similar to the similarity lower-bound [14], which has been proven for non-robust optimal trees. The idea behind the theorem is that for a split at threshold index i, the split at index $i+1$ can decrease the loss by at most one. We utilize it to prune thresholds similar to neighborhood pruning in ConTree [4]. We only show it to work one direction as the other direction is not used in our algorithm.

Theorem 2. *Let $F_f = x_0^f, \ldots, x_n^f$ be sorted ascending and $\mathcal{UB}$ be the current upper-bound. For a node ν, let Θ be the minimum loss that can be acquired when ν splits on $x_i^f + \Delta_f^r$. Then splitting on the points in $\{x_{i+j}^f + \Delta_f^r \mid 0 \leq j \leq \Theta - \mathcal{UB}\}$ cannot result in a better tree than the upper bound.*

Proof. For an arbitrary node a, let $b = x_i^f + \Delta_f^r$ be the chosen threshold and Θ be the error of the optimal tree with the chosen threshold. Let $\mathcal{C} = \{x_t \mid x_t \in D', x_t - \Delta_f^l \leq b, x_t + \Delta_f^r > b\}$. By choosing $b' = x_{i+1}^f + \Delta_f^r$, $\mathcal{C}' = \{x_t \mid x_t \in D', x_t - \Delta_f^l \leq b', x_t + \Delta_f^r > b'\} \subseteq \mathcal{C} \setminus \{x_{i+1}\}$. Now that $x_{i+1} \notin \mathcal{C}'$, i.e, it can only be on one side of the threshold, hence it can only be correctly classified. Thus, the minimum loss that can be incurred can decrease by at most one. Consequently, the next smallest $\Theta - \mathcal{UB}$ possible thresholds in F_f that are bigger than b cannot result in a better tree than the upper bound. □

Quantile-based lower bound We also provide a lower bound on the error of any candidate tree ϕ inspired closely from the work of Mazumder et al. [17]. We define $L_1(D, F, S)$ to be the minimum classification error for a tree of depth one which splits on features in F in its root and with an attacker with perturbations in S. Similarly, let $L_2(D, \phi, S)$ be the minimum error attainable from trees that can be made from ϕ. Now, we define the following auxiliary function

$$W_0^S(D, \phi) = L_1(D_{[0,a]}^{f_0}, F_1, S) + L_1(D_{[b,u(f_0)]}^{f_0}, F_2, S)$$

for $\phi = (f_0, [a, b], F_1, F_2)$. The intuition behind the lower bound is that the samples in the middle are "dropped" and the left and right subtrees are made to be independent as Figure 4 demonstrates.

Theorem 3. *W_0 is a lower bound on the true optimal error L_2.*

Proof. For a candidate tree ϕ let $b' \in [a, b]$ be the any split in the root. Let A and B be the set of points that can reach the left and right subtrees, respectively. It is easy to see that $D_{[0,a]}^{f_0} \subseteq A$ and $D_{[b,u(f_0)]}^{f_0} \subseteq B$ and $D_{[0,a]}^{f_0} \cap D_{[b,u(f_0)]}^{f_0} = \varnothing$. Hence $L_1(D_{[0,a]}^{f_0}, F_1, S) + L_1(D_{[b,u(f_0)]}^{f_0}, F_2, S) \leq L_2(D_{[0,a]}^{f_0} \cup D_{[b,u(f_0)]}^{f_0}, \phi, S) \leq L_2(D, \phi, S)$. Since the LHS is equal to W_0^S, we have shown that it is a lower bound. □

Taking inspiration from Mazumder et al. [17],a similar argument can be made to prove that the following function $W_{s'}$ is a lower bound on L_2. Let $t_0, \ldots, t_{s'}$ be almost s' equispaced in $[a, b]$. We define

$$W_{s'}^S(D, \phi) = W_0^S(D, \phi) + \min_{1 \leq j \leq s'} W_0^S(D_{[a,b]}^{f_0}, \phi_j) \quad (4)$$

for $\phi_j = (f_0, [t_{j-1}, t_j], F_1, F_2)$. This again follows the same idea as Theorem 3, however, the loss on the middle part that was dropped is also accounted. A full proof can be found in Appendix D.

5 RobTree: a Scalable Search Framework

We only explain the framework for trees of depth two, as the idea can be extended to depth three and onward following Mazumder et al. [17]. In a nutshell, the algorithm is made up of two components: an exhaustive search algorithm that is used as a subroutine, and the RobTree framework itself. We first explain the exhaustive search algorithm before the RobTree framework.

Exhaustive search algorithm The algorithm for performing an exhaustive search given a candidate ϕ can be found in Algorithm 1. It can also be used independently from the RobTree framework to find optimal robust trees, which we refer to as the "Pure Search" approach. As demonstrated by Vos et al. [24], jointly assigning leaf values and decision thresholds is challenging. Our algorithm tackles this by exhaustively iterating over all possible leaf labelings and, for each permutation, exploring all combinations of threshold values at the decision nodes. By sorting the data points per feature and reusing errors from previously evaluated thresholds, we can keep track of the subtrees each point can reach and the leaves it is misclassified in. This allows us to compute the error in $\mathcal{O}(1)$ time per permutation. We also outline other pruning methods and symmetry breaks that were used to speed up the algorithm.

