Verifiable Boosted Tree Ensembles

Stefano Calzavara*, Lorenzo Cazzaro*, Claudio Lucchese* and Giulio Ermanno Pibiri*
*Università Ca' Foscari Venezia, Italy
Email: {name.surname}@unive.it

***Abstract*—Verifiable learning advocates for training machine learning models amenable to efficient security verification. Prior research demonstrated that a specific class of decision tree ensembles – called *large-spread ensembles* – allow for robustness verification in polynomial time against any norm-based attacker. This study expands prior work on verifiable learning from basic ensemble methods based on hard majority voting to state-of-the-art *boosted tree ensembles*, such as those trained using XGBoost or LightGBM. Our formal results indicate that robustness verification is achievable in polynomial time for large-spread boosted ensembles when considering attackers based on the L_∞-norm, but remains NP-hard for other norm-based attackers. Nevertheless, we present a pseudo-polynomial time algorithm to verify robustness against attackers based on the L_p-norm for any $p \in \mathbb{N} \cup \{0\}$, which in practice grants excellent performance and enables verification methods outperforming the state of the art in terms of analysis times. Our experimental evaluation on public datasets shows that large-spread boosted ensembles are accurate enough for practical adoption, while being amenable to efficient security verification. Moreover, our techniques scale to challenging security datasets and their associated security properties proposed in prior work.**

1. Introduction

Security of Machine Learning (ML) is a hot topic nowadays, because models trained using classic supervised learning algorithms are vulnerable to *evasion attacks*, i.e., malicious perturbations of inputs designed to force mispredictions at test time [1], [2], [3]. When ML models are deployed in adversarial settings, standard performance measures such as accuracy, precision and recall do not provide appropriate guarantees, because they do not take adversarial perturbations into account. This motivated a long research line on adversarial ML and the definition of new measures such as *robustness*, which explicitly quantifies resistance to evasion attacks [4].

Unfortunately, verifying the security of ML models against evasion attacks is computationally hard, because verification must consider all the possible adversarial perturbations that the attacker may perform. In this work, we focus on the security of *tree ensembles* [5], a popular class of ML models particularly effective for non-perceptual classification tasks. Kantchelian et al. [6] were the first to prove that the robustness verification problem for tree ensembles is NP-complete when malicious perturbations are modeled by norm-based constraints. Follow-up work [7] extended this negative result to stump ensembles, i.e., ensembles including just trees of depth one, and proposed *approximate* verification approaches, which can formally prove the absence of evasion attacks, but may incorrectly report evasion attacks also for secure inputs. Exact verification approaches against specific attackers, e.g., modeled in terms of the L_∞-norm, have also been proposed [8], [9]. Yet, such approaches have to deal with the NP-hardness of robustness verification and, while effective in many practical cases, they fail when the size of the tree ensemble is large.

To improve over this bleak picture, recent work proposed *verifiable learning* for tree ensembles [10]. The key idea of verifiable learning is the development of new training algorithms that learn restricted classes of models amenable to efficient security verification, e.g., in polynomial time. Although promising, this paradigm is still limited in scope because it assumes the adoption of simple ensemble methods based on hard majority voting, i.e., each tree in the ensemble makes a class prediction and the most frequent class is returned by the ensemble. This approach suffers from limited predictive power, because each tree has the same weight upon prediction, while different trees are normally trained to overfit a subset of the training data. State-of-the-art ensemble methods such as *gradient boosting* [11] operate rather differently, because each tree in the ensemble is a *regressor* predicting a real-valued score, which intuitively models the confidence of their prediction. The ensemble prediction is then performed by summing together the individual scores and translating the result into the final class prediction. This paradigm shift makes it difficult to generalize existing work on verifiable learning to boosted tree ensembles. In simple ensemble methods based on hard majority voting, all the trees provide the same contribution to the prediction, hence the best strategy for the attacker is always targeting the trees where it is easier to mispredict. In boosted ensembles, instead, the attacker faces a dilemma: is it better to target trees with higher scores, but that are difficult to subvert, or trees with lower scores, that are instead easy to manipulate?

Contributions.

We here summarize our contributions:

1) We extend existing research on verifiable learning [10] from simple ensemble methods based

on hard majority voting to state-of-the-art boosted tree ensembles, e.g., those trained using LightGBM [12]. Our analysis shows that a restricted class of tree-based models, called *large-spread boosted ensembles*, admit exact security verification in polynomial time when considering attacks based on the L_∞-norm. We then prove that security verification remains NP-hard even for our restricted class of models when considering other norm-based attackers. Still, we present a pseudo-polynomial time algorithm to verify robustness against attackers based on the L_p-norm for any $p \in \mathbb{N} \cup \{0\}$, which in practice grants excellent performance (Sections 3 and 4).

2) We implement our efficient verification algorithms for large-spread boosted ensembles and we propose a new training algorithm for such models, deployed as a simple extension of the popular LightGBM library. To support reproducible research, we make our software available on GitHub[1] (Section 5).
3) We perform an extensive experimental evaluation on public datasets to show the accuracy and robustness of large-spread boosted ensembles with respect to traditional boosted ensembles. The net result is that our models are accurate and robust enough for practical adoption, while being amenable to efficient security verification. Moreover, we show that our algorithms can be up to $100\times$ faster than existing verification algorithms for boosted ensembles, which struggle to assess verification even when provided with significant computational resources (Section 6).
4) We also compare our large-spread boosted ensembles with the tree ensembles proposed in [10] in terms of accuracy and robustness. The results clearly show the significant gain in predictive power and robustness obtained by the large-spread boosted ensembles over the previous proposal (Section 7).
5) Finally, we show that our techniques scale to challenging security datasets and their associated security properties proposed in prior work [13]. In particular, we discuss how to encode recent security properties like Stability, Maximum Score Decrease and Small Neighborhood in our framework and we show that our verification method is able to efficiently prove them on the Twitter Spam Accounts and Twitter Spam URLs datasets (Section 8).

2. Background

We here introduce preliminary notions and some technical ingredients used throughout the paper. For readability, Table 1 summarizes the main notation.

1. https://github.com/LorenzoCazzaro/verifiable-boosted-tree-ensembles

$\vec{x}$	Instance drawn from the feature space $\mathcal{X}$
x_i	i-th component of the vector $\vec{x}$
y	Class label drawn from the set of labels $\mathcal{Y}$
d	Number of features of $\vec{x}$ (i.e., dimensionality of $\mathcal{X}$)
t	Regression tree
T	Boosted tree ensemble
N	Number of nodes of a boosted tree ensemble
m	Number of trees of a boosted tree ensemble
$\vec{\delta}$	Adversarial perturbation
$A_{p,k}$	Attacker based on L_p-norm (max perturbation k)

TABLE 1: Summary of notation.

2.1. Supervised Learning

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d-dimensional vector space of real-valued *features*. An *instance* $\vec{x} \in \mathcal{X}$ is a d-dimensional feature vector $\langle x_1, x_2, \ldots, x_d \rangle$ representing an object in the vector space $\mathcal{X}$. Each instance is assigned a class label $y \in \mathcal{Y}$ by an unknown *target* function $f : \mathcal{X} \to \mathcal{Y}$. In this work, we focus on binary classification, i.e., we let $\mathcal{Y} = \{+1, -1\}$ include just a positive and a negative class. Multi-class classification problems can be encoded in terms of binary classification by using standard techniques like one-versus-rest [14].

Supervised learning algorithms automatically learn a *classifier* $g : \mathcal{X} \to \mathcal{Y}$ from a *training set* of correctly labeled instances $\mathcal{D}_{train} = \{(\vec{x}_i, f(\vec{x}_i))\}_i$, with the goal of approximating the target function f as accurately as possible. The performance of classifiers is normally estimated on a *test set* of correctly labeled instances $\mathcal{D}_{test} = \{(\vec{z}_i, f(\vec{z}_i))\}_i$, disjoint from the training set, yet drawn from the same data distribution. For example, the standard *accuracy* measure $a(g, \mathcal{D}_{test})$ counts the percentage of test instances where the classifier g returns a correct prediction.

2.2. Boosted Tree Ensembles

A *regression tree* $t : \mathcal{X} \to \mathbb{R}$ can be recursively defined as follows: t is either a leaf $\lambda(s)$ for some real-valued score $s \in \mathbb{R}$ or an internal node $\sigma(f, v, t_l, t_r)$, where $f \in \{1, \ldots, d\}$ identifies a feature, $v \in \mathbb{R}$ is a threshold for the feature, and t_l, t_r are regression trees (left and right child) themselves. At test time, the instance $\vec{x}$ traverses the regression tree t as follows: starting from the root of t, for each traversed tree node $\sigma(f, v, t_l, t_r)$, $\vec{x}$ falls into the left sub-tree t_l if $x_f \leq v$ and into the right sub-tree t_r otherwise, until it eventually reaches a leaf $\lambda(s)$. We write $t(\vec{x}) = s$ when $\vec{x}$ reaches a leaf $\lambda(s)$ of the tree t upon prediction and we refer to the score s as the *raw prediction* of t on $\vec{x}$. Raw predictions might have multiple interpretations, representing, e.g., the probability of belonging to the positive class or the predicted value in case of a regression task. For example, Figure 1 represents a regression tree t of depth 2 where scores range in the interval [0,1] to represent the probability of belonging to the positive class. In this case $t(\langle 8, 6 \rangle) = 0.8$ and $t(\langle 12, 7 \rangle) = 0.3$.

A *boosted tree ensemble* $T : \mathcal{X} \to \mathcal{Y}$ is a classifier built on top of a set of regression trees $\{t_1, \ldots, t_m\}$, which

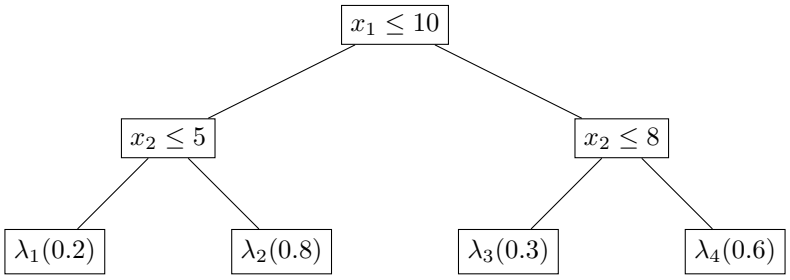


Figure 1: Example of regression tree.

aggregates individual raw predictions to produce a single class prediction $T(\vec{x})$. We use the term *boosted* to stress that these ensembles are assumed to have been trained using state-of-the-art boosting algorithms like AdaBoost [15], Gradient Boosting [11], and its popular variants such as LightGBM [12] and XGBoost [16]; for readability, we often use the terms "tree ensembles" or even just "ensembles" in the following.

Given an instance $\vec{x}$, the ensemble T computes the class prediction $T(\vec{x})$ as follows. First, the ensemble computes the raw prediction $\widehat{T}(\vec{x}) = \sum_{i=1}^{m} t_i(\vec{x})$. The raw prediction $\widehat{T}(\vec{x})$ is then transformed by an *inverse link* function $\iota : \mathbb{R} \to \mathbb{R}$, and compared against a threshold $\tau \in \mathbb{R}$: if $\iota(\widehat{T}(\vec{x})) \geq \tau$ then $T(\vec{x}) = +1$, otherwise $T(\vec{x}) = -1$. We do not assume any specific choice of ι, but we require ι to be monotonically increasing, i.e., higher scores of the raw prediction push the prediction towards the positive class.

2.3. Classifier Robustness

Robustness is a popular measure used to estimate the performance of classifiers deployed in an adversarial setting. It requires the classifier to perform a correct prediction on a test instance $\vec{x}$ and stick to the same prediction for any possible *evasion attack* attempt crafted from $\vec{x}$, e.g., by adding some maliciously crafted perturbation $\vec{\delta} \in \mathbb{R}^d$ to it. We model the *attacker* $A : \mathcal{X} \to 2^{\mathcal{X}}$ as a function from instances to sets of instances, defining the possible evasion attacks against them. We assume that $\vec{x} \in A(\vec{x})$ for all instances $\vec{x} \in \mathcal{X}$, i.e., the attacker can always leave the original instances unchanged.

