Versatile Verification of Tree Ensembles

Laurens Devos [1] **Wannes Meert** [1] **Jesse Davis** [1]

Abstract

Machine learned models often must abide by certain requirements (e.g., fairness or legal). This has spurred interested in developing approaches that can provably verify whether a model satisfies certain properties. This paper introduces a generic algorithm called VERITAS that enables tackling multiple different verification tasks for tree ensemble models like random forests (RFs) and gradient boosted decision trees (GBDTs). This generality contrasts with previous work, which has focused exclusively on either adversarial example generation or robustness checking. VERITAS formulates the verification task as a generic optimization problem and introduces a novel search space representation. VERITAS offers two key advantages. First, it provides anytime lower and upper bounds when the optimization problem cannot be solved exactly. In contrast, many existing methods have focused on exact solutions and are thus limited by the verification problem being NP-complete. Second, VERITAS produces full (bounded suboptimal) solutions that can be used to generate concrete examples. We experimentally show that our method produces state-of-the-art robustness estimates, especially when executed with strict time constraints. This is exceedingly important when checking the robustness of large datasets. Additionally, we show that VERITAS enables tackling more real-world verification scenarios.

1. Introduction

Currently, it is becoming more common for deployed machine learned models to conform to requirements (e.g., legal) or exhibit specific properties (e.g., fairness). This has motivated the development of verification approaches that are applicable to learned models. Given a specific property, these techniques verify, that is, prove whether or not the property holds. Examples of verification questions include:

- **Adversarial example generation:** Given a data example, can slightly perturbing it cause its predicted label to flip? (Szegedy et al., 2013; Goodfellow et al., 2014; Einziger et al., 2019)
- **Robustness checking:** Given a data example, what is the minimum distance to such an adversarial example? (Carlini & Wagner, 2017; Ranzato & Zanella, 2020)
- **Feature dominance:** Given a set of constraints describing a class of data examples of interest, can we find one or more attributes where changing their values would disproportionately affect the model's prediction?
- **Fairness:** Do two instances exist that differ only on their protected attributes (e.g. sex, race, age) or a proxy variable, but have different predicted labels? (Dwork et al., 2012)

If a property is violated, it is desirable to also return constructed counter examples. For instance, if an adversarial example exists it is useful to see one concrete instance as this could, for example, be added to the training set.

The popularity of additive tree ensembles (e.g., random forests (Breiman, 2001) and gradient boosted tree models (Chen & Guestrin, 2016; Ke et al., 2017; Devos et al., 2019)) has motivated interest in developing verification techniques for this model class. Most current work is characterized by its focus on (1) solving one specific type of verification problem (e.g., adversarial example generation (Einziger et al., 2019) or robustness checking (Chen et al., 2019b; Törnblom & Nadjm-Tehrani, 2020; Ranzato & Zanella, 2020)) and (2) developing exact solutions. While exact solutions are desirable, in practice they are not always feasible because verification tasks on additive ensembles are NP-complete (Kantchelian et al., 2016).

One recent approximate approach is due to Chen et al. (2019b), who proposed a graph-based approach that computes an upper bound rather than a full solution if time and space limits are exceeded. However, this approach has three important limitations. First, the search steps are very coarse-grained, and the algorithm often terminates after only a few steps. Second, the algorithm is not guided by a heuristic,

[1]Department of Computer Science, KU Leuven, Leuven, Belgium. Correspondence to: Laurens Devos <laurens.devos@kuleuven.be>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

so a selected step may not improve the bound. Practically, this translates into the algorithm generating looser bounds. Third, the approach only is able to generate a (counter) example if the search terminates.

This paper introduces a generic anytime algorithm called VERITAS – **Veri**fication of **T**rees using **A**nytime **S**earch – for addressing multiple different types of verification tasks on tree ensembles. This contrasts with existing work that focuses on a single verification task. The key insight underlying VERITAS is that a large class of verification problems can be posed as a generic constrained optimization problem. We propose a novel search space for which an admissible heuristic exists, which confers two advantages. First, it provides anytime upper and lower bounds. Second, it can generate a concrete example using the current lower bound. Empirically, on robustness checking, VERITAS finds better quality approximations than its competitor while still often being faster. Moreover, it typically results in order of magnitude speedups compared to exact approaches. Furthermore, we demonstrate VERITAS's versatility in terms of being able to address multiple verification tasks. First, we show several illustrative examples on a feature dominance task. Finally, we highlight how VERITAS can give insights into a learned ensemble's behavior on two use cases from soccer analytics. VERITAS is available as an open-source package.[1]

2. Preliminaries

2.1. Additive Tree Ensembles

The framework described in this paper reasons about **additive ensembles of binary trees**.[2] A binary tree T consists of two types of nodes. An *internal node* stores a split condition and references to a left and a right child node. The split condition is a less-than comparison $X < \tau$ defined on an attribute X for some threshold τ. Splits on binary attributes are possible using $\tau = 0.5$. A *leaf node* has no children and simply stores an output value ν called the **leaf value**. The *root node* is the only node that has no parent.

Given a data example $\boldsymbol{x}$ from the input space $\mathcal{X}$, a tree is evaluated by recursively traversing it starting from the root node. For internal nodes, the node's split condition is tested on $\boldsymbol{x}$; if the test succeeds, the procedure is recursively applied to the left child node, else, it is applied to the right child node. If a leaf node is encountered, the leaf value is returned and the procedure terminates.

The **box** of leaf l, denoted $\mathrm{box}(l)$, defines a hypercube of the input space by conjoining all split condition from the root to l. All data examples $\boldsymbol{x} \in \mathrm{box}(l)$ evaluate to leaf l. For example, $\mathrm{box}(l_2^2)$ in Figure 1 equals $\{\mathrm{Age} < 50, \mathrm{Height} \geq 200\}$. We also define the definition of box for a set of leafs: $\mathrm{box}(l^1, \ldots, l^M) = \bigcap_m \mathrm{box}(l^m)$. Two leafs **overlap** when the intersection of their boxes is non-empty. For example, in Figure 1, leaf l_2^2 overlaps with l_1^1, but does not with l_3^3.