Pruning For a decision node a we only consider splits as outlined in Theorem 1. With this, the number of decisions considered in our search algorithm per feature is cut in half. In fact, for all features f and a data point t, all thresholds $x_t^f + \Delta_f^r$ are pruned away at a node ν if t cannot possibly reach the node ν, as these thresholds are redundant. Moreover, applying Theorem 2 to a call with a newly assigned threshold $b = x_i^f + \Delta_f^r$ yielding result Θ, allows us to prune thresholds in $\{x_{i+j}^f + \Delta_f^r \mid 0 \leq j \leq \Theta - \mathcal{UB}\}$ from consideration at the current node. Taking inspiration from the trace-based pruning from Demirović et al. [9], at the beginning of each call, the search is early-stopped for trees that already misclassify more data points than the upper bound without assigning values to every node. This reduces the effectiveness of the previous pruning method, but empirically has been found to result in a shorter runtime. For a number of nodes n', there are $\frac{1}{n'+1} \cdot \frac{2n'!}{(n'!)^2}$ number of different binary trees [7]. Hence, by assuming a perfect tree structure, we avoid this large overhead. Last, we assign an arbitrary threshold value for decision nodes that have both of their children with the same classification, as the value of the threshold will not change the outcome.

Summary The pseudocode for our algorithm can be found in Algorithm 1. We augment the enumeration of all possible trees with the aforementioned pruning techniques. By sorting the data points per feature and keeping track of the points on both sides of a chosen threshold in a queue, we can implement the steps in lines 25 and 28 in $\mathcal{O}(1)$. The optimal solution for a candidate can be found by calling Search with each permutation of leaf labels $C_L \in \{0,1\}^4$.

Algorithm 1 The Exhaustive Search Algorithm

1: **Input:** data points D, upper-bound $\mathcal{UB}$ a candidate tree $\phi = (f_0, [a, b], F_1, F_2)$, current decision node ν_i, labels of each leaf C_L, current error e.
2: **Procedure** Search$(D, \mathcal{UB}, \phi, \nu_i, C_L, e)$
3: **if** $e \geq \mathcal{UB}$ **then return** e
4: $U \leftarrow \infty$
5: $\mathcal{C} \leftarrow \varnothing$ ▷ Queue of points that can reach both sides
6: $i \leftarrow 0$ ▷ Pointer to the ith biggest value for a feature
7: **if** ν_i is the root of the tree **then**
8: **for** non-pruned thresholds $\tau \in (a, \ldots, b)$ **do**
9: Set ν_i to split on $w_{f_0}^\tau + \Delta_{f_0}^r$ at f_0 in the root.
10: continue exhaustive search with the next node.
11: prune thresholds if possible
12: **else if** ν_i is the left or right decision node **then**
13: **for** f in either F_1 or F_2 depending on ν **do**
14: set ν_i to split at threshold $-\infty$
15: update e accordingly
16: **if** the children have same the classification **then**
17: assign arbitrary threshold to ν_i
18: Update e accordingly
19: **return** Search$(D, \mathcal{UB}, \phi, \nu_{i+1}, C_L, e)$
20: **for** non-pruned thresholds $\tau \in (0, \ldots, u(f))$ **do**
21: ▷ sorted ascending
22: set ν to split on $w_f^\tau + \Delta_f^r$ at feature f.
23: **while** $\mathcal{C}$.front() $\leq w_f^\tau$ **do** ▷ On feature f
24: $x \leftarrow \mathcal{C}$.pop()
25: update $\mathcal{T}_L^{S(x)}$ and e.
26: **while** $x_i^f - \Delta_f^l \leq w_f^\tau + \Delta_f^r$ **do** ▷ Sorted
27: $\mathcal{C}$.push(x_i)
28: Update $\mathcal{T}_L^{S(x)}$ and e
29: $i \leftarrow i + 1$
30: $U' \leftarrow$ Search$(D, \mathcal{UB}, \phi, \nu_{i+1}, C_L, e)$.
31: $U \leftarrow \min(U, U')$
32: prune thresholds if possible
33: **return** U

The RobTree framework The exhaustive search algorithm can be used for finding deeper trees and is used as a subroutine in the RobTree framework in Algorithm 2. The framework, adapted from the work of Mazumder et al. [17] maintains a set of "alive" candidate trees $\texttt{AL}^k$ at each iteration k. All possible trees can be made with the candidates in $\texttt{AL}^0$. At each iteration, for each candidate $\phi = (f_0, [a, b], F_1, F_2) \in \texttt{AL}^k$, the framework reduces the interval of the root split by dividing it into $s+1$ new smaller intervals. Moreover, the algorithm uses the aforementioned lower bound in Theorem 3 to prune features from F_1 and F_2 in each of the new candidates ϕ_j. Exhaustive search is used as a subroutine either when the interval $[a, b]$ cannot be made smaller or on the quantile ends to obtain potentially tighter upper bounds. The algorithm terminates when there is no potential candidate that can give a