Definition 1 (Robustness)**.** *The classifier g is* robust *against the attacker A on the instance $\vec{x}$ with true label y if and only if $\forall \vec{z} \in A(\vec{x}) : g(\vec{z}) = y$.*

Based on the definition of robustness, for a given attacker A, we can define the robustness measure $r_A(g, \mathcal{D}_{test})$ by computing the percentage of test instances where the classifier g is robust. In the following, we focus on attackers represented in terms of an arbitrary norm, i.e., the attacker's capabilities are defined by some norm function and a maximum perturbation k. Concretely, we consider the attacker defined as $A_{p,k}(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid ||\vec{z} - \vec{x}||_p \leq k\}$ for some budget k and some $p \in \mathbb{N} \cup \{0, \infty\}$.

2.4. Robustness Verification of Tree Ensembles

We here review large-spread ensembles from recent prior work [10] since the main objective of this paper is to generalize them to work over boosted tree ensembles.

The robustness verification problem for tree ensembles is NP-hard for any norm-based attacker [6]. Although different heuristics have been proposed to tame this computational complexity [6], [7], [8], NP-hardness still constitutes a roadblock to verification when the model size grows.

To cope with this complexity, recent work identified a restricted class of tree ensembles – known as *large-spread ensembles* – admitting robustness verification in polynomial time [10]. This positive result assumes ensembles based on hard majority voting, where each tree makes its own class prediction and the ensemble returns the most frequently predicted class. As we explained in Section 2.2, the boosted tree ensembles considered in this paper operate rather differently, because they add together raw predictions (scores) and use an inverse link function ι to determine the class prediction through thresholding. Hence, it is not easy to generalize them to work for boosted tree ensembles.

The key characteristic of large-spread ensembles is that the thresholds chosen for different trees are sufficiently far away that each feature can be successfully attacked in at most one tree, hence the attacks against a large-spread ensemble can be decomposed into a sum of orthogonal attacks against the individual trees [10]. This permits to compose the security analysis of the individual trees to draw conclusions about the robustness of the entire ensemble, thus enabling efficient verification. The formal definition of large-spread ensemble is given below.

Definition 2 (Large-Spread Ensemble [10])**.** *Given the ensemble $T = \{t_1, \ldots, t_m\}$, its p-spread $\psi_p(T)$ is the minimum value $||v - v'||_p$ computed for any v, v' such that there exists two different trees $t, t' \in T$ such that $\sigma(f, v, t_l, t_r) \in t$ and $\sigma(f, v', t'_l, t'_r) \in t'$ for some f, t_l, t_r, t'_l, t'_r. We say that T is* large-spread *for the attacker $A_{p,k}$ iff $\psi_p(T) > 2k$.*

The existing verification algorithm for large-spread ensembles is based on a tree annotation procedure, which we also leverage in this paper. The annotation procedure associates each node of the individual trees with a symbolic representation of the set of instances that may traverse it in presence of adversarial manipulations. The procedure first annotates the root of the tree with the d-dimensional

hyper-rectangle $(-\infty, +\infty]^d$, meaning that every instance will traverse the root. Children are then annotated by means of a recursive tree traversal: concretely, if the parent node $\sigma(f, v, t_1, t_2)$ is annotated with $(l_i, r_i]^d$, then the annotations of the roots of t_1 and t_2 are defined as $(l_i^1, r_i^1]^d$ and $(l_i^2, r_i^2]^d$ respectively:

$$(l_i^1, r_i^1] = \begin{cases} (l_i, r_i] \cap (-\infty, v] = (l_i, \min\{r_i, v\}] & \text{if } i = f \\ (l_i, r_i] & \text{otherwise,} \end{cases}$$

and:

$$(l_i^2, r_i^2] = \begin{cases} (l_i, r_i] \cap (v, +\infty) = (\max\{l_i, v\}, r_i] & \text{if } i = f \\ (l_i, r_i] & \text{otherwise.} \end{cases}$$

Given an annotated tree and an instance $\vec{x}$, it is possible to identify the minimal adversarial perturbation required to push $\vec{x}$ into any given leaf $\lambda(s)$. In particular, let $H = (l_1, r_1] \times \ldots \times (l_d, r_d]$ be the hyper-rectangle annotating $\lambda(s)$, then we define $\mathit{dist}(\vec{x}, \lambda(s)) = \vec{\delta} \in \mathbb{R}^d$, where:[2]

$$\forall i \in [1, d] : \delta_i = \begin{cases} 0 & \text{if } x_i \in H_i = (l_i, r_i] \\ l_i - x_i + \varepsilon & \text{if } x_i \leq l_i \\ r_i - x_i & \text{if } x_i > r_i. \end{cases}$$

The value $||\vec{\delta}||_p$ is the norm of the minimal perturbation required to push $\vec{x}$ into the leaf $\lambda(s)$. Let then $L(t, \vec{x})$ be the set of the leaves of t that are reachable by $\vec{x}$ as the result of adversarial manipulations, that is:

$$L(t, \vec{x}) = \{\lambda(s) \in t \mid ||\mathit{dist}(\vec{x}, \lambda(s))||_p \leq k\}.$$

Note that we are only interested in the norms of the adversarial manipulations, and not in the hyper-rectangles needed to compute them. By exploiting this fact, it is possible to compute the set $L(t, \vec{x})$ and $\{||\mathit{dist}(\vec{x}, \lambda(s))||_p \mid \lambda(s) \in L(t, \vec{x})\}$ in $O(N)$, i.e., in linear time with respect to the number of nodes of the tree ensemble [10]. We assume this complexity for subsequent analyses.

3. Robustness Verification of Large-Spread Boosted Ensembles

Large-spread ensembles drive away from the negative NP-hardness result of robustness verification [10]. However, this prior work just considers a simple ensemble method based on hard majority voting and does not apply to state-of-the-art boosting schemes based on regression trees, such as GBDTs. Here, we generalize prior work by investigating the robustness verification problem for large-spread boosted tree ensembles. To this aim, we first identify the new challenges to deal with these models and then propose suitable verification algorithms.

2. We write $l_i - x_i + \varepsilon$ to stand for the minimum floating point number which is greater than $l_i - x_i$. We also assume here that H is not empty, i.e., there does not exist any $(l_j, r_j]$ in H such that $l_j \geq r_j$.

3.1. Optimization Problem

The first key insight of this work is that, given a large-spread boosted ensemble T and an instance $\vec{x} \in \mathcal{X}$ with true label y, we can identify the optimal evasion attack strategy for the attacker $A_{p,k}$ by solving an optimization problem. In particular, the optimization problem allows one to identify the least adversarial manipulation $\vec{\delta}$ such that $T(\vec{x}+\vec{\delta}) \neq y$, if any exists. In our formalization, we leverage the insight that any attack against a large-spread ensemble can be decomposed into a sum of orthogonal adversarial perturbations operating against the individual trees $t_i \in T$, hence the optimal evasion attack can be identified by finding the sub-ensemble $T' \subseteq T$ including the best trees to target and adding up their corresponding perturbations.

Our focus on boosting significantly complicates the formulation with respect to prior work on large-spread ensembles. In particular, the existing robustness verification algorithm from [10] assumes ensembles based on hard majority voting of the class predictions, hence all the trees equally contribute to the ensemble prediction and the best sub-ensemble T' to target just includes those trees requiring the least amount of adversarial perturbation to predict the wrong class. In the case of boosting, each tree outputs a real-valued raw prediction and different attacks lead to different changes to such predictions, making the optimal evasion attack strategy harder to identify. More specifically, the challenge is that attacks cannot be totally ordered in general, because some attacks require a small perturbation (good for the attacker) but have just a limited impact on the raw prediction (bad for the attacker), while other attacks require a large perturbation (bad for the attacker) but have a large impact on the raw prediction (good for the attacker). For example, consider the regression tree in Figure 1 and the instance $\langle 9.1, 0.1\rangle$ with label -1. This instance would normally fall in the leaf $\lambda_1(0.2)$ upon prediction, however assume it might be corrupted by the attacker so as to reach any of the other three leaves.[3] To significantly affect the raw prediction, the best choice of the attacker would be pushing the instance into the leaf $\lambda_2(0.8)$, which returns a rather different score; however this requires adding 5 to the second feature. Pushing the instance into the leaf $\lambda_3(0.3)$ has a much lower impact on the original raw prediction, however it just requires adding 1 to the first feature. Although this second attack is less impactful than the first one, it is cheaper and might still be enough to force a prediction error when the tree is used within a boosted ensemble.

The best way to combine different attacks is thus formalized as an optimization problem. For each tree $t_i \in T$ and leaf $\lambda(s_{ij}) \in L(t_i, \vec{x})$, we define the *adversarial gain* of $\lambda(s_{ij})$ as the advantage that the attacker gets when the instance $\vec{x}$ is forced into $\lambda(s_{ij})$ rather than in the original leaf that is reached in t_i.

3. To make the example more readable, we assume that adversarial perturbations are discrete with a tick $\varepsilon = 0.1$. Our formalization works for real-valued perturbations as required by the considered norms.

Definition 3 (Adversarial Gain)**.** *Consider an instance $\vec{x}$ with true label y and assume that $\vec{x}$ reaches the leaf $\lambda(s_o)$ when traversing the tree t_i upon prediction. For any leaf $\lambda(s_{ij}) \in L(t_i, \vec{x})$, we define the* adversarial gain *as follows:*

$$G(t_i, \vec{x}, y, \lambda(s_{ij})) = \begin{cases} s_o - s_{ij} & \text{if } y = +1 \\ s_{ij} - s_o & \text{if } y = -1 \end{cases}.$$

Intuitively, a leaf has a positive adversarial gain whenever it moves the instance closer to the wrong class than the original prediction. For example, consider again the decision tree in Figure 1 and the instance $\langle 9.1, 0.1\rangle$ with label -1. This instance originally reaches the leaf $\lambda_1(0.2)$ upon prediction; if the instance instead reaches the leaf $\lambda_4(0.6)$ as the result of adversarial manipulations, the adversarial gain is $0.6-0.2 = 0.4$, because an attack moving from $\lambda_1(0.2)$ to $\lambda_4(0.6)$ grants an advantage of 0.4 to the positive class over the negative class in terms of raw predictions. Notice that the definition of adversarial gain assumes that the inverse link function ι is monotonically increasing, because the true label determines whether the original raw prediction should be increased or decreased to lead to a prediction error.

We are finally ready to formulate our optimization problem. For each tree $t_i \in T$ and leaf $\lambda(s_{ij}) \in L(t_i, \vec{x})$, we introduce a new variable $z_{ij} \in \{0, 1\}$. Then, determining the optimal evasion attack strategy for the instance $\vec{x}$ with true label y against the ensemble T can be done as follows.

Problem 1 (Optimal Attack Strategy for Large-Spread Boosted Ensembles)**.** *Determine the value assignment of the variables $\{z_{ij}\}_{ij}$ from the set $\{0, 1\}$ which solves the following optimization problem:*

$$\text{maximize} \quad \sum_{i,j} z_{ij} \cdot G(t_i, \vec{x}, y, \lambda(s_{ij})) \tag{1}$$

$$\text{subject to} \quad ||\sum_{i,j} z_{ij} \cdot \mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p \leq k, \tag{2}$$

$$\forall i : \sum_{j} z_{ij} \leq 1. \tag{3}$$

In the following, we let Γ stand for the optimal value of the objective function taken by the identified solution.

In words, the attacker wants to maximize the total adversarial gain (1), which is the best strategy to force the wrong prediction, under two constraints: (2) the L_p-norm of the adversarial perturbation required to perform the attack is bounded above by the maximum perturbation k, and (3) just a single attack per tree is chosen, because a single leaf of each tree is reached upon prediction. The formal correctness of this intuition is proved in the next section.