An additive ensemble of trees is a sum of trees $\boldsymbol{T} = T^1 + \cdots + T^M$. We use l_i^m to denote the ith leaf of tree T^m. Given a data example $\boldsymbol{x}$, $\boldsymbol{T}$ evaluates to the sum of the evaluations of all trees. The set of leaf nodes whose leaf values contribute to the output of the ensemble is called an **output configuration** of the ensemble. The box of an output configuration is by definition non-empty. For all data examples $\boldsymbol{x}$ in an output configuration's box, the output of the ensemble is fixed because the same leafs are activated each time. For example, in Figure 1, $\mathrm{box}(l_1^1, l_1^2, l_1^3) = \{\mathrm{Age} < 40, \mathrm{Height} < 200, \mathrm{BMI} < 28\}$. All data examples in that box will evaluate to $\nu_1^1 + \nu_1^2 + \nu_1^3$.

2.2. MERGE: Robustness Verification

Chen et al. (2019b) present a method for robustness verification of tree ensembles. Their algorithm produces a lower bound $\underline{\delta}$ on the distance to the closest adversarial example for a particular example $\boldsymbol{x}$ and uses a 10-step binary search. Given a model $\boldsymbol{T}$, a data example $\boldsymbol{x}$ with predicted label $\boldsymbol{T}(\boldsymbol{x})$, and a maximum distance δ, they verify whether an example $\tilde{\boldsymbol{x}}$ in the area of $\boldsymbol{x}$, $||\boldsymbol{x} - \tilde{\boldsymbol{x}}||_\infty < \delta$, exists that flips the predicted label: $\boldsymbol{T}(\tilde{\boldsymbol{x}}) \neq \boldsymbol{T}(\boldsymbol{x})$. If such an example exists, the area surrounding $\boldsymbol{x}$ is shrunk by decreasing δ, else, the area is expanded by increasing δ. This is repeated 10 times or until enough precision on $\underline{\delta}$ is obtained.

To prove that no $\tilde{\boldsymbol{x}}$ exists with a flipped label within a distance δ from $\boldsymbol{x}$, Chen et al. use a graph representation of the ensemble model and a *merge* procedure defined on the independent sets[3] of the graph to compute an upper bound on the ensemble's output. When the upper bound is less than zero, it is impossible for the ensemble to output the positive class for any example in the area surrounding $\boldsymbol{x}$.

The graph $\boldsymbol{G}$ has one vertex for each leaf l_i^m of each tree T^m in ensemble $\boldsymbol{T}$. Two vertexes are connected by an edge when their boxes overlap. Figure 1 shows an ensemble and its corresponding graph representation. When verifying robustness, only the neighborhood surrounding an example $\boldsymbol{x}$ is considered. This is accomplished by including only the leafs that are accessible by examples close to $\boldsymbol{x}$. Specifically, only leafs l_i^m for which a $\tilde{\boldsymbol{x}} \in \mathrm{box}(l_i^m), ||\boldsymbol{x} - \tilde{\boldsymbol{x}}||_\infty < \delta$ exists are included in the graph. For example in Figure 1, with $\delta = 5$ and $\boldsymbol{x} = \{\mathrm{Age}\!: 60, \mathrm{Height}\!: 180, \mathrm{BMI}\!: 21\}$, only leafs l_2^1, l_1^2, and l_1^3 are included.

The graph has two key properties: (1) there is a one-to-one correspondence between the trees in the ensemble and the in-

[1] https://github.com/laudv/veritas

[2] Note that all trees can be represented as binary trees.

[3] An independent set is a subset of vertexes such that no two vertexes in the set are connected by an edge.

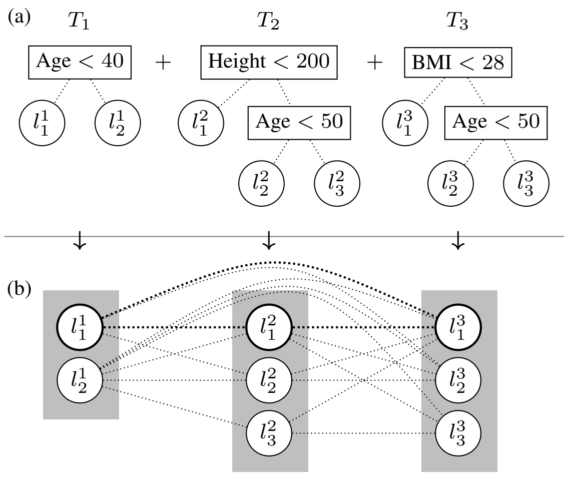

Figure 1. An example ensemble with (a) three trees and (b) its multipartite graph transformation $\boldsymbol{G}$, with l_i^m the ith leaf of tree T_m. The graph consists of $M = 3$ independent sets, one for each tree, and are denoted by a gray rectangular background. Edges in $\boldsymbol{G}$ connect leafs that have overlapping boxes. A max-clique contains exactly one vertex from each independent set (tree), and corresponds to an output configuration of the ensemble. An example max-clique in $\boldsymbol{G}$ is $[l_1^1, l_1^2, l_1^3]$ denoted by bold vertexes and bold edges.

dependent sets, and (2) there is a one-to-one correspondence between an output configuration and a max-clique.

Lemma 1 (Lemma 1, proof in (Chen et al., 2019b)**).** *A set of leafs is a max-clique in G iff it is an output configuration.*

Chen et al.'s main contribution is realizing that **merging** independent sets maintains the property in Lemma 1 and improves the upper bound on the ensemble's output $\bar{b}$:

$$\sum_m \max_i \nu_i^m, \tag{1}$$

where m ranges over the remaining independent sets and ν_i^m are the leaf values of the vertexes in independent set m.

The MERGE algorithm has three major drawbacks. First, it is coarse-grained: the number of merge *levels* or steps is at most $\lceil \log_2(M) \rceil$. The computational difficulty of each level grows exponentially, both in terms of time and memory, which means in practice that the algorithm stalls after only a two or three levels. This greatly limits the claimed *anytime* nature of MERGE. Second, it is not guided by a heuristic and most of its work does not improve the bound in Equation 1. Third, the algorithm only produces a full solution when it runs to completion, i.e., all independent sets are merged into one. In Chen et al.'s (Chen et al., 2019b) experiments, MERGE always produced better approximations than its competitors while being faster on 15 out of 18 reported cases. That is why we use MERGE as a baseline in our experiments.