Algorithm 2 The RobTree Algorithm

1: **Input:** Data D, integer $s \geq 2$.
2: Initiliaze $\texttt{AL}^0 \leftarrow \bigcup_{f_0 \in \mathcal{F}} \{(f_0, [0, u(f_0)], \mathcal{F}, \mathcal{F})\}$
3: $k \leftarrow 1$
4: $\mathcal{UB} \leftarrow \infty$
5: **while** $|\texttt{AL}^{k-1}| > 0$ **do**
6: $\quad \texttt{AL}^k \leftarrow \varnothing$
7: $\quad$ **for** $(f, [a, b], F_1, F_2) \in \texttt{AL}^{k-1}$ **do**
8: $\quad\quad$ **if** $b - a \leq s$ **then**
9: $\quad\quad\quad U' \leftarrow \min_{c_l \in \{0,1\}^4} \text{Search}\big(D, \mathcal{UB}, \phi, \nu_0, c_l, 0\big)$
10: $\quad\quad\quad \mathcal{UB} \leftarrow \min(\mathcal{UB}, U')$
11: $\quad\quad$ **else**
12: $\quad\quad\quad (t_0, \ldots, t_s)$ almost s-equi-spaced in $[a, b]$
13: $\quad\quad\quad$ **for** $t_j \in \{t_0, \ldots, t_s\}$ **do**
14: $\quad\quad\quad\quad \phi_j \leftarrow (f_0, [t_j, t_j], F_1, F_2)$
15: $\quad\quad\quad\quad U' \leftarrow \min_{c_l \in \{0,1\}^4} \text{Search}\big(D, \mathcal{UB}, \phi_j,$
16: $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \nu_0, c_l, 0\big)$
17: $\quad\quad\quad\quad \mathcal{UB} \leftarrow \min(\mathcal{UB}, U')$
18: $\quad\quad\quad \texttt{AL}^k \bigcup_{j=1}^{s} \{(f_0, [t_{j-1}, t_j], F_{1,j}, F_{2,j})\}$
19: **return** $\mathcal{UB}$

better solution. While the aforementioned steps are inspired by Mazumder et al. [17], one novel addition we make to the Quant-BnB algorithm is that we prune the possible thresholds from the root using Theorem 2 after the calls on lines 9 and 15.

Feature Pruning As in Quant-BnB [17], the next set of alive candidates is determined in line 17 of Algorithm 2. For a new candidate ϕ_j, we prune features from $f_1 \in F_1$ by checking whether there is possibly an optimal tree that splits on f_1 on the left child of the root. We do this through the use of an easy to compute lower bound W_s^S. We do the same for features in F_2. More formally, while creating $s + 1$ new candidates from a candidate ϕ, we use the following sets to the determine the sets F_1 and F_2 for candidate $0 \leq j \leq s$. Let $(t_0, \ldots, t_s)$ be almost s equi-spaced in $[a, b]$.

$$F_{1,j} = \{f_1 \in F_1 \mid W_{s'}^S(D, (f_0, [t_{j-1}, t_j], \{f_1\}, F_2) < \mathcal{UB}\}$$
$$F_{2,j} = \{f_2 \in F_2 \mid W_{s'}^S(D, (f_0, [t_{j-1}, t_j], F_1, \{f_2\}) < \mathcal{UB}\}$$

If the lower bound is higher than the upper bound, that feature can be pruned away without risk as shown by Theorem 3. We can compute L_1 in $\mathcal{O}(|D| \cdot |\mathcal{F}|)$ time by adapting Algorithm 1 for trees of depth one.

Warm starting Vos et al .[25] found that warm starting the MIP model yielded the best results in their experiments. We employed a similar idea in RobTree as we warm-start our algorithm with a run of GROOT, providing an upper bound to initialize our algorithm with.

6 Experimental Setup and Results

The design of our experiments is made to answer the following questions: 1) how do RobTree and Pure Search's runtimes compare to the state-of-the-art robust optimal tree algorithms; 2) to what extent does warm starting RobTree and Pure Search improve their performance; 3) how does RobTree's anytime performance compare to the other state-of-the-art algorithms. RobTree's test accuracy is discussed in Appendix A as Vos et al. [25] have already shown the benefit of optimal robust decision trees on test data.

Dataset(OpenML)	n	p	Maj.	Avg. n'
breast-cancer	683	9	.650	9.9
cylinder-bands	277	37	.643	22.1
haberman	306	3	.735	30.0
blood-transfusion	748	4	.762	43.0
diabetes	768	8	.651	142.5
ionosphere	351	34	.641	190.4
wine	6497	11	.633	231.2
banknote-authentication	1372	4	.555	1016.5

Table 1: Summary of the datasets from OpenML. p is the number of features, Maj. is the ratio of the majority class, and n' is the number of distinct values per feature. Rows sorted by average n'.