Very importantly, note that constraint (2) leverages the observation that the optimal evasion attack against a large-spread ensemble can be decomposed into a sum of adversarial perturbations operating against the individual trees [10]. Moreover, observe that the optimal attack strategy may not be successful in flipping the ensemble prediction. Once a solution to the optimization problem is found, it is possible to verify robustness in two ways, illustrated below.

3.2. Basic Verification Algorithm

The first version of the verification algorithm explicitly constructs the smallest perturbation $\vec{\delta}_{opt}$ that might lead to an attack and then checks whether it is successful or not. In particular, given a large-spread boosted ensemble T and an instance $\vec{x}$ with true label y, the basic algorithm ("BV", henceforth) verifies robustness against $A_{p,k}$ as follows:

1) Let $T(\vec{x}) = y'$. If $y' \neq y$, return False. Otherwise, solve Problem 1 for the given input and initialize $\vec{\delta}_{opt}$ to the vector $\langle 0, \ldots, 0\rangle$.
2) For each $z_{ij} = 1$, add the vector $\mathit{dist}(\vec{x}, \lambda(s_{ij}))$ to the adversarial perturbation $\vec{\delta}_{opt}$.
3) Check whether $T(\vec{x} + \vec{\delta}_{opt}) = y$ holds true and return the outcome of this check.

The following theorem formalizes the correctness of the basic verification algorithm. The proof is provided in Appendix A.

Theorem 1. *The basic verification algorithm $BV(T, \vec{x}, y, p, k)$ returns True if and only if T is robust on the instance $\vec{x}$ with true label y against the attacker $A_{p,k}$.*

Complexity. If R is the time for solving Problem 1, the complexity of this algorithm is $R + O(dN)$, because $O(dN)$ is the time to perform the additions in step (2) and step (3) is done in $O(N)$.

3.3. Efficient Verification Algorithm

A more efficient robustness verification algorithm does not construct the optimal evasion attack $\vec{\delta}_{opt}$, but directly leverages the semantics of the adversarial gain. In particular, given a large-spread boosted ensemble T and an instance $\vec{x}$ with true label y, the efficient algorithm ("EV", henceforth) verifies robustness against $A_{p,k}$ as follows:

1) Let $T(\vec{x}) = y'$. If $y' \neq y$, return False. Otherwise, solve Problem 1 for the given input to determine the maximum value Γ taken by function (1).
2) Let $\widehat{T}(\vec{x}) = s$. If $y = +1$, return True if $\iota(s - \Gamma) \geq \tau$ and False otherwise. If $y = -1$, return True if $\iota(s + \Gamma) < \tau$ and False otherwise.

The following theorem formalizes the correctness of the efficient verification algorithm. The proof is provided in Appendix B.

Theorem 2. *The efficient verification algorithm $EV(T, \vec{x}, y, p, k)$ returns True if and only if T is robust on the instance $\vec{x}$ with true label y against the attacker $A_{p,k}$.*

Complexity. If R is the time for solving Problem 1, the complexity of this algorithm is $R+O(N)$ because computing $\widehat{T}(\vec{x})$ takes $O(N)$ time. Note that this complexity is lower than the complexity of the previous basic verification algorithm.

4. Solving the Optimization Problem

The robustness verification algorithms presented in Section 3 build on a solution to Problem 1, so we now discuss how this problem can be solved. In the following, we focus on the efficient verification algorithm from Section 3.3 for simplicity. Thus, we discuss how the maximum value Γ of the objective function (1) can be computed for different values of p, and omit how the actual value assignment of the variables z_{ij} is determined.

4.1. Solution for L_∞-Attackers

We start from the case L_∞, which is the easiest and most efficient to solve. Recall that $||\vec{x}||_\infty = \max\{|x_i| \mid 1 \leq i \leq d\}$. We observe that for any set of pairwise orthogonal vectors $\{\vec{\delta}_i\}_i$ we have $||\sum_i \vec{\delta}_i||_\infty = \max\{||\vec{\delta}_i||_\infty\}$. This means that constraint (2) is satisfied as long as we just consider leaves $\lambda(s_{ij})$ such that $||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_\infty \leq k$, i.e., the leaves in the set $L(t_i, \vec{x})$. Since these are the only leaves considered in our optimization problem, constraint (2) is trivially satisfied by definition. Constraint (3) is also straightforward to enforce, because it just states that we can select at most one leaf per tree, hence the best attack strategy amounts to picking the leaf with the highest adversarial gain in each tree. In other words, to maximize function (1) we can then use the following algorithm:

1) Initialize $\Gamma = 0$. For each tree $t_i \in T$, compute $L(t_i, \vec{x})$.
2) For each $t_i \in T$, find the leaf $\lambda(s_{ij}) \in L(t_i, \vec{x})$ with the largest adversarial gain $G(t_i, \vec{x}, y, \lambda(s_{ij}))$; in case of ties, arbitrarily break them to select one leaf. Then, sum $G(t_i, \vec{x}, y, \lambda(s_{ij}))$ to Γ.

Complexity. We recall that computing the sets $L(t_i, \vec{x})$ for all trees $t_i \in T$ takes $O(N)$ time. Since the leaf with the largest adversarial gain in each t_i can be identified while computing $L(t_i, \vec{x})$, the total complexity of the algorithm is $R = O(N)$. It follows that also the EV algorithm runs in $O(N)$ time, i.e., robustness verification can be performed in linear time with respect to the number of nodes of the ensemble.

4.2. Solution for L_0-Attackers

The case L_0 is more complicated. Again, recall that $||\vec{x}||_0 = |\{i \,|\, x_i \neq 0\}|$. We start from the observation that, for any set of pairwise orthogonal vectors $\{\vec{\delta}_i\}_i$, we have $||\sum_i \vec{\delta}_i||_0 = \sum_i ||\vec{\delta}_i||_0$. Hence, constraint (2) can be rewritten as:

$$\sum_{i,j} z_{ij} \cdot ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0 \leq k.$$

With this insight, we observe that our optimization problem is reminiscent of a variant of the classic 0-1 *knapsack problem* [17], where the adversarial gain represents the value of the items and the L_0-norm of the adversarial perturbations represents their weight. We first review the knapsack problem and how it can be efficiently solved with dynamic programming.

Given a set of items S, each with a value v and a weight w, and a maximum capacity W, the goal of 0-1 knapsack is to choose the best items to pick to maximize the overall value without exceeding the capacity W. Formally, for each item $h \in S$ we introduce a new variable $z_h \in \{0, 1\}$, and we formulate 0-1 knapsack as the optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_h z_h \cdot v_h \\ \text{subject to} \quad & \sum_h z_h \cdot w_h \leq W. \end{aligned}$$

Assuming that all weights are positive integers, we can build a matrix M of size $(|S|+1) \times (W+1)$, where the entry $M[h, w]$ stores the maximum value that can be obtained with capacity less than or equal to w using up to h items. The matrix M can be defined as follows:

- $M[0, w] = 0$ for all $0 \leq w \leq W$, i.e., the maximum value is 0 when the current maximum weight w is 0 and no item can be taken yet.
- $M[h, w] = M[h-1, w]$ when $w \geq 1$ and $w_h > w$, i.e., if the weight of item h exceeds the current maximum weight w, then h cannot be taken.
- $M[h, w] = \max\{M[h-1, w], M[h-1, w-w_h]+v_h\}$ when $w \geq 1$ and $w_h \leq w$, i.e., if the weight of item h does not exceed the current maximum weight w, then h can either be taken or not, depending on its value.

The solution to the problem is then found in the lower right corner of the matrix M. This simple dynamic programming algorithm has complexity $O(|S| \cdot W)$.

We now discuss how the same idea can be generalized to efficiently solve Problem 1. For each tree $t_i \in T$ and leaf $\lambda(s_{ij}) \in L(t_i, \vec{x})$ such that $G(t_i, \vec{x}, \lambda(s_{ij})) > 0$, we compute $||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0$. Note that the value of such computation is always a positive integer by definition of $||\cdot||_0$. The key difference of Problem 1 with respect to 0-1 knapsack is that we have to enforce constraint (3), which ensures that just a single attack per tree is chosen. To deal with this, we create a matrix M of size $(m+1) \times (k+1)$ defined as follows:

- $M[0, q] = 0$ for all $0 \leq q \leq k$.
- $M[i, q] = M[i-1, q]$ when $i \geq 1$ and $\forall\, \lambda(s_{ij}) \in L(t_i, \vec{x}) : ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0 > q$.
- $M[i, q] = \max\{M[i-1, q], \max Q\}$ when $i \geq 1$ and $\exists\, \lambda(s_{ij}) \in L(t_i, \vec{x}) : ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0 \leq q$, where

$$\begin{aligned} Q = \{&M[i-1, q - ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0] + G(t_i, \vec{x}, y, \lambda(s_{ij})) \mid \\ &\lambda(s_{ij}) \in L(t_i, \vec{x}) \wedge ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0 \leq q\}. \end{aligned}$$

The idea of the solving algorithm is equivalent to the dynamic programming approach for 0-1 knapsack, where the second clause encodes the case where the tree t_i cannot be attacked within the current maximum perturbation q and the third clause instead encodes the case where t_i can be attacked, with the algorithm determining whether this

should be done or not. Note that the third clause admits the possibility of having a set of possible attacks against t_i, in which case we select the one maximizing the adversarial gain against the ensemble $t_1, \ldots, t_i$. Soundness comes from the observation that, in a large-spread ensemble, each feature can be successfully attacked in at most one tree. Hence, an optimal attack against the ensemble $t_1, \ldots, t_{i-1}$ cannot be invalidated when considering the next tree t_i, because the features manipulated by an attack against t_i cannot affect the prediction paths of $t_1, \ldots, t_{i-1}$, i.e., the verification problem has an optimal sub-structure. The solution to the problem Γ is again found in the lower right corner of the matrix M, i.e., $M[m][k]$.

Complexity. We compute the sets $L(t_i, \vec{x})$ for all trees $t_i \in T$ in $O(N)$ time. The computation of $\max Q$ takes $O(2^D)$ time, where D is the maximal tree depth in the ensemble. The total complexity of the algorithm is therefore $R = O(m \cdot k \cdot 2^D)$ since $N = O(m \cdot 2^D)$. It follows that also the EV algorithm runs in this time. Technically speaking, the complexity of the algorithm is *pseudo-polynomial*, because there is no formal guarantee that k is bounded by a polynomial function of $m \cdot 2^D$. Nevertheless, k is very small compared to $m \cdot 2^D$ in practical cases, because the number of features that a meaningful L_0-norm attacker can perturb is much smaller than the size of the ensemble. Indeed, observe that an L_0-norm attacker who can arbitrarily corrupt k features can trivially break the robustness of any classifier when k grows large.

4.3. Solution for L_p-Attackers

Finally, we deal with the case L_p with $p \in \mathbb{N}$. Again, we start from a simple observation: for any set of pairwise orthogonal vectors $\{\delta_i\}_i$ and any $p \in \mathbb{N}$, we have $||\sum_i \vec{\delta}_i||_p = (\sum_i ||\vec{\delta}_i||_p^p)^{1/p}$. Hence, constraint (2) can be rewritten as:

$$\sum_{i,j} z_{ij} \cdot ||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p^p \leq k^p.$$

We can then leverage the same idea of the case L_0, but some additional care is needed. In particular, observe that $||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p^p$ is not necessarily a positive integer, so the previous formulation cannot be readily applied. A possible solution to work again with positive integers is to multiply each $||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p^p$ by a suitable power of 10. In particular, let η stand for a normalization function which multiplies its argument by 10^ℓ, where $\ell \geq 0$ is a constant large enough to ensure that all the arguments of η are transformed into integers, e.g., if $\ell = 3$ and $x = 0.123$, then $\eta(x) = 123$. We can therefore reuse the same formulation we described for the case L_0, where we replace $||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_0$ with $\eta(||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p^p)$ and k with $\eta(k^p)$.