3. VERITAS: Verification as a Constrained Optimization Problem

Consider the following optimization problems:

Robustness checking Let $\boldsymbol{T}$ be a binary classifier that classifies example $\boldsymbol{x}$ as negative. If $\max_{||\boldsymbol{x}-\tilde{\boldsymbol{x}}||_\infty < \delta} T(\tilde{\boldsymbol{x}}) < 0$, then no positively classified perturbed example can exist within distance δ from $\boldsymbol{x}$.

Fairness Suppose an insurance company estimates its clients' health scores using model $\boldsymbol{T}$ and wants to know which attributes affect the health score of middle-aged women the most. Given two middle-aged female individuals $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that differ only in a single attribute, the most dominant attribute is the one that maximizes $\boldsymbol{T}(\boldsymbol{x}_2) - \boldsymbol{T}(\boldsymbol{x}_1)$. The insurance company could take precautions if the most dominant attribute is a protected attribute, because that could be considered unfair.

Both these examples belong to a class of verification problems that can be modeled as **a generic optimization problem**: *find two examples that (1) satisfy the given constraints and (2) maximize the difference between the outputs of two models.* Formally, the optimization problem is:

$$\max_{\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}} \boldsymbol{T}_2(\boldsymbol{x}_2) - \boldsymbol{T}_1(\boldsymbol{x}_1) \quad \text{subject to} \quad \mathcal{C}(\boldsymbol{x}_1, \boldsymbol{x}_2). \tag{2}$$

where $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ are models, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are examples from the input space $\mathcal{X}$, and $\mathcal{C} : \mathcal{X}^2 \to \{\text{true}, \text{false}\}$ is a function defining constraints on the input space. In many cases, we only consider a single model $\boldsymbol{T}$. Taking $\boldsymbol{T}_1$ to be the trivial model that always predicts 0.0, and $\boldsymbol{T}_2 = \boldsymbol{T}$, yields the maximization problem:

$$\max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{T}(\boldsymbol{x}) \quad \text{subject to} \quad \mathcal{C}(\boldsymbol{x}). \tag{3}$$

Minimizing $\boldsymbol{T}$ is also possible by taking $\boldsymbol{T}_1 = \boldsymbol{T}$, and making $\boldsymbol{T}_2$ the trivial model. Note that we simplify $\mathcal{C}$ because the example of the trivial model does not affect the outcome.

The algorithm presented in this paper, VERITAS, is a search algorithm that solves the optimization problem in Equation 2. The algorithm operates in a novel search space inspired by Chen et al.'s graph representation $\boldsymbol{G}$. It is fine-grained as a single step is cheap, heuristically guided and does not waste time in irrelevant parts of the search space. The algorithm is anytime and outputs ever-improving upper bounds, lower bounds, and suboptimal full solutions as it is running. As more time and memory is provided, the bounds grow closer until the exact full solution is found. As this problem is NP-complete (Kantchelian et al., 2016), it might take a long time until the bounds converge to an exact solution. This makes the fine-grained anytime nature of VERITAS particularly compelling.

We will first introduce the search space. Then, we will show how this space is used to solve the optimization problem in Equation 2. For ease of explanation, we will first tackle the single-instance setting in Equation 3, and then extend our method to support the two-instance setting.

3.1. The Search Space S

The search is not executed directly in $\boldsymbol{G}$, but rather in a **sound and complete** search space S derived from $\boldsymbol{G}$. The states of S are cliques in $\boldsymbol{G}$ and are represented as sequences of leaf nodes. Starting from the initial empty sequence $[\,]$ at depth 0, the search space recursively builds up output configurations by adding leafs to the state in order, one leaf per tree. Specifically, a state $s = [l^1_{i_1}, \ldots, l^m_{i_m}]$ at search depth m is expanded to states $C(s)$ at depth $m+1$:

$$\begin{aligned} C([l^1_{i_1}, \ldots, l^m_{i_m}]) = \\ \{[l^1_{i_1}, \ldots, l^m_{i_m}, l^{m+1}] \mid l^{m+1} \in L^{m+1}, \\ \mathrm{box}(l^1_{i_1}, \ldots, l^m_{i_m}, l^{m+1}) \neq \emptyset\}, \end{aligned} \tag{4}$$

where L^{m+1} is the set of all leafs of tree T^{m+1}. The expand function C ensures that each expanded state is again a clique in G by allowing only states with non-empty boxes. When depth $m = M$ is reached, an output configuration, or equivalently a max-clique is found. The search space has two important properties.

Theorem 1. *Each state at depth $m = M$ corresponds to an output configuration (i.e., a max-clique in $\boldsymbol{G}$), and the search space enumerates all output configurations, that is, the enumeration is sound and complete.*

Proof. We show that S is *sound* and *complete*.

Soundness: We show that any state $s = [l^1_{i_1}, \ldots, l^M_{i_M}]$ is an output configuration. By definition of C in Equation 4, the box of the state is non-empty. For each data example $\boldsymbol{x}$ in the state's box, it holds that $\boldsymbol{x} \in \mathrm{box}(l^m_{i_m})$, $m = 1, \ldots, M$. Because of the properties of tree evaluation and the definition of box, tree T^m evaluates $\boldsymbol{x}$ to leaf $l^m_{i_m}$. Therefore, a data example $\boldsymbol{x}$ exists that activates all leaf nodes in s, and the configuration is valid. Using Lemma 1, it follows that s is a max-clique in $\boldsymbol{G}$.

Completeness: Assume an output configuration $O = \{l^1_{i_1}, \ldots, l^M_{i_M}\}$ so that $[l^1_{i_1}, \ldots, l^M_{i_M}]$ is not a state in the search space. Let $s = [l^1_{i_1}, \ldots, l^m_{i_m}] \subset O$ and $t = [l^1_{i_1}, \ldots, l^m_{i_m}, l^{m+1}_{i_{m+1}}] \subseteq O$ be sequences of leaf nodes such that s is a state in S, but t is not. It must be that $t \notin C(s)$. This can only be true when $\mathrm{box}(t) = \mathrm{box}(l^1_{i_1}, \ldots, l^m_{i_m}, l^{m+1}_{i_{m+1}}) = \bigcap_{m'=1}^{m+1} \mathrm{box}(l^{m'}_{i_{m'}}) = \emptyset$. This contradicts the fact that O is an output configuration. $\square$

3.2. Finding The Best State at Depth M

To solve the optimization problem in Equation 3, we need to find the state in S at depth M with the maximum output that adheres to the constraints $\mathcal{C}$. Constructing S explicitly and choosing the optimal state by enumerating all possibilities is intractable in practice. For that reason, we introduce an **admissible and consistent** heuristic to traverse the space in a best-first fashion. This allows us to only materialize the parts of S that are relevant to the problem.