Parameters To answer these questions, we follow a similar experimental setup to that of Vos et al. [25], assuming an attacker can perturb each (normalized) feature value by a fixed value ϵ. Formally, we assume $\Delta_f^l = \Delta_f^r = \epsilon \; \forall f \in \mathcal{F}$. As in Vos et al. [25], for each selected dataset, we choose ϵ values corresponding to 25%, 50%, and 75% of the adversarial accuracy range, which is the range from the minimum to the maximum adversarial accuracy. This approach avoids selecting ϵ values that are too small, rendering the adversary ineffective, or too large, reducing the learning problem to a trivial majority-class prediction game. We set the hyperparameter $s = 3$, and dynamically compute s' as $s' = \lfloor \frac{0.6ns}{b-a} \rfloor$ for a candidate $\phi = (f_0, [a, b], F_1, F_2)$, following Mazumder et al. [17].

Algorithms We benchmark RobTree against Pure Search to assess the suitability of the framework inspired from Mazumder et al. [17] over a pure search-based method in adversarial settings. Pure Search is benchmarked by running RobTree with $s = \infty$. Additionally, in Appendix A we benchmark our algorithm against GROOT, a non-optimal algorithm in terms of accuracy. As shown by Vos et al. [23], GROOT can produce strong results on most datasets, making it a suitable choice as a baseline on adversarial test accuracy.

ROCT includes several variants based on MILP and SAT-solving algorithms. In our runtime and accuracy benchmarking, we compare RobTree with the versions highlighted in the original paper by Vos et al. [25], specifically LSU-MaxSAT and MILP-warm. Both approaches represent each data point as a variable over a given domain and optimize an objective function under constraints using a general-purpose solver. In contrast, RobTree explicitly searches over the space of all possible trees, employing pruning techniques to reduce the search space. Hence, benchmarking RobTree against ROCT is important in assessing the scalability of search methods in comparison to MILP methods that employ a general solver.

Datasets To evaluate whether our algorithm improves upon ROCT, we use the same datasets as those used by Vos et al.

Dataset	ϵ	ROCT		Ours			
		LSU-MaxSAT	MILP warm	Pure Search	Pure Search warm	RobTree	RobTree warm
banknote-authentication	.07	456	-	6	4	**2**	3
	.09	1354	-	13	14	**6**	**6**
	.11	-	-	23	24	**13**	15
blood-transfusion	.01	50	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.02	46	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.03	29	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
breast-cancer	.28	5	464	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.39	4	376	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.45	4	528	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
cylinder-bands	.23	226	-	14	**9**	28	10
	.28	166	-	8	**6**	20	12
	.45	89	-	3	**2**	13	9
diabetes	.05	-	-	12	13	**10**	11
	.07	-	-	**13**	14	15	15
	.09	987	-	**12**	13	13	13
haberman	.02	5	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.03	5	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
	.05	4	-	$<$ **1**	$<$ **1**	$<$ **1**	$<$ **1**
ionosphere	.20	424	-	39	31	41	**29**
	.28	98	-	30	**23**	42	25
	.36	60	-	**18**	20	41	40
wine	.02	-	-	272	256	218	**175**
	.03	-	-	**495**	545	682	686
	.04	-	-	**464**	468	691	707

Table 2: Runtime performance (in seconds) of the algorithms across different datasets and ϵ values for a fixed depth of two. Best values marked with bold and timeouts with '-'. Averaged over five runs and rounded. RobTree and Pure Search have a significantly lower runtime in most instances.

Algorithm	Wins	Timeouts
LSU-MaxSAT	0	6
MILP-warm	0	21
Pure Search	**14**	**0**
Pure Search-warm	13	0
RobTree	13	0
RobTree-warm	12	0

Table 3: Results aggregated. Pure Search has the most amount of wins and no timeouts.

[25] in both our speed and anytime performance benchmarks. This ensures that both algorithms are evaluated on datasets where ROCT has demonstrated state-of-the-art performance, allowing for a fair and meaningful comparison. A summary of the datasets from OpenML[1] is provided in Table 1.

Although testing accuracy is not the primary focus of our experiments, we adopt the same 80%-20% train-test split used by Vos et al. [25] to report RobTree's accuracy on test data.

Experiment Conditions All experiments were conducted on a machine with an Intel(R) Core™ i5-9300H CPU (4 cores) with 8GB RAM, running four experiments in parallel. Each algorithm was ran on each dataset for the varying values of ϵ for a maximum of 30 minutes. In case of a timeout, the best tree found so far by the algorithm was used to calculate the adversarial accuracy. The experiment setup and the algorithm can be found on GitHub.[2]

Runtime The runtime of each algorithm for each dataset and ϵ pair is shown in Table 2, with aggregate results summarized in Table 3. A striking observation is that the MILP-based method frequently times out and fails to prove optimality in most cases. Even when MILP-warm does not timeout, it exhibits runtimes that are orders of magnitude higher than those of the other algorithms.

Both RobTree and Pure Search solve the instances well within the time limit. In fact, in all but a few instances, both variants complete in a fraction of the time required by the other algorithms. As noted by Mazumder et al. [17], the Quant-BnB algorithm exhibits poor scalability with respect to the number of features. Our results suggest that a similar limitation may apply to RobTree in the robustness setting, as both `ionosphere` and `cylinder-bands` contain more than 30 features, leading to RobTree performing close/worse than Pure Search in those instances.