Complexity. Similarly to the case L_0, it follows that the time for solving Problem 1 is $R = O(m \cdot \eta(k^p) \cdot 2^D)$, which is again pseudo-polynomial, but observe that the term $\eta(k^p)$ may grow very fast, because the normalization factor ℓ appears as an exponent of 10. Still, when ℓ is small, one can directly apply the proposed approach to establish exact robustness bounds. In practice, ℓ can be made small by performing a discretization of the feature space, which is a common pre-processing practice. Even in absence of discretization, one may mitigate the complexity by enforcing a properly reduced value of ℓ in the verification phase. By taking $\lfloor \eta(||\mathit{dist}(\vec{x}, \lambda(s_{ij}))||_p^p) \rfloor$ and $\lceil \eta(k^p) \rceil$ in constraint (2), we get a conservative approximation for robustness verification, because distances are under-approximated and the maximum adversarial perturbation is over-approximated with an arbitrarily chosen precision. This way, one may occasionally get false alarms where robust instances are flagged as not robust, but no evasion attack is missed.

4.4. NP-Hardness Result

We proposed verification algorithms for the case $p \in \mathbb{N} \cup \{0\}$ based on 0-1 knapsack, hence running in pseudo-polynomial time. Here we show that such limitation is fundamental, because there exists no polynomial time algorithm for the case $p \in \mathbb{N} \cup \{0\}$ despite our large-spread requirement. The proof is provided in Appendix C.

Theorem 3. *The robustness verification problem for large-spread boosted ensembles is NP-hard when considering attackers based on the L_p-norm, for any $p \in \mathbb{N} \cup \{0\}$.*

5. Implementation

We here describe the implementation of CARVE-GBM, our robustness verification tool for large-spread boosted ensembles. We then present a simple training algorithm for large-spread boosted ensembles, implemented on top of the popular LightGBM library.

CARVE-GBM. CARVE-GBM is a C++ implementation of the EV algorithm presented in Section 3.3, which uses the techniques in Section 4 to solve the underlying optimization problem. It currently supports large-spread boosted ensembles trained using LightGBM, but it can be easily extended to other input formats, e.g., to support XGBoost.

Training Algorithm. We implemented a training algorithm for large-spread boosted ensembles on top of the popular learning algorithm for boosted tree ensembles available in LightGBM, based on Gradient Boosting. In particular, we extended LightGBM to keep track of the thresholds used for each feature upon tree construction. Once the threshold v has been chosen for feature f in a tree t_i, all the other thresholds v' for the same feature which would violate the large-spread condition (Definition 2) are not considered for node splitting when constructing the trees t_j with $j > i$. Note that this may effectively prevent the reuse of f within the next trees, in particular when all the other thresholds v' are close to v. This "black-listing" method reduces, tree by tree, the available splitting options for creating new nodes. Eventually, this may force the training algorithm to stop the construction of new trees. In the experimental section we show that this worst-case scenario is not detrimental in terms of accuracy of the generated model.

Dataset	Instances	Features	Distribution
FMNIST	13,866	784	50%/50%
MNIST	14,000	784	51%/49%
Webspam	350,000	254	70%/30%

TABLE 2: Datasets statistics

6. Experimental Evaluation

In this section we experimentally evaluate the accuracy and robustness of the proposed large-spread boosted ensembles with respect to the state of the art. Moreover, we evaluate the primary objective of our large-spread models, i.e., we assess whether they are amenable to efficient robustness verification.

6.1. Methodology

Our experimental evaluation is performed on three public datasets: Fashion-MNIST (FMNIST) [18] , MNIST [19] and Webspam [20]. We reduce FMNIST and MNIST to binary classification tasks by considering two subsets of them. In particular, for FMNIST we consider the instances with class 0 (T-shirt/top) and 3 (Dress), while for MNIST we keep the instances representing the digits 2 and 6. The key characteristics of the chosen datasets are reported in Table 2. Each dataset is partitioned into a training set, validation set and a test set, using 55/15/30 stratified random sampling as customary. Additionally, we reduce the size of the Webspam test set to match the number of instances of the FMNIST test set using stratified random sampling.

We then use the following methodology:

1) We perform hyper-parameter tuning to identify the best-performing traditional boosted tree ensemble, here also referred as traditional Gradient Boosted Decision Trees (GBDT) model, trained using LightGBM. Evaluation is performed on the validation set looking for the ensemble with highest accuracy. We train up to 500 regression trees, setting an early stopping criterion of 50 boosting rounds. We tune the number of leaves from the set $\{16, 32, 64, 128, 256\}$, setting the learning rate to 0.1.
2) We perform the same hyper-parameter tuning to identify the best-performing large-spread boosted tree ensemble trained using our algorithm. We do this for different norms (L_∞, L_0, L_1) and different magnitudes of the adversarial perturbation k. We choose the values of k based on those used in previous work and to obtain roughly the same decrease in robustness among datasets for each norm. For example, the value $k = 0.1$ for L_1-norm on FMNIST is also used in [7], and we consider smaller perturbations for Webspam, as models trained on Webspam are shown to be too fragile for larger perturbations in [8].
3) We assess the accuracy and robustness of both the traditional boosted ensemble and our large-spread boosted ensemble to understand how the enforced model restriction impacts on classic performance measures in different settings (different norms and magnitudes of adversarial perturbations).
4) We also assess the efficiency and scalability of robustness verification with respect to existing state-of-the-art tools for different norms.

For all the experiments, we fix the inverse link function ι to the identity and we set the threshold τ to 0.

Baselines. Robustness verification for large-spread models can be efficiently performed using CARVE-GBM, while traditional models without the large-spread restriction must be analyzed using existing verification tools. We consider two approaches as baselines. For L_∞ we consider SILVA [9], a state-of-the-art verification tool for tree ensembles. SILVA uses abstract interpretation and in particular the hyper-rectangle abstract domain to perform exact robustness verification against attackers based on the L_∞-norm. For L_0 and L_1, we instead consider an exact approach based on mixed integer linear programming (MILP) [6].

Since SILVA and MILP may not scale on the entire test set, we enforce a timeout of 10 seconds per instance and we compute approximate values of robustness as required. In particular, when the verification tool goes in timeout, we consider its output as unknown, and we compute lower and upper bounds to robustness by considering such unknown instances as non-robust or robust respectively. We present approximate results using the $\pm$ notation.

6.2. Accuracy and Robustness

Table 3 compares the accuracy and the robustness of traditional GBDT models trained using LightGBM and the large-spread boosted ensembles trained using our algorithm.

Results for L_∞-Attackers. We observe that the large-spread condition preserves the accuracy of the trained models: in all the cases, the accuracy of large-spread boosted ensembles is equivalent to the accuracy of GBDT models. Although prior work already showed that large-spread ensembles can be accurate, some performance degradation was observed [10]. Our use of boosting further relaxes the practical limitations of the large-spread condition, likely because the trees of a boosted ensemble are specifically trained to compensate the weaknesses of their predecessors, yielding models which are equivalent to the state of the art. In turn, the large-spread condition is beneficial to robustness verification. Indeed, a state-of-the-art tool like SILVA can only provide approximate robustness bounds, because verification takes too much time. The bounds are rather large and make robustness verification unreliable: for many cases, the uncertainty is above ± 0.10. For example, in the case of the Webspam dataset with the highest perturbation, robustness ranges between 0.61 and 0.97. This gap is unacceptable for a credible security verification. CARVE-GBM is instead able to establish exact robustness bounds for our large-spread

Dataset	p	k	Accuracy		Robustness	
			GBDT	Large-Spread	GBDT	Large-Spread
FMNIST	∞	0.005	0.97	0.97	0.93 ± 0.01	0.96
		0.01	0.97	0.97	0.81 ± 0.12	0.89
		0.015	0.97	0.97	0.68 ± 0.21	0.88
	0	1	0.97	0.96	0.94	0.96
		2	0.97	0.96	0.90	0.90
		3	0.97	0.96	0.86	0.85
	1	0.05	0.97	0.97	0.87	0.89
		0.1	0.97	0.96	0.81	0.83
		0.15	0.97	0.96	0.72	0.66
MNIST	∞	0.01	0.99	0.99	0.98	0.99
		0.02	0.99	0.99	0.93 ± 0.02	0.98
		0.03	0.99	0.99	0.85 ± 0.10	0.97
	0	1	0.99	0.99	0.77 ± 0.21	0.99
		2	0.99	0.99	0.67 ± 0.21	0.98
		3	0.99	0.99	0.44 ± 0.21	0.96
	1	0.05	0.99	0.99	0.87 ± 0.05	0.99
		0.1	0.99	0.99	0.86 ± 0.04	0.96
		0.15	0.99	0.99	0.84 ± 0.04	0.84
Webspam	∞	0.0004	0.99	0.99	0.90 ± 0.08	0.97
		0.0006	0.99	0.99	0.84 ± 0.13	0.97
		0.0008	0.99	0.99	0.79 ± 0.18	0.96
	0	1	0.99	0.96	-	0.91
		2	0.99	0.96	-	0.86
		3	0.99	0.96	-	0.79
	1	0.002	0.99	0.97	-	0.95
		0.003	0.99	0.96	-	0.93
		0.004	0.99	0.96	-	0.89

TABLE 3: Accuracy and robustness (against $A_{p,k}$) measures for traditional GBDT models and large-spread boosted ensembles. The robustness against $A_{\infty,k}$ of the GBDT models is obtained using SILVA, while MILP is used against $A_{p,k}$ with $p = \{0, 1\}$. CARVE-GBM is used for computing the robustness of the large-spread models for $p \in \{\infty, 0, 1\}$. Robustness of GBDT over Webspam for the $p \in \{0, 1\}$ case is omitted, because MILP does not scale to those models.

models. Observe that, in all cases, the robustness of large-spread models is close to the most optimistic robustness estimate of the GBDT models and quite close to 1 in general.

Results for L_0-Attackers. In this case, the large-spread restriction introduces a very limited accuracy loss with respect to the GBDT models. The highest loss is on the Webspam dataset, where the large-spread models sacrifice 0.03 of accuracy. This is definitely acceptable, because their accuracy is still 0.96, which is close to 1. As to robustness, we observe that MILP struggles against the analyzed models. In particular, for the Webspam dataset it is completely unable to provide a reasonable robustness estimate, because more than 90% of the instances go in timeout after 10 seconds. Again, CARVE-GBM is able to establish exact robustness bounds very efficiently in all cases, showing that the robustness of the large-spread models is reasonably high. In particular, even against the strongest attacker, their robustness is still above the frequency of the majority class (0.70). For the other two datasets, the robustness of large-spread models is either comparable to the most optimistic robustness estimate of the GBDT models or even much better. The major improvement is for the MNIST dataset, where robustness against the strongest attacker increases from $0.44 + 0.21 = 0.65$ (at best) to 0.96 when moving from GBDT models to large-spread models. The increase in robustness is a byproduct of the large-spread condition: by enforcing thresholds in different trees to be far away, evasion attacks are empirically more complex to craft.

Results for L_1-Attackers. Also the evaluation against L_1-attackers is largely positive. Again the large-spread condition does not enforce any significant loss of accuracy compared to GBDT models, because the highest loss is 0.03 (again on the Webspam dataset). The robustness of the large-spread models is generally higher than the robustness of GBDT models or equal to the most optimistic estimate, with just a few exceptions. These cases happen when the best-performing large-spread model is noticeably smaller than the best-performing GBDT model, hence empirically less robust against evasion attacks. This is not a negative result since the primary goal of the large-spread condition is to enable efficient robustness verification rather than to enhance the robustness of the boosted tree ensembles. Thus, in some cases, the best large-spread model may have a slightly lower robustness than the corresponding traditional model. These cases can be avoided by integrating robustness verification into the hyper-parameter tuning

pipeline of large-spread models, thus looking for the model showing the highest robustness or the best trade-off between accuracy and robustness. However, this integration can not be easily implemented for GBDT models because robustness verification does not always scale. Hence, we did not tune the hyper-parameters of the large-spread models taking into account the robustness of the model but only their accuracy, to keep the experimental comparison fair.