The search is based on A*. It maintains an OPEN list of states in S and repeatedly *expands* the state in OPEN with the best f-value until a state at depth M is found. It removes the current best state $s = [l^1_{i_1}, \ldots, l^m_{i_m}]$ from OPEN and adds all its successors $C(s)$ to OPEN. The f-value is the sum of two values: $g(s)$, the sum of the leaf values of the leafs in the state, and the heuristic $h(s)$, an estimation of the remaining value to a state at depth M:

$$g(s) = \sum_{m'=1}^{m} \nu^{m'}_{i_{m'}}, \tag{5}$$

$$h(s) = \sum_{m'=m+1}^{M} h_{m'}(s), \tag{6}$$

$$\begin{aligned} h_{m'}(s) = \max\{\ \nu^{m'}_j \mid l^{m'} \in L^{m'}, \\ \mathrm{box}(l^1_{i_1}, \ldots, l^m_{i_m}, l^{m'}) \neq \emptyset\ \}. \end{aligned} \tag{7}$$

Intuitively, the heuristic $h(s)$ sums upper bounds $h_{m'}(s)$ for all remaining trees $T^{m'}$ for which no leaf has been added to the state yet. The upper bound $h_{m'}(s)$ is the maximum leaf value of any leaf of $T^{m'}$ that overlaps with all leafs in s. Note that we do not need to keep a VISITED list, because there are no cycles. We show that this heuristic leads us to the optimal solution next.

Theorem 2. *The heuristic search in S is guaranteed to find the max-clique in $\boldsymbol{G}$ which corresponds to the optimal output configuration of the optimization problem in Equation 3. (Proof in supplement.)*

3.3. Anytime Upper and Lower Bound Estimates

Depending on the size and difficulty of the problem, the search defined above might take a long time to find a solution. However, even when the search is terminated prematurely, the current best state in the OPEN list still contains useful information: because h is admissible, the current best f-value is an **upper bound** $\bar{b}$ on the optimal output.

To also produce a lower bound on the maximum output of the ensemble, we borrow ideas from Anytime Repairing A* (ARA*) (Likhachev et al., 2004). The importance of the heuristic in the f-value is reduced by ϵ, $0 < \epsilon \leq 1$ as follows:

$$f(s) = g(s) + \epsilon h(s). \tag{8}$$

This promotes deeper solutions, yielding full solutions much quicker. However, when $\epsilon < 1$, the solutions might no longer be optimal because $\epsilon h(s)$ is no longer an admissible heuristic. Nonetheless, the suboptimality is bounded by ϵ: for a suboptimal solution $\tilde{\boldsymbol{x}}$, the optimal solution's output is no larger than $\boldsymbol{T}(\tilde{\boldsymbol{x}})/\epsilon$. A suboptimal solution $\tilde{\boldsymbol{x}}$ is at least as good as the optimal solution, so $\boldsymbol{T}(\tilde{\boldsymbol{x}})$ is a **lower bound** $\underline{b}$ on the optimal output. While ϵ-tightness is a useful property of ARA*, we mainly use ARA* for its ability to produce suboptimal solutions quickly. We have found that, overall, A*'s upper bounds tend to converge faster and more smoothly, and are thus preferred over ARA*'s upper bounds.

Both the upper bound and the lower bound are anytime: The best f-value in the OPEN list is always accessible. The ϵ value in the relaxed f-score of Equation 8 can be gradually increased each time a suboptimal solution is found. Using ARA* means that it is not necessary to restart the search from scratch: the OPEN list can be reused, making the incremental increase of ϵ a cheap operation.[4]

3.4. Incorporating Constraints into the Search

A naive way of incorporating the constraint function $\mathcal{C}$ in Equations 2 and 3 is to ignore it during the search, and filter solutions returned by the search. This is inefficient, as the constraints often rule out large chunks of the search space.

For that reason, we lift $\mathcal{C}$ to the search state level. For example, considering the example in Figure 1 and the constraint Age > 60, we reject states $[l_1^1]$, $[l_2^1, l_2^2]$, and $[l_2^1, l_1^2, l_2^3]$.

Let $\mathcal{C}_s : S \to \{\text{true}, \text{false}\}$ be a function mapping the states of the search space to accept or reject. The function has the following two properties:

1. It is consistent with $\mathcal{C}$: a state s is accepted by $\mathcal{C}_s$ if and only if it is possible to find a data example $\boldsymbol{x}$ that sorts to the leafs in s and $\mathcal{C}(\boldsymbol{x})$ is true.

2. It is consistent across states: if $\mathcal{C}_s(s) = \text{false}$, and $t \supseteq s$ is a descendant from s, then $\mathcal{C}_s(t) = \text{false}$. In words: if a state is rejected by $\mathcal{C}_s$, then all its state expansions must also be rejected. This property ensures that we do not reject states that are the predecessor of a valid state.

Because all split conditions in the tree ensemble are simple linear constraints, the two properties are trivially satisfied. The constraint function $\mathcal{C}_s$ can be integrated into the search procedure by only adding accepted states to the OPEN list. For simple linear constraints, we can prune the graph $\boldsymbol{G}$ before the search starts, just like in the MERGE algorithm.

[4]Our case is more simple than the general case in (Likhachev et al., 2004) because we do not have cycles.