[1] www.openml.org

[2] https://github.com/sandemiroren1/robtree

However, RobTree scales well with the number of unique values per feature compared to the other state-of-the-art algorithms. As Table 2 demonstrates, in the datasets with a high number of unique feature values, such as `banknote-authentication`, `diabetes` and `wine`, RobTree and Pure Search are the only algorithms that do not timeout. This indicates that LSU and MILP-warm scale poorly with the number of variables, which increases with the size of the dataset while RobTree runs in a fraction of the time limit.

Interestingly, Pure Search appears to outperform RobTree on the largest dataset, `wine`, particularly for larger values of ϵ. This behavior may be attributed to the fact that, as ϵ increases, the selected thresholds tend to produce increasingly similar results. Consequently, RobTree could be struggling to effectively prune the search space, wasting significant time computing lower bounds.

Warm Starting As evident in Table 2, for RobTree warm starting does seem to provide a speedup, especially for datasets with a high number of features such as `cylinder-bands`. This can be attributed to the framework being able to prune features from candidate trees early on due to the initially tight upper bound. However, for Pure Search warm starting does not seem as effective as the achieved runtimes are similar. This result seems to indicate that the pruning strategies used in our Pure Search algorithm do not rely on a tight upper bound as much as RobTree.

Anytime Performance Similar to the models from ROCT, RobTree is an *anytime algorithm*, meaning it can be stopped early and return the best solution found up to that point. As shown in Figure 5, RobTree exhibits a steep reduction in training error alongside LSU-MaxSAT. However, it converges to near-optimal results earlier than LSU-MaxSAT. Pure Search, on the other hand, takes time to catch up to LSU-MaxSAT but still converges to a near-optimal result faster. It is also worth noting that Pure Search tends to converge to a near-optimal solution last. This is likely because it exhaustively searches all trees with a fixed root feature before considering others, limiting early exploration of diverse regions of the search space.

Additionally, when warm-started with a run of GROOT, RobTree and Pure Search begin with a solution that is already close to optimal, making them the first to reach near-optimal performance. This suggests that warm-starting significantly improves their anytime performance as they spend most of their runtime confirming optimality. In fact, RobTree and Pure Search are the only algorithms to achieve optimal adversarial training accuracy as seen in Table 4. Interestingly, warm-starting does not seem to have a similar effect on MILP-warm's upper bound over time. Upon contact with the authors, this was attributed to MILP-warm's implementation. Last, despite LSU-MaxSAT not always achieving optimality, its mean training accuracy differs only slightly from the optimal value.

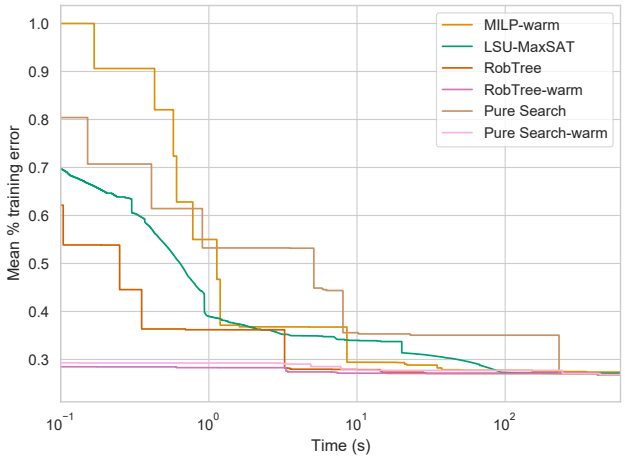


Figure 5: Anytime performance of the algorithms. LSU-MaxSAT and RobTree see a steep increase in their accuracy early on, and the algorithms later catch up. The graph was obtained by running each algorithm on each dataset once for an ϵ corresponding to %50 of the adversarial range and then averaging the results.

Algorithm	Mean adversarial accuracy	Optimal Results
GROOT	0.709	2
LSU-MaxSAT	0.733	21
MILP-warm	0.730	18
RobTree/Pure Search	**0.733**	**24**

Table 4: Aggregate results on adversarial accuracy (in range $[0, 1]$). LSU-MaxSAT and MILP-warm occasionally fail to achieve optimal training accuracy.

7 Responsible Research

Reproducibility We provide our code on Github.[3] Our repository not only includes our algorithm, but it also includes our testing environment with detailed documentation on how to conduct the experiments at home. The datasets are free to use and are linked in the paper as well, following the FAIR paradigm. Moreover, in this paper we thoroughly describe the conditions and parameters used in conducting the experiment. We believe that similar results can be achieved by following the same methodology on different hardware. One limitation of our setup is that while it runs out of the box on Linux, extra steps are needed to recreate it exactly on Windows and macOS. Providing out of the box alternatives for those operating systems could improve reproducibility. Another limitation would be that GUROBI is licensed software, so our results would not be possible without the acquisition of a license.

Integrity of Results We test our algorithm on datasets and parameters used in previous literature. Hence, we believe our experiment setup was not biased towards arriving at a specific result. However, we did not do a deep discussion of the

[3] https://github.com/sandemiroren1/robtree

choice of datasets and whether the epsilon values in previous results are fitting for our use. Hence, any bias in previous work would be reflected in our work as well. Moreover, our research could have some bias towards the implementation specifics of the algorithms, as differences in implementation can result in varying results for the same algorithm. Thus, our results may be biased towards RobTree due to the emphasis placed on its implementation being optimized. Last, LLMs were not used to generate ideas, code, or original text in the report. We believe this makes our work not impacted by biases that an LLM can introduce.