6.3. Performance and Scalability

Next, we turn our attention to assess the benefits on robustness verification enabled by our large-spread boosted ensembles.

Performance. In the first experiment, we train a traditional GBDT model of the same size (number of trees and nodes) of our best-performing large-spread boosted ensemble and we compare the robustness verification times of SILVA (for L_∞) and MILP (for L_0, L_1) against CARVE-GBM. We limit the analysis to 500 instances of the test set sampled using stratified random sampling, because our competitors often take too much time to run.[4] The results are shown in Table 4. Note that only entries in the same row (i.e., with same k) can be directly compared, because large-spread boosted ensembles trained for different values of k may differ in terms of size, e.g., number of trees. Entries in the same row determine how CARVE-GBM fares against existing verification tools for models of the same size.

The first observation we make is that CARVE-GBM is significantly faster than SILVA in the vast majority of cases. For example, for the FMNIST dataset and the highest perturbation, the verification times of CARVE-GBM are three orders of magnitude lower than SILVA (4 seconds vs. 3,448 seconds). The same happens for the Webspam dataset, where CARVE-GBM takes 4 seconds while SILVA takes 2,142 seconds for the highest perturbation. The MNIST dataset shows a different trend, because SILVA is very efficient there. We do not have a clear explanation for this observation, except that even NP-hard problems can be easy to solve in specific cases and for some reason the MNIST model seems straightforward to verify. However, we stress that SILVA does not provide formal complexity bounds, but reuses an abstract interpretation engine hoping that it scales to the NP-hardness of robustness verification. This is the case for MNIST, but the other two datasets make SILVA struggle. Finally, we discuss the comparison with MILP: again, CARVE-GBM always performs better than its competitor. For example, for L_0-attackers on the Webspam dataset, verification time reduces from 902 seconds to 2 seconds, i.e., a reduction of two orders of magnitude. A similar picture can be drawn for L_1-attackers.

Scalability. In our second experiment, we assess the scalability of robustness verification by increasing the model size. In particular, we set the maximum number of leaves to 32 and we vary the number of trees in the ensemble trained on the FMNIST dataset to understand the trend of the robustness verification times. Results are shown in Figure 2, where we plot the speedup of CARVE-GBM over competitors. As we can see, the speedup ranges from around 10 times to more than 400 times for the largest ensembles of 125 trees, i.e., verification times are reduced by at least one order of magnitude. Remarkably, this positive finding is also artificially penalized by the enforcement of a 30 seconds timeout per instance on SILVA and MILP. If these tools were allowed to run for a longer time, the comparison would be even more in favor of CARVE-GBM, which is able to analyze each instance in less than one second.

4. Using the full test set can only improve the picture in favour of CARVE-GBM, in particular for the Webspam dataset, whose full test set includes around 100,000 instances; this sheer size would magnify the difference in the measured verification times.

Dataset	p	k	Verification Time	
			Baseline	CARVE-GBM
FMNIST	∞	0.005	172	12
		0.010	290	3
		0.015	3,448	4
	0	1	23	1
		2	23	1
		3	23	1
	1	0.05	74	8
		0.10	37	4
		0.15	39	4
MNIST	∞	0.01	1	13
		0.02	4	4
		0.03	3	3
	0	1	23	1
		2	23	1
		3	23	1
	1	0.05	157	8
		0.10	42	3
		0.15	46	4
Webspam	∞	0.0004	216	5
		0.0006	989	4
		0.0008	2,142	4
	0	1	902	2
		2	902	2
		3	902	2
	1	0.002	278	5
		0.003	179	4
		0.004	424	5

TABLE 4: Robustness verification times (in seconds) for the first 500 instances in the test set.

7. Comparison With Large-Spread Ensembles

We here compare our large-spread boosted ensembles with the large-spread ensembles with hard majority voting proposed in [10] in terms of accuracy and robustness. We recall that the large-spread ensembles from [10] are forests of independently trained classification trees using hard majority voting, while our large-spread boosted ensembles are ensembles of regression trees based on additive aggregation, trained using Gradient Boosting, a state-of-the-art boosting scheme.

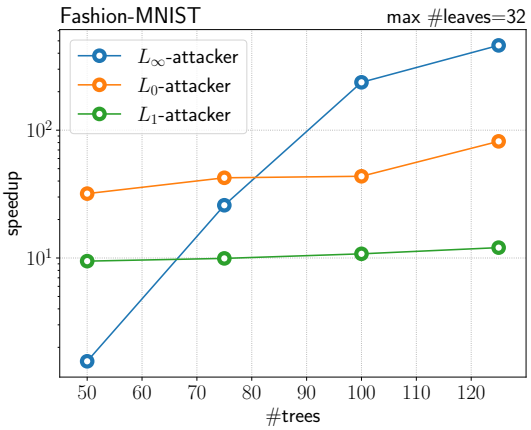


Figure 2: Speedup of the robustness verification time enabled by the use of CARVE-GBM over competitors.

We perform the experimental comparison using the same experimental methodology used in [10]. In particular, we use the FMNIST, MNIST, and Webspam datasets as described in Section 6.1, and we consider ensembles with 101 trees and maximum depth 6. We use the same hyper-parameter values reported in [10] to train the large-spread ensembles since the training algorithm for large-spread ensembles is guaranteed to converge for those values and give the best models in terms of accuracy and robustness. We then train large-spread boosted ensembles of the same size, setting the number of leaves to 64. We perform the training of the models for the L_∞, L_1, L_2 norms and with $k = \{0.005, 0.01, 0.015\}$ for FMNIST and MNIST and $k = \{0.0002, 0.0004, 0.0006\}$ for Webspam as done in [10].

Table 5 shows the accuracy and robustness of large-spread ensembles, abbreviated as LSE, and large-spread boosted ensembles, abbreviated as LSBE. We obtain the robustness of the large-spread ensembles using CARVE [10], while we use CARVE-GBM to verify the robustness of our large-spread boosted ensembles. We do not compare the performance of the two verifiers since robustness verification is performed in a matter of seconds by both tools. We observe that large-spread boosted ensembles present an accuracy higher or, at worst, equal to the one of large-spread ensembles on all the considered datasets. The gain in accuracy is particularly significant on the Webspam dataset. For example, for $k = 0.0006$ and $p = \{\infty, 1, 2\}$, the accuracy of the large-spread ensemble is 0.85, while the accuracy of the large-spread boosted ensemble is 0.97 (+0.12). This is a consequence of the adoption of the training algorithm based on Gradient Boosting, that trains trees that compensate for the errors of the predecessors, while the training algorithm in [10] trains independent trees. Our large-spread boosted ensembles are also more robust than large-spread ensembles in general. Again, the highest gain in robustness is achieved on the Webspam dataset. For example, for $k = 0.0006$ and $p = \infty$, the robustness of the large-spread ensemble is 0.82, while the robustness of our large-spread boosted ensemble is 0.96 (+0.14). The same gain is observed for the same value of k and $p \in \{1, 2\}$. This achievement in robustness is a consequence of both the large-spread condition and the gain in accuracy obtained by the large-spread boosted ensembles with respect to the large-spread ensembles, since the definition of robustness requires the correct classification of the tested instances to be verified.

Dataset	p	k	Accuracy		Robustness	
			LSE	LSBE	LSE	LSBE
FMNIST	∞	0.005	0.96	0.97	0.93	0.95
		0.010	0.94	0.97	0.91	0.92
		0.015	0.92	0.97	0.89	0.88
	1	0.005	0.96	0.97	0.94	0.95
		0.010	0.94	0.97	0.93	0.94
		0.015	0.92	0.97	0.91	0.94
	2	0.005	0.96	0.97	0.93	0.95
		0.010	0.94	0.97	0.91	0.94
		0.015	0.92	0.97	0.89	0.92
MNIST	∞	0.005	0.99	0.99	0.97	0.99
		0.010	0.99	0.99	0.97	0.99
		0.015	0.99	0.99	0.94	0.99
	1	0.005	0.99	0.99	0.98	0.99
		0.010	0.99	0.99	0.98	0.99
		0.015	0.99	0.99	0.97	0.99
	2	0.005	0.99	0.99	0.98	0.99
		0.010	0.99	0.99	0.98	0.99
		0.015	0.99	0.99	0.95	0.99
Webspam	∞	0.0002	0.91	0.98	0.90	0.97
		0.0004	0.89	0.98	0.86	0.97
		0.0006	0.85	0.97	0.82	0.96
	1	0.0002	0.91	0.98	0.90	0.98
		0.0004	0.89	0.98	0.87	0.97
		0.0006	0.85	0.97	0.83	0.97
	2	0.0002	0.91	0.98	0.90	0.97
		0.0004	0.89	0.98	0.86	0.97
		0.0006	0.85	0.97	0.82	0.96

TABLE 5: Accuracy and robustness (against $A_{p,k}$) measures for large-spread ensembles (LSE) models of [10] and our large-spread boosted ensembles (LSBE). The robustness against $A_{\infty,k}$ of the LSE models is obtained using CARVE [10], while we used CARVE-GBM to compute the robustness of the LSBE models.

8. Application to Security-Related Tasks

In this section, we show that our large-spread boosted ensembles can perform challenging security-related binary classification tasks, i.e., spam URLs and spam accounts detection, showing state-of-the-art performance. Moreover, our verification algorithm can be used to efficiently verify relevant security properties from previous work [13] on large-spread boosted ensembles. These properties can be encoded in our framework, yet they go beyond the traditional robustness property based on the L_p-norm.

8.1. Datasets and Security Properties

Datasets. We evaluate large-spread boosted ensem-

Dataset	Features	#train	#test	Distribution
Spam Accounts	15 (8)	36,000	4,000	49%/51%
Spam Urls	25 (4)	295,869	63,401	56%/44%

TABLE 6: Security-related datasets statistics

bles on two security-related datasets used in previous work [21], [22]: Twitter Spam Accounts [23], and Twitter Spam URLs [24]. Tree-based classifiers are well suited for classifying these tabular datasets related to binary classification tasks. Table 6 shows some statistics about the dataset. Recent work [13] analyzed the cost incurred by the attacker in perturbing each feature in the real world for the two datasets, defining a subset of low-cost features that the attacker can easily modify without incurring high economic costs or making the attack suspicious. We report the number of low-cost features for each dataset in round brackets.

Security Properties. Verifying the robustness of tree-based models in security-related tasks is fundamental. However, recent work [13] highlighted that the traditional robustness property may not suffice for assessing the security of tree-based classifiers in some specific security-related tasks while still being popular and suited for assessing the security in other tasks. For this reason, other properties have been defined to complement the traditional robustness property, specifically for security-related classification tasks. We here focus on three properties defined in [13] for the tasks of spam accounts detection and spam urls detection: Stability, Maximum Score Decrease, and Small Neighborhood.[5]

In the following definitions, we suppose that J is the subset of low-cost features. We define the properties below to be satisfied by a boosted tree ensemble T, but the definition can be extended to any classifier that returns a raw score prediction that is transformed by a monotonically increasing inverse link function. We first define *local stability* that states that, given an instance $\vec{x} \in \mathcal{X}$ and a low-cost feature i, for any instance $\vec{z}$ differing from it only for values of the feature i the difference between the raw score predictions $\widehat{T}(\vec{x})$ and $\widehat{T}(\vec{z})$ is bounded.