3.5. Maximizing the Difference Between Two Models

Now we explain the necessary adjustments to enable solving Equation 2. The search space is modified as follows: a state is extended to consist of two sequences of leafs, one sequence for each instance. The expand function alternately expands the first and the second instance using Equation 4. The f-value maximized by the search is updated to the difference between the f-values of the two instances:

$$\begin{aligned} f(s_1, s_2) &= f_2(s_2) - f_1(s_1) \\ &= g(s_2) - g(s_1) + h_2(s_2) - h_1(s_1). \end{aligned} \tag{9}$$

The g (Equation 5) and h_2 (Equation 6) remain unchanged. The first heuristic h_1 is an admissible and consistent heuristic with respect to the minimizing variant of Equation 3 obtained by replacing $\max$ by $\min$ in Equation 7. This makes $h_2 - h_1$ an admissible and consistent heuristic for the optimization problem in Equation 2 (proof in appendix).

3.6. Summarizing the Setup

We now have all necessary tools to answer verification questions. First, train a tree ensemble. Second, define the constraint function $\mathcal{C}_s$. The complexity of this step greatly depends on the verification question.[5] Third, give the learned model and $\mathcal{C}_s$ to VERITAS, which will produce a stream of decreasing upper bounds $\bar{b}$, and a stream of increasing lower bounds $\underline{b} = \boldsymbol{T}(\tilde{\boldsymbol{x}})$, with $\tilde{\boldsymbol{x}}$ suboptimal full solutions. Given infinite time and memory, the bounds converge: $\bar{b} = \underline{b}$, and the last $\tilde{\boldsymbol{x}}$ is the optimal solution.

3.7. Complexity Analysis

The worst-case complexity of VERITAS is $O(L^M)$, with L the average number of leafs per tree and M the number of trees. While this is the same as MERGE, VERITAS outperforms MERGE in practice. VERITAS only needs $m - 1$ smaller cliques to reach any m-clique, and will only explore cliques that challenge the current best output value based on the heuristic estimate. MERGE, on the other hand, only considers cliques at level l (of size T^l)[6] after having enumerated **all** cliques in the previous levels. The number of cliques per level grows exponentially. This allows VERITAS to more quickly visit larger cliques that have tighter output bounds while using less memory.

The worst-case complexity of $h(s)$ in terms of box overlap checks is $O(ML)$: the overlap between the box of s and the boxes of all leafs of the remaining trees is checked.

[5]In general, this step can be framed as a SAT problem: is it possible that an $\boldsymbol{x} \in \text{box}(s)$ exists that satisfies the constraints.

[6]T is a parameter of MERGE controlling the number of independent sets merged per level

4. Experimental Evaluation

The goal of our experiments is two-fold. First, we compare VERITAS to current approaches on the standard problem of checking the robustness of tree ensembles. Second, we want to highlight the versatility of VERITAS by looking at two general verification use cases that most existing methods cannot address: finding dominant attributes in a specific context, and asking domain specific questions about a model.

4.1. Verifying Robustness

Given an example $\boldsymbol{x}$, robustness checking tries to find the l_∞ distance δ to the closest adversarial example $\tilde{\boldsymbol{x}}$ such that the labels for $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ are different. That is, find the smallest δ such that $||\boldsymbol{x} - \tilde{\boldsymbol{x}}||_\infty < \delta$, and $\boldsymbol{T}(\boldsymbol{x}) \neq \boldsymbol{T}(\tilde{\boldsymbol{x}})$. Because this is a NP-hard problem, finding the optimal δ is often computationally expensive. This is particularly true if one wants to check all examples in a large training set. In our experiments, the average time to find the exact solution using MILP for reasonably large datasets ranges from around 10 seconds up to more than 2500 seconds per example. Given a training set of 100,000 examples, checking each one would yield times ranging from 11 days to 2890 days. Hence, approximations are needed which introduce a tradeoff between the time taken to produce an approximate solution and the quality of the found solution.

We compare VERITAS to the exact MILP approach (Kantchelian et al., 2016) and the highly performant approximate approach MERGE (Chen et al., 2019b) for robustness verification with the aim of addressing two questions:

Q1 How does each approach perform in terms of the time and solution quality tradeoff?

Q2 How does each approach's ability to produce a sufficiently accurate solution vary as a function of time?

4.1.1. DATASETS AND METHODOLOGY

We used the original author's implementation of MERGE. We used our own implementation of the MILP approach with Gurobi 9.1.1 (Gurobi Optimization, 2021) as the solver. The mathematical model is exactly as described in the original paper (Kantchelian et al., 2016). An overview of the parameters per dataset is given in the supplementary material. All experiments ran on an Intel(R) Xeon(R) CPU E3-1225 with 32GiB of memory. VERITAS's memory usage was restricted to 1GiB, and never used more than 150MiB. MERGE's memory limit was increased to 8GiB as it often failed to run with 4GiB of memory.

We compare on seven commonly used datasets for checking robustness (e.g., (Chen et al., 2019b)). All models were trained using XGBoost (Chen & Guestrin, 2016) using the

Table 1. Average l_∞-distance δ to the nearest adversarial example, the time t in seconds, speed-up factor with respect to MILP, and the standard deviation σ_t of time t. The results are aggregated over n examples. MILP produces exact results, whereas MERGE and VERITAS produce lower bounds. The δ values for MERGE and VERITAS are expressed as ratios $\delta^{\text{approx}}/\delta^{\text{exact}}$. Values closer to 100% are better. The best results are indicated with a *.

		MILP	MERGE	VERITAS	
covtype	δ	0.00796	56.9 %	99.2 %	*
$(n = 500)$	t	37.3	2.88	0.505	*
			13 ×	74 ×	
	σ_t	25.6	32.2	0.0737	*
f-mnist	δ	8.07	86.8 %	90.8 %	*
$(n = 900)$	t	85.2	0.253 *	6.47	
			337 ×	13 ×	
	σ_t	62	0.0874*	0.307	
higgs	δ	0.00183	46.3 %	88.1 %	*
$(n = 100)$	t	2640	2.87	1.45	*
			922 ×	1828 ×	
	σ_t	1560	14.3	0.292	*
ijcnn1	δ	0.0212	90 %	99.5 %	*
$(n = 500)$	t	9.31	2.82	0.33	*
			3 ×	28 ×	
	σ_t	1.96	25.9	0.0674	*
mnist	δ	10.4	83.5 %	85.5 %	*
$(n = 900)$	t	30.2	4.26	2.21	*
			7 ×	14 ×	
	σ_t	17.5	9.52	0.121	*
webspam	δ	0.00119	68.5 %	94.7 %	*
$(n = 500)$	t	36.2	1.17 *	1.34	
			31 ×	27 ×	
	σ_t	13.3	13.1	0.225	*
mnist2v6	δ	6.74	91.7 %	95.1 %	*
$(n = 500)$	t	0.66	0.0707*	1.6	
			9 ×	0 ×	
	σ_t	0.178 *	0.655	0.329	

same number of trees and tree depth as reported in (Chen et al., 2019b). We performed hyperparameter optimization to tune the learning rate. All details of the datasets and parameters are summarized in the supplementary materials. All datasets except MNIST and Fashion-MNIST were min-max-normalized. The robustness values for MNIST and Fashion-MNIST are pixel intensity values. Note that in boosted tree ensembles, classes are assigned as follows: for binary classification, the positive class is predicted when $\boldsymbol{T}(\boldsymbol{x}) > 0$; for multi-classification, multiple one-versus-all ensembles are trained, and the class with the highest weight is predicted.