8 Conclusions and Future Work

We present RobTree, a scalable branch and bound algorithm for robust optimal decision trees. Taking close inspiration from the work of Mazumder et al. [17] we adapt the Quant-BnB algorithm to a setting with an adversary with a box-shaped attack model.

RobTree outperforms state-of-the-art methods by up to two orders of magnitude in runtime. Moreover, the exhaustive search subroutine we use, Pure Search has shown itself to be scalable for depth two on its own. We believe that the theorems provided in this paper and the various adaptations of pre-existing pruning techniques should pave the way for more search-based methods for robustness. Furthermore, the state-of-the-art can be sped up drastically with the theorem we propose.

Future work could be investigating other lower bounds to be used in the algorithm. Extension to depth three and adapting the algorithm to be more scalable in terms of the number of features could also be a next step. Deeper analysis of various combination of pruning methods could also provide useful insights.

References

[1] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of AAAI-20*, pages 3146–3153, 2020.

[2] Víctor Blanco, Alberto Japón, and Justo Puerto. Robust optimal classification trees under noisy labels, 2020.

[3] Leo Breiman et al. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[4] Cătălin E Briţa, Jacobus GM van der Linden, and Emir Demirović. Optimal classification trees for continuous feature data using dynamic programming with branch-and-bound. In *Proceedings of the AAAI-25*, volume 39, pages 11131–11139, 2025.

[5] Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: Training evasion-aware decision trees, 2019.

[6] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 1122–1131. PMLR, 2019.

[7] David Guichard. An Introduction to Combinatorics and Graph Theory. https://www.whitman.edu/mathematics/cgt_online/book/section03.05.html. Accessed: 2025-06-02.

[8] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[9] Emir Demirović, Christian Schilling, and Anna Lukina. In search of trees: Decision-tree policy synthesis for black-box systems via search. In *Proceedings of the AAAI-25*, number 26, pages 27250–27257, 2025.

[10] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine learning in finance*. Springer, 2020.

[11] Jack Good, Torin Kovach, Kyle Miller, and Artur Dubrawski. Feature learning for interpretable, performant decision trees. *Advances in Neural Information Processing Systems*, 36:66571–66582, 2023.

[12] Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of global optimization*, 81:233–260, 2021.

[13] Jun-Qi Guo, Ming-Zhuo Teng, Wei Gao, and Zhi-Hua Zhou. Fast provably robust decision trees and boosting. In *International Conference on Machine Learning*, pages 8127–8144. PMLR, 2022.

[14] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *Advances in neural information processing systems*, 32, 2019.

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

[16] Nathan Justin, Sina Aghaei, Andres Gomez, and Phebe Vayanos. Optimal robust classification trees. In *The aaai-22 workshop on adversarial machine learning and beyond*, 2021.

[17] Rahul Mazumder, Xiang Meng, and Haoyue Wang. Quant-bnb: A scalable branch-and-bound method for optimal decision trees with continuous features. In *Proceedings of ICML-22*, pages 15255–15277, 2022.

[18] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

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

[20] J. Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[21] Harry Surden. Machine learning and law: An overview. *Research handbook on big data law*, pages 171–184, 2021.

[22] Jacobus G. M. van der Linden, Daniël Vos, Mathijs M. de Weerdt, Sicco Verwer, and Emir Demirović. Optimal or greedy decision trees? revisiting their objectives, tuning, and performance, 2025.

[23] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In *Proceedings of ICML-21*, pages 10586–10595, 2021.

[24] Daniël Vos and Sicco Verwer. Adversarially robust decision tree relabeling. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 203–218. Springer, 2022.

[25] Daniël Vos and Sicco Verwer. Robust optimal classification trees against adversarial examples. In *Proceedings of AAAI-22*, pages 8520–8528, 2022.

A Test Accuracy of RobTree

Overall, as it can be seen in Table 6 and Table 5, RobTree achieves state-of-the-art results on test data, having the most amount of wins. Further research can be done in the future on determining the thresholds to be a certain distance away from the sides of the box depicting the attacker's perturbation range.

B Training Accuracy of RobTree

RobTree and Pure Search achieve optimal results in all instances as depicted in Table 7. LSU-MaxSAT, despite not always delivering optimal results, delivers values that are only slightly lower than optimal, while MILP-warm on average is 3% off.

C Dropping and Combining Data Points

We show the rest of theorems that were used in the paper but not proven in the main text. We use similar proof ideas to Mazumder et al. [17]. We show the following theorem, which is used in various proofs.