Definition 4 (Local Stability)**.** *Given a feature $i \in J$ and a constant $c \in \mathbb{R}$, the boosted tree ensemble T satisfies* local stability *on $\vec{x}$ if and only if $\forall \vec{z} \in \mathcal{X}.[\forall j \neq i.(z_j = x_j)] \implies |\widehat{T}(\vec{x}) - \widehat{T}(\vec{z})| \leq c$.*

The *local maximum score decrease* property instead bounds the maximum decrease in the raw score prediction on the instance $\vec{z}$ with respect to the raw score prediction on the instance $\vec{x}$ where $\vec{z}$ differs from $\vec{x}$ only for the values of the low-cost features.

Definition 5 (Local Maximum Score Decrease)**.** *Given a set of low-cost features J, a constant $c \in \mathbb{R}$ and an instance $\vec{x} \in \mathcal{X}$, the boosted tree ensemble T satisfies* local maximum score decrease *on $\vec{x}$ if and only if $\forall \vec{z} \in \mathcal{X}.[\forall i \notin J.(z_i = x_i)] \implies \widehat{T}(\vec{x}) - \widehat{T}(\vec{z}) \leq \iota^{-1}(c)$.*

Finally, the *local small neighborhood* property specifies that, given an instance $\vec{x}$, the raw score prediction $\widehat{T}(\vec{x})$ remains stable in a neighborhood of $\vec{x}$.

Definition 6 (Local Small Neighborhood)**.** *Let σ_i be the standard deviation of the input feature $i \in [1, d]$. Given constants $c, \varepsilon \in \mathbb{R}$ and an instance $\vec{x} \in \mathcal{X}$, the boosted tree ensemble T satisfies* local small neighborhood *on $\vec{x}$ if and only if $\forall \vec{z} \in \mathcal{X}.[max_i\{|z_i - x_i|/\sigma_i\} \leq \varepsilon] \implies |\widehat{T}(\vec{x}) - \widehat{T}(\vec{z})| \leq c \cdot \varepsilon$.*

Verification. We show in Appendix D how to encode the three properties in our framework and how to verify the properties efficiently using CARVE-GBM.

8.2. Experimental Methodology and Results

We perform our experimental evaluation using the training set and test set already provided for the two security-related tasks. We obtain a validation set by splitting the training set using 80-20 stratified random sampling.

We then use the following methodology:

1) We perform hyper-parameter tuning to identify the best-performing traditional GBDT model for each dataset in the same way as explained in section 6.1, with the exception that we look for the model with the best F1 score. We prefer the F1 score since it reflects the performance of the classifier on the recognition of the positive class, i.e., the spam, in particular, which is the principal target of spam detection, penalizing a high number of false positives and false negatives.
2) We perform the same hyper-parameter tuning to identify the best-performing large-spread model trained using our algorithm for each dataset. However, since the best large-spread model performs poorly in classifying the positive class on the Twitter spam URLs dataset, we additionally tune the weight of positive labels using line search from one to two with increments of 0.1. We refer to Appendix D for details about enforcing the large-spread condition for efficiently verifying each security-related property.

We set the values of constants of the definitions of the three properties as done in the related work [13]; we refer to Appendix D for the details about the specific used values.

Table 7 shows the accuracy and F1 score of large-spread models on the test sets. We report in round brackets the difference in accuracy and F1 score with respect to the corresponding best GBDT models. Moreover, we show the percentage of instances of the test set on which the three security-related robustness properties are satisfied by the large-spread models and the cumulative verification time using CARVE-GBM. We do not verify these properties on

5. We actually consider local versions of the global properties defined in [13], because our framework currently supports just the verification of traditional local properties.

Dataset	Property	Accuracy	F1 score	% Instances	Verification Time
Twitter Spam Accounts	Stability	0.95 (-0.00)	0.95 (-0.01)	100%	3
	Maximum Score Decrease	0.94 (-0.01)	0.94 (-0.02)	100%	1
	Small Neighborhood	0.93 (-0.02)	0.93 (-0.03)	100%	1
Twitter Spam URLs	Stability	0.97 (-0.02)	0.97 (-0.02)	100%	12
	Maximum Score Decrease	0.99 (-0.00)	0.98 (-0.01)	100%	8
	Small Neighborhood	0.99 (-0.00)	0.98 (-0.01)	100%	4

TABLE 7: Accuracy, F1 score, percentage of instances of the test set on which the property is satisfied by the best large-spread model, and time required (in seconds) for verifying the property on the test set for the best large-spread model. We report in round brackets the difference in accuracy and F1 score with respect to the best-performing traditional GBDT model. We do not report the verification time required by a baseline on the GBDT model since no baseline in the literature directly verifies these properties.

traditional GBDT models since no baseline allows us to verify them directly on these models.

The results highlight that large-spread models observe a negligible loss in accuracy and F1 score, i.e., at most three points, with respect to the best-performing traditional GBDT models on the two datasets. In addition, the results support that large-spread models are robust if evaluated with respect to security properties defined for specific security-related tasks. In particular, the large-spread models exhibit the three security properties on all the instances of the test sets of Twitter Spam Accounts and Twitter Spam URLs datasets, using the constants in [13] in the definition of the three properties. Finally, the verification of the three properties with CARVE-GBM is efficient: it requires at most three seconds for verifying each property on the Twitter Spam Accounts test set and 12 seconds at most for verifying the local stability property on the Twitter Spam URLs test set, which contains around 16 times the number of instances of the test set of the Twitter Spam Accounts dataset. These positive results highlight that large-spread boosted ensembles are applicable in challenging security-related tasks since they satisfy specific security properties. Moreover, CARVE-GBM verifies these properties in a matter of seconds.

9. Related Work

We already discussed that prior work studied the complexity of robustness verification for decision tree ensembles [6], [7]. This problem is NP-complete for arbitrary L_p-norm attackers, even when considering decision stump ensembles [7], [25]. Despite this negative result, prior work proposed several approaches to robustness verification for tree ensembles [6], [8], [9], [26], [27], [28], [29], [30]. Though effective in many cases, these techniques are bound to fail for large ensembles and complex datasets. We experimentally showed that state-of-the-art verification tools do not scale and are much less efficient than our approach based on large-spread ensemble training; thus, we generalize preliminary results on verifiable learning for simple random forest models [10] to the gradient boosting setting.

Numerous studies in the literature have delved into algorithms for training tree ensembles that exhibit robustness against evasion attacks [6], [21], [22], [25], [31], [32], [33], [34], [35], [36]. However, our work complements these endeavors. Our primary objective is not enforcing robustness; rather, robustness may emerge as a byproduct of our training algorithm. Our focus lies in facilitating efficient robustness verification for the trained models. Combining our approach with existing robust training algorithms may further improve their performance against evasion attacks.

Previous work also focused on how to turn ML models into models whose robustness can be certified. In particular, Randomized Smoothing [37] has been proposed to turn a generic classifier into a new classifier certifiably robust. While both Randomized Smoothing and our technique may induce a loss in the accuracy of the model for certifying robustness to large perturbations, our work shows significant differences to Randomized Smoothing. In particular, our verification algorithm provides deterministic robustness guarantees, while many certifiable methods based on Randomized Smoothing provide probabilistic ones. More recent work [38] proposes a Randomized Smoothing approach specific for tree-based models providing deterministic robustness guarantees, but it supports only decision stumps, while our technique supports generic boosted tree ensembles.

It is essential to note that our work addresses the classic definition of robustness, termed *local* robustness in recent literature on global robustness and related properties [13], [39], [40]. The aim of this line of research is to achieve security verification independent of the selection of a specific test set, thereby enhancing the credibility of security proofs. While acknowledging the popularity and relative ease of dealing with local robustness, we leave the extension of our framework to global robustness for future work.

Significant efforts have been invested in the robustness verification of deep neural networks (DNNs). Traditional approaches for exact verification, often based on Satisfiability Modulo Theories (SMT) [41], [42], [43] and integer linear programming [44], [45], [46], [47], frequently face scalability issues with large DNNs, similar to tree ensembles. To address these challenges, various proposals, such as pruning the original DNN [48] and identifying specific classes of DNNs for more efficient robustness verification [49], have been presented. Xiao et al. [50] introduced the concept of

co-designing model training and verification, emphasizing training models that balance accuracy and robustness to facilitate exact verification. While prior techniques offer empirical efficiency guarantees, our proposal stands out by providing a formal complexity reduction through the development of a polynomial time algorithm. Furthermore, our research focuses on tree ensembles rather than DNNs.

Lastly, recent work explored the adversarial robustness of model ensembles [51]. The key finding of this study demonstrated that a combination of "diversified gradient" and "large confidence margin" serves as sufficient and necessary conditions for certifiably robust ensemble models. While this result may not directly apply to non-differentiable models like decision tree ensembles, the idea of diversifying models aligns with our large-spread condition. We plan to explore connections with this proposition in future work.

10. Conclusion

In this work, we generalized existing endeavours on verifiable learning from simple ensembles based on hard majority voting to state-of-the-art boosted ensembles, such as those trained using LightGBM [12]. We formally characterized robustness verification for large-spread boosted ensembles in terms of an optimization problem and we proposed efficient techniques to solve it in practical cases. We experimentally showed on public datasets that our verifiable learning techniques allows one to train models offering state-of-the-art accuracy, while being amenable to efficient robustness verification. Our analysis also confirmed that robustness verification for traditional tree ensembles does not scale when increasing the model size, thus reinforcing the importance of verifiable learning.

As future work, we would like to investigate different approaches to verifiable learning for tree ensembles, besides the large-spread condition advocated in existing work. Moreover, we plan to study how verifiable learning can be applied to fundamentally different classes of machine learning models, such as DNNs.

Acknowledgements. We thank the reviewers for their constructive feedback, which has greatly contributed to the improvement of this paper. This research was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, by the European Union - Next-GenerationEU - PNRR – M.4 C.2, I.1.1 - PRIN 2022 WHAM!, 2022ZZX57L, H53D23003750006 and by iNEST (Interconnected NordEst Innovation Ecosystem, Project ID: ECS 00000043), funded by PNRR (Mission 4.2, Investment 1.5) NextGeneration EU.

References

[1] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *ECML PKDD*, 2013.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLR*, 2014.

[3] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, "Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection," *ACM Trans. Priv. Secur.*, vol. 24, no. 4, pp. 27:1–27:31, 2021.

[4] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.

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

[6] A. Kantchelian, J. D. Tygar, and A. D. Joseph, "Evasion and hardening of tree ensemble classifiers," in *ICML*, 2016.

[7] Y. Wang, H. Zhang, H. Chen, D. S. Boning, and C. Hsieh, "On lp-norm robustness of ensemble decision stumps and trees," in *ICML*, 2020.

[8] H. Chen, H. Zhang, S. Si, Y. Li, D. S. Boning, and C. Hsieh, "Robustness verification of tree-based models," in *NeurIPS*, 2019.

[9] F. Ranzato and M. Zanella, "Abstract interpretation of decision tree ensemble classifiers," in *AAAI*, 2020.

[10] S. Calzavara, L. Cazzaro, G. E. Pibiri, and N. Prezza, "Verifiable learning for robust tree ensembles," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, CA, USA, November 26-30, 2023*, 2023.

[11] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[13] Y. Chen, S. Wang, Y. Qin, X. Liao, S. Jana, and D. A. Wagner, "Learning security classifiers with verified global robustness properties," in *ACM CCS*, 2021.

[14] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007. [Online]. Available: https://www.worldcat.org/oclc/71008143

[15] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: https://doi.org/10.1006/jcss.1997.1504

[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[17] D. Pisinger and P. Toth, "Knapsack problems," *Handbook of Combinatorial Optimization: Volume1–3*, pp. 299–428, 1998.

[18] "Fashion-MNIST Dataset," https://www.openml.org/search?type=data&sort=runs&id=40996&status=active, accessed: 2024-10-15.

[19] "MNIST Dataset," https://www.openml.org/search?type=data&sort=runs&id=554&status=active, accessed: 2024-10-15.

[20] "Webspam Dataset," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html, accessed: 2024-10-15.