4.1.2. Results

Table 1 shows the results for **Q1**. It shows the (approximate) distance to the nearest adversarial example as well as the average time in seconds and its standard deviation to perform robustness checking for each approach. MILP always returns an exact δ whereas VERITAS and MERGE return a lower bound. VERITAS always achieves a better average lower bound than MERGE, being never further than 15% away from the optimal value, and deviating by less than 5% on 4 out of 7 datasets. Being exact comes at a substantial cost in time for MILP: VERITAS is faster on six of the seven dataset. When VERITAS wins, it achieves massive time saving by being between 13 and 1828 times faster than MILP. Time wise, VERITAS is faster than MERGE on four datasets and tends to have a much smaller standard deviation.

To answer **Q2**, we explore how the fraction of examples that are verified varies as a function of time. We consider an example verified when (1) the δ value is produced within the time budget, and (2) the quality of the δ value is good enough. We measure the quality of the δ_i of task i as follows:

$$|\delta_i - \delta_i^{\text{exact}}| < \frac{1}{n} \sum_j |\delta_j^{\text{MERGE}} - \delta_j^{\text{exact}}|,$$

that is, the absolute error with respect to the optimal solution is less than the absolute error of MERGE averaged over all examples. As we use MERGE as the state-of-the-art reference algorithm, we also use MERGE's mean absolute error as a reference error value.

Figure 2 shows the results for this experiment. When one algorithm verifies more examples than another for a specific time budget t, its curve is above those of the other algorithms. VERITAS dominates MERGE for either all or the vast majority of the time budgets on five out of the six datasets. Moreover, it is quite difficult to control MERGE's run time, which is reflected in the straight-line behavior for larger time budgets.[7] For Fashion-MNIST, MNIST and webspam, MERGE can initially be faster than VERITAS. In its first level, MERGE is effective at removing many leaf value combinations without the overhead of a heuristic, improving the bounds quickly. However, as the number of cliques generated per level grows exponentially, the time it takes MERGE to expand and store all of them dwarfs the overhead of the heuristic later on. Finally, MILP can only verify more than 50% of the examples within 10 seconds on the ijcnn1 and MNIST2v6 datasets. For ijcnn1, this might be explained by the fact that a smaller model with only 60 trees is used. For MNIST2v6, which uses a 1000-tree model, the reason why MILP does so well is still unclear.

[7]The gentle curved increase for covertype and MNIST is due to variance on the run times.

4.1.3. Discussion

VERITAS produces better solutions than MERGE, and often does it in less time. Moreover, it is fine-grained: the user has full control over the run time of the algorithm, and VERITAS produces better approximations as it is given more time. Therefore the time-versus-approximation-quality trade-off can be effectively explored.

In contrast, while MERGE is generally fast, it does not offer fine-grained control. For example, for 10 random examples from the covertype dataset, MERGE takes 0.43 seconds on average using parameters $T = 2$ and $L = 2$. Changing L to 3 increases the average run time to 11 seconds. For $L = 4$, MERGE does not produce any results as it hits the 8GiB memory limit after 8.5 minutes of total run time. A parameter configuration that results in an average run time between 0.43 and 11 seconds simply does not exist.

We presented the MILP algorithm as an exact method, but in reality, MILP also produces anytime upper (UB) and lower bounds (LB). However, MILP's bounds tends to be very loose until right before it finds the optimal solution. Hence, they are generally not informative within the time ranges of MERGE and VERITAS. To illustrate this, we recorded the time needed for the gap in MILP's bounds to reach 20%, i.e., $\text{LB}/\text{UB} \geq 0.8$. Over all experiments in Table 1, MILP's total run time is 114 hours and stopping when a gap of 20% is reached would save a mere 12 minutes. In comparison, VERITAS's total run time is 2.7 hours and 82% of the time, the gap between its answer and the optimal solution is within 20%. Moreover, stopping MILP when it reaches a 20% gap versus waiting until the optimal solution only rarely results in a speed-up of more than 5%: 4.0% (covtype), 0.4% (f-mnist), 0.0% (higgs), 0.7% (mnist), 2.6% (ijcnn1), 0.2% (webspam), 2.6% (mnist2v6).

Finally, with state-of-the-art solvers like Gurobi (Gurobi Optimization, 2021), the MILP formulation performs better than previously reported. In some cases, the value of having exact results might outweigh the computational cost.

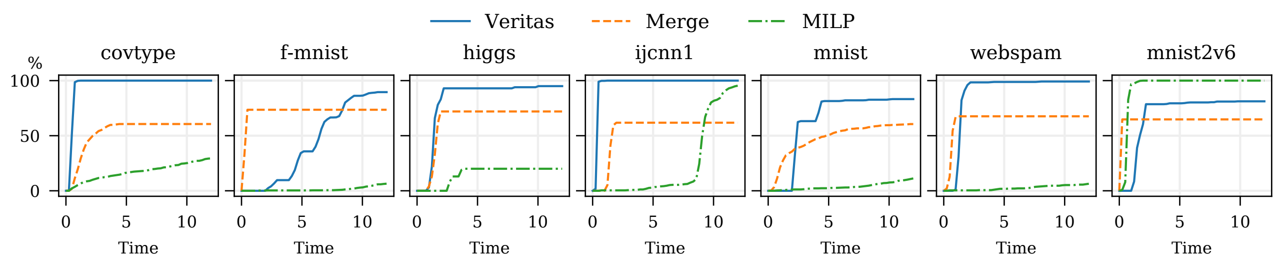

Figure 2. Fraction of instances that are verified as time progresses. An instance is considered verified when (1) a δ value is produced within the time limit, and (2) the absolute distance to the optimal robustness value (MILP) is less than the mean absolute error of MERGE.