Theorem 4. *For any D and $x_1, x_2 \in [a, b]$ such that $x_1 < x_2$ it holds that $L_2(D^{f_0}_{[0,x_1]} \cup D^{f_0}_{[x_2,b]}, \phi, S) \geq L_1(D^{f_0}_{[a,x_1]}, F_1, S) + L_1(D^{f_0}_{[x_2,b]}, F_2, S)$*

Proof. Let t^* be the split threshold of an optimal tree that can be created from ϕ for a dataset D. Let $\mathcal{C}$ denote the data points that can be directed to both subtrees from the root by an adversary. It holds that each element in $\mathcal{C}$ is in one of the following sets: $D^{f_0}_{[a,x_1]}, D^{f_0}_{[x_2,b]}$. By separating the data points in D to the aforementioned two sets, both subtrees have data points that are disjoint. Hence all the data points in $\mathcal{C}$ are directed to one side rather than both. All other data points are unaffected. □

D Proof of the Tighter Lower Bound

We have shown that W^S_0 is a lower bound on the true optimal error L_2. We also show that $W^S_{s'}$ is a lower bound for L_2.

Theorem 5. *$W^S_{s'}$ is a lower bound for L_2.*

Proof. Take an arbitrary candidate tree $\phi = (f_0, [a, b], F_1, F_2)$ and a set of data points D. It is trivial to see that $L_2(D^{f_0}_{[0,a]} \cup D^{f_0}_{[b,u(f_0)]}, \phi, S) + L_2(D^{f_0}_{[a,b]}, \phi, S) \leq L_2(D, \phi, S)$. Let $t_0, \ldots, t_{s'}$ be almost s'-equi-spaced in $[a, b]$ We show that $\min_{1 \leq j \leq t} W^S_0(D^{f_0}_{[a,b]}, \phi_j) \leq L_2(D^{f_0}_{[a,b]}, \phi, S)$ for $\phi_j = (f_0, [t_{j-1}, t_j], F_1, F_2)$. Let $j^* \in [a, b]$ be the indice such that $L_2(W^S_{[a,b]}, \phi, S) = L_2(W^S_{[a,b]}, (f_0, [t_{j^*}, t_{j^*}], F_1, F_2), S)$ i.e the optimal split threshold for the inner part. Then by definition there exists a value $j' \in [1, s']$ such that $t_{j'-1} \leq t_{j^*} \leq t_{j'}$. Moreover we can see that

$$
\begin{aligned}
&W^S_0(D^{f_0}_{[a,b]}, (f_0, [t_{j'-1}, t_{j'}], F_1, F_2)) \\
&= L_1(D^{f_0}_{[a,t_{j'-1}]}, F_1, S) + L_1(D^{f_0}_{[t_{j'},b]}, F_2, S) \\
&\leq L_1(D^{f_0}_{[a,t_{j^*}]}, F_1, S) + L_1(D^{f_0}_{[t_{j^*},b]}, F_2, S) \\
&\leq L_2(D^{f_0}_{[a,t_{j^*}]} \cup D^{f_0}_{[t_{j^*},b]}, \phi, S) \\
&\leq L_2(D^{f_0}_{[a,b]}, \phi, S)
\end{aligned}
$$

Moreover we know that $\min_{1 \leq j \leq t} W^S_0(D^{f_0}_{[a,b]}, \phi_j) \leq W^S_0(D^{f_0}_{[a,b]}, (f_0, [t_{j'-1}, t_{j'}], F_1, F_2))$ Hence we know that $\min_{1 \leq j \leq t} W^S_0(D^{f_0}_{[a,b]}, \phi_j) \leq L_2(D^{f_0}_{[a,b]}, \phi, S)$. Hence $W^S_{s'}$ is a lower bound for L_2. □

Algorithm	Mean adversarial accuracy	Standard error adversarial accuracy	Mean rank	Standard error rank	Number of wins
GROOT	0.715	0.013	4.375	0.561	7
LSU-MaxSAT	0.734	0.014	2.667	0.384	10
MILP-warm	0.727	0.014	3.042	0.440	10
Pure Search	0.729	0.014	1.792	0.199	13
Pure Search-warm	0.729	0.014	1.792	0.199	13
RobTree	0.730	0.014	1.792	0.248	14
RobTree-warm	0.730	0.014	1.792	0.248	14

Table 5: Aggregate test scores, RobTree has the most wins.