[21] H. Chen, H. Zhang, D. S. Boning, and C. Hsieh, "Robust decision trees against adversarial examples," in *ICML*, 2019.

[22] Y. Chen, S. Wang, W. Jiang, A. Cidon, and S. Jana, "Cost-aware robust tree ensembles for security applications," in *USENIX Security Symposium*, 2021.

[23] K. Lee, B. D. Eoff, and J. Caverlee, "Seven months with the devils: A long-term study of content polluters on twitter," in *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*, L. A. Adamic, R. Baeza-Yates, and S. Counts, Eds. The AAAI Press, 2011. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2780

[24] H. Kwon, M. B. Baig, and L. Akoglu, "A domain-agnostic approach to spam-url detection via redirects," in *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. Kim, K. Shim, L. Cao, J. Lee, X. Lin, and Y. Moon, Eds., vol. 10235, 2017, pp. 220–232. [Online]. Available: https://doi.org/10.1007/978-3-319-57529-2_18

[25] M. Andriushchenko and M. Hein, "Provably robust boosted decision stumps and trees against adversarial attacks," in *NeurIPS*, 2019.

[26] J. Törnblom and S. Nadjm-Tehrani, "Formal verification of input-output mappings of tree ensembles," *Sci. Comput. Program.*, vol. 194, p. 102450, 2020.

[27] S. Calzavara, P. Ferrara, and C. Lucchese, "Certifying decision trees against evasion attacks by program analysis," in *ESORICS*, 2020.

[28] L. Devos, W. Meert, and J. Davis, "Verifying tree ensembles by reasoning about potential instances," in *SDM*, 2021.

[29] G. Einziger, M. Goldstein, Y. Sa'ar, and I. Segall, "Verifying robustness of gradient boosted models," in *AAAI*, 2019.

[30] N. Sato, H. Kuruma, Y. Nakagawa, and H. Ogawa, "Formal verification of a decision-tree ensemble model and detection of its violation ranges," *IEICE Trans. Inf. Syst.*, vol. 103-D, no. 2, pp. 363–378, 2020.

[31] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, "Treant: training evasion-aware decision trees," *Data Min. Knowl. Discov.*, vol. 34, no. 5, pp. 1390–1420, 2020.

[32] D. Vos and S. Verwer, "Efficient training of robust decision trees against adversarial examples," in *ICML*, 2021.

[33] F. Ranzato and M. Zanella, "Genetic adversarial training of decision trees," in *GECCO*, 2021.

[34] J. Guo, M. Teng, W. Gao, and Z. Zhou, "Fast provably robust decision trees and boosting," in *ICML*, 2022.

[35] D. Vos and S. Verwer, "Robust optimal classification trees against adversarial examples," in *AAAI*, 2022.

[36] ——, "Adversarially robust decision tree relabeling," in *ECML PKDD*, 2022.

[37] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 1310–1320. [Online]. Available: http://proceedings.mlr.press/v97/cohen19c.html

[38] M. Z. Horváth, M. N. Müller, M. Fischer, and M. T. Vechev, "(de-)randomized smoothing for decision stump ensembles," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/146b4bab3f8536a07905f25d367b4924-Abstract-Conference.html

[39] S. Calzavara, L. Cazzaro, C. Lucchese, F. Marcuzzi, and S. Orlando, "Beyond robustness: Resilience verification of tree-based classifiers," *Comput. Secur.*, vol. 121, 2022.

[40] K. Leino, Z. Wang, and M. Fredrikson, "Globally-robust neural networks," in *ICML*, 2021.

[41] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *CAV*, 2017.

[42] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *CAV*, 2019.

[43] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV*, 2017.

[44] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *NeurIPS*, 2016.

[45] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *CoRR*, vol. abs/1706.07351, 2017.

[46] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *ICLR*, 2019.

[47] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NFM*, 2018.

[48] D. Guidotti, F. Leofante, L. Pulina, and A. Tacchella, "Verification of neural networks: Enhancing scalability through pruning," in *ECAI*, 2020.

[49] K. Jia and M. C. Rinard, "Efficient exact verification of binarized neural networks," in *NeurIPS*, 2020.

[50] K. Y. Xiao, V. Tjeng, N. M. M. Shafiullah, and A. Madry, "Training for faster adversarial robustness verification via inducing relu stability," in *ICLR*, 2019.

[51] Z. Yang, L. Li, X. Xu, B. Kailkhura, T. Xie, and B. Li, "On the certified robustness for ensemble models and beyond," in *ICLR*, 2022.

[52] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

Appendix A. Proof of Theorem 1

Proof. We prove the two directions separately:

($\Rightarrow$) Assume that $BV(T, \vec{x}, y, p, k)$ returns True. In this case we know that $T(\vec{x}) = y$, hence we just need to prove that $\forall z \in A_{p,k}(\vec{x}) : T(\vec{z}) = y$ to conclude. Since $BV(T, \vec{x}, y, p, k)$ returns True, we know that $T(\vec{x}+\vec{\delta}_{opt}) = y$ for the adversarial perturbation $\vec{\delta}_{opt}$ constructed by the algorithm. Assume for simplicity that $y = +1$, a similar argument applies to the case $y = -1$. Since $T(\vec{x}) = +1$, we know that $\iota(\widehat{T}(\vec{x})) \geq \tau$. We now formalize the following intuition: since ι is monotonically increasing, the best strategy to force a prediction error is lowering $\widehat{T}(\vec{x})$ as much as possible to push it below τ, so that the predicted class becomes -1. Since T is large-spread, any evasion attack against T can be decomposed into a sum of pairwise orthogonal perturbations $\vec{\delta}_1, \ldots, \vec{\delta}_m$ that do not affect the prediction paths of different trees when they are added together [10]. Let $\vec{\delta} = \sum_{i=1}^{m} \vec{\delta}_i$ and, for each $\vec{\delta}_i$, let $\lambda(s_{ij})$ be the leaf that it is reached by $t_i(\vec{x}+\vec{\delta}_i)$, which is the same leaf reached by $t_i(\vec{x}+\vec{\delta})$. This implies that the sum of the adversarial gains $G(t_i, \vec{x}, y, \lambda(s_{ij}))$ is equal to $\widehat{T}(\vec{x}) - \widehat{T}(\vec{x}+\vec{\delta})$, i.e., such sum identifies how much $\widehat{T}(\vec{x})$ can be lowered by $\vec{\delta}$. Note that we are implicitly using constraint (3) here, since we consider leaves belonging to different trees. Now observe that $T(\vec{x}+\vec{\delta}_{opt}) = +1$ implies that $\iota(\widehat{T}(\vec{x}+\vec{\delta}_{opt})) \geq \tau$. Since $\vec{\delta}_{opt}$ is the perturbation maximizing the sum of the adversarial gains, i.e., $\vec{\delta}_{opt}$ lowers $\widehat{T}(\vec{x})$ as much as possible, we have that $\forall z \in A_{p,k}(\vec{x}) : \widehat{T}(\vec{z}) \geq \tau$, hence $\forall z \in A_{p,k}(\vec{x}) : T(\vec{z}) = +1$.

($\Leftarrow$) Assume that T is robust on $\vec{x}$. In this case we know that $T(\vec{x}) = y$, hence we just need to prove that $T(\vec{x}+\vec{\delta}_{opt}) = y$ for the adversarial perturbation $\vec{\delta}_{opt}$ constructed by the algorithm to conclude. Since T is robust on $\vec{x}$, we know that $\forall z \in A_{p,k}(\vec{x}) : T(\vec{z}) = y$. Hence, we just need to show that $\vec{x}+\vec{\delta}_{opt} \in A_{p,k}(\vec{x})$, which is equivalent to proving that $||\vec{\delta}_{opt}||_p \leq k$. This follows from constraint (2) by definition of $\vec{\delta}_{opt}$.

□

Appendix B. Proof of Theorem 2

Proof. The intuition of the proof is similar to the idea used in the proof of Theorem 1, hence we provide just a proof sketch based on Figure 3. The picture shows how the inverse link function ι (in blue) maps a raw prediction $\hat{T}(\vec{x})$ on the x-axis to a class label on the y-axis via the threshold τ. The ι function shown here is just an example; it is not necessarily a linear function, but it can be an arbitrary monotonically increasing function. Let $s_\tau = \iota^{-1}(\tau)$ be the decision boundary separating the two classes. The model is *not* robust on the instance $\vec{x}$ when scenario (a) happens. In this case, an attack is possible: for an instance with label $y = +1$, raw score prediction $\hat{T}(\vec{x}) = s^+$ and maximum adversarial gain Γ^+, we have $s^+ - \Gamma^+ < s_\tau$, thus the instance is assigned the wrong class by ι as $\iota(s^+ - \Gamma^+) < \tau$. Symmetrically, for an instance with label $y = -1$, raw score prediction $\hat{T}(\vec{x}) = s^-$, and maximum adversarial gain Γ^-, we have $s^- + \Gamma^- \geq s_\tau$. Scenario (b) shows instead the case where no attack is possible and the model is robust, because the maximum adversarial gain is not sufficient to change the assigned label. □

Appendix C. Proof of Theorem 3

Proof. We show a reduction from the Subset Sum Problem (SSP) [52] to robustness verification for large-spread boosted ensembles. We consider the variant of SSP where all inputs are positive, which is still NP-hard. For simplicity, we define the reduction for the case $p = 1$ and we then explain how the construction can be generalized to an arbitrary $p \in \mathbb{N} \cup \{0\}$.

Let $\mathbb{S}$ be a multiset of integers and let $\mathbb{G}$ be an integer, the goal of SSP is determining whether there exists any subset of $\mathbb{S}$ which sums to $\mathbb{G}$. Let $\mathbb{S} = \{s_1, \ldots, s_m\}$ be a multiset of m integers and $\zeta = \frac{1}{m+1}$. We construct a large-spread boosted ensemble T with m trees such that T is robust against $A_{1,\mathbb{G}}$ on the instance $\vec{x} = \langle \zeta, \ldots, \zeta \rangle$ with true label -1 if and only if there does *not* exist a subset of $\mathbb{S}$ which sums to $\mathbb{G}$. Each tree t_i is a regression stump, i.e., a regression tree of depth 1, built on top of a different feature i. Specifically, $t_i = \sigma(i, s_i, \lambda(0), \lambda(s_i))$. We stipulate that the inverse link function ι used by the ensemble T is the identity function and that the threshold τ is set to $\mathbb{G}$.

We have that $\widehat{T}(\vec{x}) = 0 < \tau$, hence $T(\vec{x}) = -1$. This means that T is robust on $\vec{x}$ if and only if there does not exist any $\vec{z} \in A_{1,\mathbb{G}}(\vec{x})$ such that $\widehat{T}(\vec{z}) \geq \tau$. Since T is large-spread, finding such $\vec{z}$ is equivalent to finding pairwise orthogonal perturbations $\vec{\delta}_1, \ldots, \vec{\delta}_m$ such that $||\sum_{i=1}^{m} \vec{\delta}_i||_1 \leq \mathbb{G}$ and $\sum_{i=1}^{m} t_i(\vec{x}+\vec{\delta}_i) \geq \tau = \mathbb{G}$. Let $\{\vec{\delta}_j\}_j$ be the multiset of the non-zero perturbations, we now observe the following inequalities.