4.2. Finding Dominant Attributes: YouTube

For this experiment we use a dataset generated from trending YouTube videos. The task is to predict the order of magnitude of views given a bag-of-words representation of the words used in the title and description of the video. We use a GBDT model with 100 trees of depth 10. Given a number of initial words, we ask VERITAS to produce k additional words such that the predicted view count is maximized. The state function $\mathcal{C}_s$ checks the *at-most-k* constraint. Some examples:

- *live, breaking, news, war* $\rightarrow$ adding words *big* and *trailers* increases the prediction by 2 orders of magnitude.
- *epic, challenge* $\rightarrow$ adding words *album*, *video*, and *remix* increases the prediction by almost 5 orders of magnitude.

This approach represents a generic strategy that allows reasoning about the (inflated) importance or dominance of one or more attributes. Other examples are fairness: given a set of constraints on the input space, maximize or minimize the output of the ensemble when only varying the values of one or more protected (proxy) attributes. If VERITAS finds examples of individuals that are treated significantly differently, then that might indicate unwanted model behavior.

4.3. Domain Specific Questions: Soccer

Another strength of our approach is that it can provide insights into learned models. To motivate this, consider the canonical task in soccer analytics of assigning a value to on-the-ball actions performed by players (Rudd, 2011; Decroos et al., 2019). These approaches value actions by estimating the probability of scoring in each game state, which is often done with a tree ensemble (Decroos et al., 2019). Then actions are valued by how much they increase this probability. An open question is in what situations is it useful to pass the ball backwards?[8] To answer this question for tree ensembles, one could look at all backward passes and simply select those assigned a positive value. What would be more useful would be to characterize situations where the model thinks passing the ball backwards would be useful. This is possible with VERITAS. To illustrate this capability, we analyze the action-value models and expected goals (xG) (Lucey et al., 2014) models. These models are trained using event stream data, which is a common source of data about professional soccer matches that is collected by having human annotators watch soccer matches and record information such as the location and time of on-the-ball events like passes and shots. The models used are similar in size and complexity to those used in professional soccer scouting software.

[8]This question was discussed on panel at sports analytics conference: https://www.youtube.com/watch?v=LA9-V6_ZIUg

Understanding Action-Value Models For this experiment we train a model that predicts the probability of scoring a goal in soccer within the next 10 actions (Decroos et al., 2019). The model has 126 trees (early stopping) of depth 10. The input for the model is two consecutive game states described by the position of the ball and the two action types (pass, shot, etc.).

We investigate two questions (1) *what ball action from the midfield will maximize the probability of scoring?* and (2) *in which contexts does a backwards pass increase the probability of scoring in the next 10 actions?* Because the action types are one-hot encoded, only one of these variables can be set to one in a valid instance. Therefore, to ensure that VERITAS generates legal instances we need to include a state constraint function $\mathcal{C}_s$ that imposes a *one-out-of-k* constraint for the one-hot-encoded action types.

Figure 3 shows two generated instances. The pass for the first question (shown in blue) shows an aggressive probing pass to a dangerous area near the goal. The pass for the second shows a cutback from the touchline to the center of the pitch. These types of passes often create a dangerous situation since they force the goalie to rapidly reposition themselves.

Understanding xG Models. We train a model that predicts

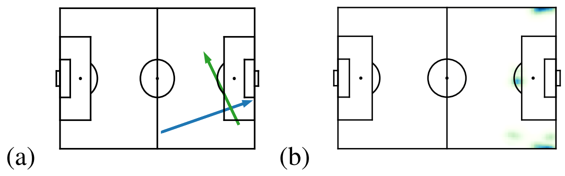

Figure 3. (a) The blue arrow shows the optimal position for a pass from the midfield to end. The green arrow shows the result for the backwards pass question: VERITAS generates a cut back with a positive goal probability. (b) A heatmap indicating where VERITAS generates instances with the highest goal probability.

the probability that a shot results in a goal using the publicly available toolset *soccer-xg* (Robberechts & Davis, 2020).[9] The model consists of 100 trees of depth 4.

We answer: what are the optimal locations to shoot from outside of the penalty box? We used VERITAS to generate 200 examples of shots from outside the penalty box that would have the highest chance of resulting in a goal. Figure 3.b is a heatmap showing the locations on the pitch for the instances generated by VERITAS. One cluster of instances is found on the edge of the box, directly in front of the center of the goal. This makes sense and corresponds to areas where it is advantageous for teams to shoot from (Van Roy et al., 2021). However, the instances generated near the corner spots are unexpected. Therefore we investigated the data. In the square 5m around the corner spot, there are 11 shots and 8 goals, which yields an extremely high 72% conversion rate. One possible explanation is to recall that this data is recorded by human annotators. If a player kicks the ball from the locations near the corner, the annotators are likely labeling the action as a pass or cross and are *only assigning an action the label of a shot in the unlikely event that it results in a goal or save*. This highlights how verification can identify unexpected patterns in the data, and hence provide insight into, e.g., how the data was collected and annotated.

5. Related work

A considerable amount of work has been done on verification of tree ensembles. Most work has focused on *adversarial attacks* (Einziger et al., 2019; Zhang et al., 2020) and robustness (Kantchelian et al., 2016; Chen et al., 2019b; Ranzato & Zanella, 2020; Törnblom & Nadjm-Tehrani, 2021). Törnblom & Nadjm-Tehrani introduced the VoTE framework, a system that enumerates all *equivalence classes* – sets of data examples that evaluate to the same output value, equivalent to the concept of output configurations in this paper – and checks whether some property holds. The properties that can be tested are general. However, the approach is limited by the number of equivalence classes, which quickly grows exponentially large (Törnblom & Nadjm-Tehrani, 2020).