			ROCT		Ours			
Dataset	ϵ	GROOT	LSU MaxSAT	MILP warm	Pure Search	Pure Search warm	RobTree	RobTree warm
banknote-authentication	.07	0.731	0.796	0.796	0.800	0.800	**0.804**	**0.804**
	.09	0.669	0.720	0.705	**0.724**	**0.724**	**0.724**	**0.724**
	.11	0.618	0.640	0.640	**0.644**	**0.644**	**0.644**	**0.644**
blood-transfusion	.01	0.700	0.733	0.733	**0.740**	**0.740**	0.693	0.693
	.02	0.760	0.767	0.767	**0.787**	**0.787**	0.780	0.780
	.03	0.760	0.760	0.760	0.760	0.760	**0.767**	**0.767**
breast-cancer	.28	0.832	**0.854**	**0.854**	**0.854**	**0.854**	**0.854**	**0.854**
	.39	0.781	**0.818**	**0.818**	**0.818**	**0.818**	**0.818**	**0.818**
	.45	0.737	**0.774**	**0.774**	**0.774**	**0.774**	**0.774**	**0.774**
cylinder-bands	.23	**0.714**	**0.714**	**0.714**	**0.714**	**0.714**	**0.714**	**0.714**
	.28	**0.750**	0.679	0.679	0.679	0.679	0.679	0.679
	.45	**0.696**	0.679	0.679	0.679	0.679	0.679	0.679
diabetes	.05	0.714	0.721	**0.747**	0.714	0.714	0.714	0.714
	.07	**0.682**	0.656	0.623	0.656	0.656	0.656	0.656
	.09	0.610	**0.649**	0.604	0.604	0.604	0.604	0.604
haberman	.02	0.726	**0.806**	0.726	0.742	0.742	0.742	0.742
	.03	0.710	**0.790**	0.742	0.742	0.742	**0.790**	**0.790**
	.05	0.726	0.677	**0.742**	0.677	0.677	0.677	0.677
ionosphere	.20	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**
	.28	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**	**0.817**
	.36	0.732	**0.775**	**0.775**	**0.775**	**0.775**	**0.775**	**0.775**
wine	.02	**0.675**	0.672	**0.675**	0.673	0.673	0.673	0.673
	.03	0.599	0.659	0.633	**0.662**	**0.662**	**0.662**	**0.662**
	.04	0.600	0.652	0.633	**0.657**	**0.657**	**0.657**	**0.657**

Table 6: The adversarial accuracy of the trees generated by each algorithm on the test data. RobTree still gives state-of-the-art results.

			ROCT		Ours			
Dataset	ϵ	GROOT	LSU MaxSAT	MILP warm	Pure Search	Pure Search warm	RobTree	RobTree warm
banknote-authentication	.07	0.718	**0.789**	**0.789**	**0.789**	**0.789**	**0.789**	**0.789**
	.09	0.664	**0.720**	0.700	**0.720**	**0.720**	**0.720**	**0.720**
	.11	0.613	**0.655**	**0.655**	**0.655**	**0.655**	**0.655**	**0.655**
blood-transfusion	.01	0.766	**0.786**	**0.786**	**0.786**	**0.786**	**0.786**	**0.786**
	.02	0.773	**0.774**	**0.774**	**0.774**	**0.774**	**0.774**	**0.774**
	.03	0.771	**0.773**	**0.773**	**0.773**	**0.773**	**0.773**	**0.773**
breast-cancer	.28	0.844	**0.868**	**0.868**	**0.868**	**0.868**	**0.868**	**0.868**
	.39	0.766	**0.810**	**0.810**	**0.810**	**0.810**	**0.810**	**0.810**
	.45	0.725	**0.751**	**0.751**	**0.751**	**0.751**	**0.751**	**0.751**
cylinder-bands	.23	0.706	**0.715**	**0.715**	**0.715**	**0.715**	**0.715**	**0.715**
	.28	0.688	**0.701**	**0.701**	**0.701**	**0.701**	**0.701**	**0.701**
	.45	0.688	**0.701**	**0.701**	**0.701**	**0.701**	**0.701**	**0.701**
diabetes	.05	0.663	**0.686**	**0.686**	**0.686**	**0.686**	**0.686**	**0.686**
	.07	0.625	**0.668**	0.664	**0.668**	**0.668**	**0.668**	**0.668**
	.09	0.660	**0.661**	**0.661**	**0.661**	**0.661**	**0.661**	**0.661**
haberman	.02	0.762	**0.766**	0.762	**0.766**	**0.766**	**0.766**	**0.766**
	.03	0.750	**0.758**	**0.758**	**0.758**	**0.758**	**0.758**	**0.758**
	.05	0.701	**0.738**	**0.738**	**0.738**	**0.738**	**0.738**	**0.738**
ionosphere	.20	**0.779**	**0.779**	**0.779**	**0.779**	**0.779**	**0.779**	**0.779**
	.28	**0.764**	**0.764**	**0.764**	**0.764**	**0.764**	**0.764**	**0.764**
	.36	0.732	**0.743**	**0.743**	**0.743**	**0.743**	**0.743**	**0.743**
wine	.02	0.669	0.673	0.669	**0.679**	**0.679**	**0.679**	**0.679**
	.03	0.586	0.656	0.633	**0.658**	**0.658**	**0.658**	**0.658**
	.04	0.599	0.647	0.633	**0.649**	**0.649**	**0.649**	**0.649**

Table 7: Training accuracies of each algorithm on each dataset epsilon pair. RobTree and Pure Search are the only algorithms to always give an optimal result

Algorithm	Mean adversarial accuracy	Standard error adversarial accuracy	Mean rank	Standard error rank	Number of wins
GROOT	0.709	0.013	6.417	0.345	2
LSU-MaxSAT	0.733	0.012	1.500	0.276	21
MILP-warm	0.730	0.012	2.250	0.451	18
Pure Search	0.733	0.012	1.000	0.000	24
Pure Search-warm	0.733	0.012	1.000	0.000	24
RobTree	0.733	0.012	1.000	0.000	24
RobTree-warm	0.733	0.012	1.000	0.000	24

Table 8: Aggregated training accuracies. RobTree and Pure Search posses best results.