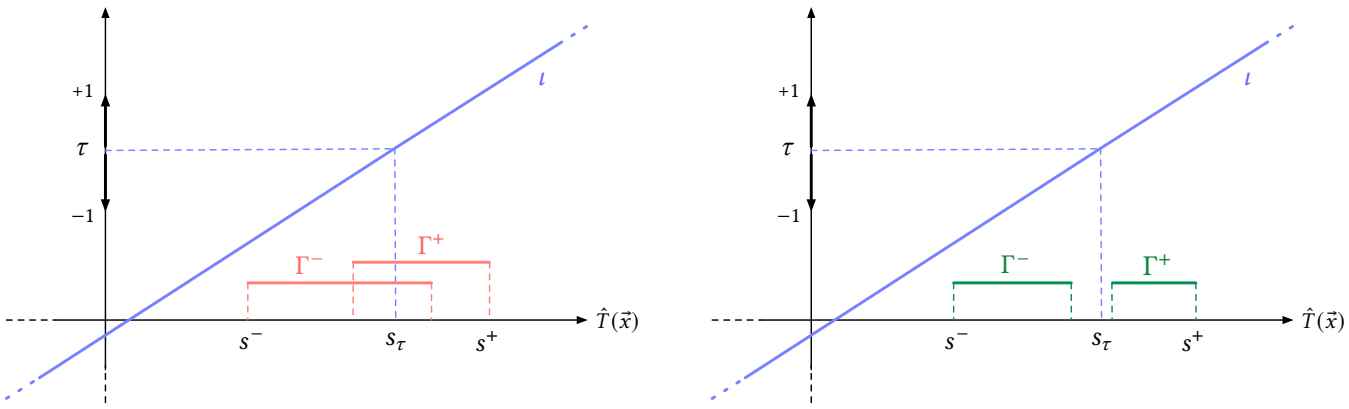
1) $||\sum_{i=1}^{m} \vec{\delta}_i||_1 = ||\sum_j \vec{\delta}_j||_1 = \sum_j ||\vec{\delta}_j||_1 = \sum_j (s_j - \zeta) \leq \mathbb{G}$
2) $\sum_{i=1}^{m} t_i(\vec{x}+\vec{\delta}_i) = \sum_j t_j(\vec{x}+\vec{\delta}_j) = \sum_j s_j \geq \mathbb{G}$

By using the first inequality, we obtain:

$$\sum_j s_j \leq \mathbb{G} + \sum_j \zeta \leq \mathbb{G} + m \cdot \zeta < \mathbb{G} + 1.$$

Combined with the second inequality, this leads to:

$$\mathbb{G} \leq \sum_j s_j < \mathbb{G} + 1,$$



(a) attack **(b)** no attack

Figure 3: Correctness of robustness verification for the efficient algorithm EV.

which implies $\sum_j s_j = \mathbb{G}$, because all the elements of the summation are integers. We conclude that T is robust on $\vec{x}$ if and only if there does not exist $\{s_j\}_j$ such that $\sum_j s_j = \mathbb{G}$, i.e., SSP does not have a solution. Of course, this reduction operates in polynomial time.

The reduction generalizes to the case $p > 1$ by observing that, when working with large-spread ensembles, the norm of the sum of adversarial perturbations can be related to the sum of the their norms. In particular, for any set of pairwise orthogonal perturbations $\{\delta_i\}_i$ and any $p \in \mathbb{N}$, we have $||\sum_i \vec{\delta}_i||_p = (\sum_i ||\vec{\delta}_i||_p^p)^{1/p}$. Hence, the reduction can be performed by making the following changes: set $t_i = \sigma(i, s_i^p, \lambda(0), \lambda(s_i^p))$, increase the maximum adversarial perturbation from $\mathbb{G}$ to $\mathbb{G}^p$, and similarly set the threshold $\tau = \mathbb{G}^p$.

Finally, the case $p = 0$ uses a different reduction, but the underlying intuition remains the same. The constructed large-spread ensemble includes regression trees, rather than regression stumps. For each integer $s_i \in \mathbb{S}$ we generate s_i features, call them $f_1^i, \ldots, f_{s_i}^i$. The corresponding regression tree t_i is a right chain of depth s_i, where we inductively define the sub-tree rooted at layer j as $t'_j = \sigma(f_j^i, 1, \lambda(0), t'_{j+1})$ with the base case $t'_{s_i} = \sigma(f_{s_i}^i, 1, \lambda(0), \lambda(s_i))$. This way, a successful evasion attack against t_i requires a perturbation $\vec{\delta}_i$ such that $||\vec{\delta}_i||_0 = s_i$ and the corresponding adversarial gain is s_i. Note that, in this setting, for each tree there is only one relevant leaf $\lambda(s_i)$ to consider, i.e., the leaf providing a positive adversarial gain s_i. Since we do not need to materialize every leaf in t_i, the reduction is performed in polynomial time. Correctness follows by the same argument used for the case $p = 1$. □

Appendix D. Verification of Security-Related Properties

We show here how to encode in our framework the three properties defined in Section 8.1. Furthermore, we show that it is possible to verify them on large-spread boosted ensembles by solving Problem 1. Finally, we provide more details about the experimental methodology presented in Section 8.2.

Verification. We first discuss how to modify the procedure to find the minimal adversarial perturbation required to push $\vec{x}$ into any given leaf $\lambda(s)$ of a tree in the ensemble. We want to take into account the fact that an attacker can easily modify only the low-cost features in J. Let $H = (l_1, r_1] \times \ldots \times (l_d, r_d]$ be the hyper-rectangle annotating $\lambda(s)$, we define $dist(\vec{x}, \lambda(s)) = \vec{\delta} \in \mathbb{R}^d$ as:

$$\forall i \in [1, d] : \delta_i = \begin{cases} 0 & \text{if } x_i \in H_i = (l_i, r_i] \\ l_i - x_i + \varepsilon & \text{if } x_i \leq l_i \wedge i \in J \\ r_i - x_i & \text{if } x_i > r_i \wedge i \in J \\ \infty & \textit{otherwise.} \end{cases}$$

The value $||\vec{\delta}||_p$ will be ∞ for $p \neq 0$ if one of the components of $\vec{\delta}$ corresponding to a feature not in J is not 0, i.e., a feature that is not low-cost has been perturbed. When $p = 0$, we assign ∞ to $||\vec{\delta}||_0$ if at least one component of $\vec{\delta}$ is ∞.

Second, we discuss how to formalize the attacker describing the instances $\vec{z}$ different from $\vec{x}$ of each definition. Since the large-spread condition (Definition 2) is defined with respect to an attacker $A(\vec{x})$, defining the attacker is fundamental to understand how to enforce the large-spread condition to efficiently verify each security-related property. Given $\vec{x} \in \mathcal{X}$, the local stability property predicates over all the instances that differ from $\vec{x}$ on only one of the low-cost features in J. Thus, given the feature $i \in J$, an attacker including exactly these instances can be formalized as $A(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid ||\vec{z} - \vec{x}||_0 \leq 1 \wedge (\forall j \neq i : x_j = z_j)\}$. The large-spread property with respect to this attacker can be then enforced for L_0-norm only on the feature i. Instead, the local maximum score decrease property predicates over all the instances that differ from $\vec{x}$ in the value of at least one feature in J. Again, the attacker including all these instances can be formalized as $A(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid (||\vec{z} - \vec{x}||_0 \leq |J|) \wedge (\forall j \notin J : x_j = z_j)\}$. The large-spread property with respect to this attacker can then be enforced for the L_0-norm

on all the features in J. Finally, the local small neighborhood property predicates over all the instances $\vec{z}$ such that the absolute value of the difference of the values of the feature i of $\vec{z}$ and $\vec{x}$ divided by σ_i is at most ε, where σ_i is the standard deviation of the feature i, for every feature $i \in [1, d]$. The attacker including all these instances can be formalized as $A(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid \forall i \in [1, d] : |z_i - x_i|/\sigma_i \leq \varepsilon\}$. The large-spread condition can then be enforced on the boosted tree ensemble by requiring that the distance between each threshold of the feature i (not necessarily low-cost) on different trees divided by σ_i must be greater than $2 \cdot \varepsilon$.

Now, we are ready to discuss how to verify the three local properties. Given the large-spread boosted ensemble T and $\vec{x} \in \mathcal{X}$, verifying the local stability and local small neighborhood properties requires verifying $\forall \vec{z} \in A(\vec{x}). |\widehat{T}(\vec{x}) - \widehat{T}(\vec{z})| \leq c$, where $A(\vec{x})$ is the attacker of the corresponding property. Instead, local maximum score decrease requires to verify $\forall \vec{z} \in A(\vec{x}). \widehat{T}(\vec{x}) - \widehat{T}(\vec{z}) \leq c$. Note that c represents a generic constant here, to be substituted with the constants or product of constants appearing in the definitions of the properties. To verify these properties, we can leverage the efficient solving algorithm of Problem 1 after choosing the appropriate true label y, setting τ properly, and supposing that the boosted tree ensemble satisfies the large-spread condition corresponding to the attacker of the property to verify. We show how to verify $\widehat{T}(\vec{z}) - \widehat{T}(\vec{x}) \leq c$ and $\widehat{T}(\vec{z}) - \widehat{T}(\vec{x}) \geq -c$, following the intuition provided in the proof of Theorem 2:

1) Given $\vec{x} \in \mathcal{X}$ and $A(\vec{x})$ of the property, to verify $\widehat{T}(\vec{z}) - \widehat{T}(\vec{x}) \leq c$, set $y = -1$ and solve Problem 1 for the given input to determine the maximum total adversarial gain Γ. Γ represent the maximum increase of the raw score of $\vec{x}$ that they can obtain by applying the optimal perturbation to $\vec{x}$. Let $\widehat{T}(\vec{x}) = s$ and $\tau = \iota(s + c)$. If $\iota(s + \Gamma) \leq \tau = \iota(s + c)$, then $s + \Gamma \leq s + c$ and $\Gamma \leq c$, since ι is monotonically increasing. Then, return True if $\iota(s + \Gamma) \leq \tau$, False otherwise.
2) Given $\vec{x} \in \mathcal{X}$ and $A(\vec{x})$ of the property, to verify $\widehat{T}(\vec{z}) - \widehat{T}(\vec{x}) \geq -c$, set $y = +1$ and solve Problem 1 for the given input to determine the maximum total adversarial gain Γ. Γ represent the maximum decrease of the raw score of $\vec{x}$ that they can obtain by applying the optimal perturbation to $\vec{x}$. Let $\widehat{T}(\vec{x}) = s$ and $\tau = \iota(s - c)$. If $\iota(s - \Gamma) \geq \tau = \iota(s - c)$, then $s - \Gamma \geq s - c$ and $\Gamma \leq c$, since ι is monotonically increasing. Then, return True if $\iota(s - \Gamma) \geq \tau$, False otherwise.

Details of the Experimental Methodology. Here, we provide more details about the methodology of the experimental evaluation in Section 8.2.

We set the values of constants from the definition of the three properties as done in the related work [13]: $c = 8$ for local stability; $c = 0.98$ for local maximum score decrease; $\varepsilon = 0.1$ and $c = 50$ for the Twitter Spam Accounts dataset and $\varepsilon = 1.5$ and $c = 10$ for the Twitter Spam URLs dataset for local small neighborhood. Since LightGBM uses the logistic function to compute the probabilities from the raw score predictions, we set ι^{-1} from the local maximum score decrease property to the logit function. Finally, we test local stability on the feature *NumDailyTweets* of the Twitter Spam Accounts dataset and *Tweet Count* of the Twitter Spam URLs dataset (we refer to [13] for details about the features of the two datasets), while we suppose that the attacker of the local maximum score decrease property is not allowed to perturb multiple low-cost features (as supposed in [13]).

Appendix E.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper focuses on robustness verification for decision tree classifiers. It identifies a class of classifiers – large-spread boosted tree ensembles – for which efficient verification is possible.

E.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

1) This paper extends verifiable learning from basic ensemble methods to state-of-the-art boosted tree ensembles, formulating an optimal attack strategy and proposing two verification algorithms. This appears to be helpful and valuable for building better robustness verification methods in tree-based models. The formulation of optimal attack strategy and the solutions may provide insights for robustness verification.
2) The application to security-related tasks also shows the importance.
3) The paper is well-motivated, and the formal proof and the evaluation results demonstrate the effectiveness of the proposed algorithm.

E.4. Noteworthy Concerns

1) The paper only considers the binary classification problem and all experiments are conducted in binary classification case. It is unclear how the proposed methods work and perform in multi-class cases in practice.