Logical SMT theorem provers have also been used (Einziger et al., 2019; Sato et al., 2019; Devos et al., 2021), as have mixed-integer linear programming tools (Kantchelian et al., 2016). These approaches translate ensemble models to their respective languages and apply general purpose solvers to prove certain properties of the models. Others have applied tools from program analysis like *abstract interpretation* to verification of tree ensembles (Ranzato & Zanella, 2020; Drews et al., 2020; Calzavara et al., 2020; Törnblom & Nadjm-Tehrani, 2019; 2021). There is also work that focuses on learning robust models (Chen et al., 2019a; Calzavara et al., 2019). Rather than verifying the robustness of existing models, these methods build models that are less susceptible to adversarial attacks. Wang et al. show how the approach of Chen et al. can be extended to any p-norm. They show that the complexity of robustness checking can vary depending on the used norm for some models (Wang et al., 2020).

We did not compare to Silva (Ranzato & Zanella, 2020) and VoTE (Törnblom & Nadjm-Tehrani, 2021) because these systems do not estimate the distance to the closest adversarial example. Rather, they only check robustness for the easiest, most restrictive case (e.g. for MNIST, they only check $\exists \tilde{\boldsymbol{x}} : ||\boldsymbol{x} - \tilde{\boldsymbol{x}}||_\infty < 1$).

6. Conclusion

We introduced VERITAS, a tree ensemble verification tool that is capable of solving verification tasks that can be modeled as a generic optimization problem. It operates in a novel sound and complete search space and traverses that space using an admissible and consistent heuristic. VERITAS is the first fine-grained anytime algorithm to produce both an upper and a lower bound on the output of a tree ensemble model. Additionally, it also generates full suboptimal solutions that converge to the optimal solution when given enough time and memory. We empirically show that VERITAS outperforms the state of the art in terms of quality of the bounds and that for many tasks, VERITAS is also faster, and its run time can be controlled to a much finer degree.

Acknowledgements

LD is supported by Research Foundation-Flanders (FWO). This research received funding from KU Leuven Research Fund (C14/17/070), Research Foundation-Flanders under EOS No. 3099257, and the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” program.

[9]https://github.com/ML-KULeuven/soccer_xg

References

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Calzavara, S., Lucchese, C., and Tolomei, G. Adversarial training of gradient-boosted decision trees. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2429–2432, 2019.

Calzavara, S., Ferrara, P., and Lucchese, C. Certifying decision trees against evasion attacks by program analysis. In *Computer Security – ESORICS 2020*, pp. 421–438, Cham, 2020. Springer International Publishing.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. IEEE, 2017.

Chen, H., Zhang, H., Boning, D., and Hsieh, C.-J. Robust decision trees against adversarial examples. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1122–1131, Long Beach, California, USA, 2019a.

Chen, H., Zhang, H., Si, S., Li, Y., Boning, D., and Hsieh, C.-J. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems 32*, pp. 12317–12328. Curran Associates, Inc., 2019b.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM, 2016.

Decroos, T., Bransen, L., Van Haaren, J., and Davis, J. Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1851–1861, 2019.

Devos, L., Meert, W., and Davis, J. Fast gradient boosting decision trees with bit-level data structures. In *Proceedings of ECML PKDD*. Springer, 2019.

Devos, L., Meert, W., and Davis, J. Verifying tree ensembles by reasoning about potential instances. In *SIAM International Conference on Data Mining proceedings*. Alexandria, Virginia, U.S, SIAM, 2021.

Drews, S., Albarghouthi, A., and D'Antoni, L. Proving data-poisoning robustness in decision trees. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 1083–1097, 2020.

Dwork, C., Hardt, M., Pitassi, T., Reingold, O., and Zemel, R. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226. ACM, 2012.

Einziger, G., Goldstein, M., Sa'ar, Y., and Segall, I. Verifying robustness of gradient boosted models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2446–2453, 2019.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples, 2014.

Gurobi Optimization, L. Gurobi optimizer reference manual, 2021. URL http://www.gurobi.com.

Kantchelian, A., Tygar, J. D., and Joseph, A. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pp. 2387–2396, 2016.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pp. 3146–3154, 2017.

Likhachev, M., Gordon, G. J., and Thrun, S. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pp. 767–774, 2004.

Lucey, P., Bialkowski, A., Monfort, M., Carr, P., and Matthews, I. "Quality vs quantity": Improved shot prediction in soccer using strategic features from spatiotemporal data. In *Proc. of MIT Sloan Sports Analytics Conference*, 2014.

Ranzato, F. and Zanella, M. Abstract interpretation of decision tree ensemble classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5478–5486, 2020.

Robberechts, P. and Davis, J. How data availability affects the ability to learn good xg models. In *Machine Learning and Data Mining for Sports Analytics*, pp. 17–27, 2020. ISBN 978-3-030-64912-8.

Rudd, S. A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains. In *New England Symposium on Statistics in Sports*, 2011.

Sato, N., Kuruma, H., Nakagawa, Y., and Ogawa, H. Formal verification of decision-tree ensemble model and detection of its violating-input-value ranges. *arXiv preprint arXiv:1904.11753*, 2019.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2013.

Törnblom, J. and Nadjm-Tehrani, S. An abstraction-refinement approach to formal verification of tree ensembles. In *International Conference on Computer Safety, Reliability, and Security*, pp. 301–313. Springer, 2019.

Törnblom, J. and Nadjm-Tehrani, S. Formal verification of input-output mappings of tree ensembles. *Science of Computer Programming*, pp. 102450, 2020.

Törnblom, J. and Nadjm-Tehrani, S. Scaling up Memory-Efficient Formal Verification Tools for Tree Ensembles. *arXiv e-prints*, art. arXiv:2105.02595, May 2021.

Van Roy, M., Robberechts, P., Yang, W.-C., De Raedt, L., and Davis, J. Leaving goals on the pitch: Evaluating decision making in soccer. In *Proceedings of the MIT Sloan Conference on Sports Analytics*, 2021.

Wang, Y., Zhang, H., Chen, H., Boning, D., and Hsieh, C.-J. On lp-norm robustness of ensemble decision stumps and trees. In *Proceedings of Machine Learning and Systems 2020*, pp. 11281–11291. 2020.

Zhang, C., Zhang, H., and Hsieh, C.-J. An efficient adversarial attack for tree ensembles. In *Advances in Neural Information Processing Systems*, volume 33, pp. 16165–16176, 2020.