Beyond Robustness: Resilience Verification of Tree-Based Classifiers

Stefano Calzavara[a], Lorenzo Cazzaro[a], Claudio Lucchese[a], Federico Marcuzzi[a], Salvatore Orlando[a]

[a]*Ca' Foscari University of Venice*

Abstract

In this paper we criticize the robustness measure traditionally employed to assess the performance of machine learning models deployed in adversarial settings. To mitigate the limitations of robustness, we introduce a new measure called resilience and we focus on its verification. In particular, we discuss how resilience can be verified by combining a traditional robustness verification technique with a data-independent stability analysis, which identifies a subset of the feature space where the model does not change its predictions despite adversarial manipulations. We then introduce a formally sound data-independent stability analysis for decision trees and decision tree ensembles, which we experimentally assess on public datasets and we leverage for resilience verification. Our results show that resilience verification is useful and feasible in practice, yielding a more reliable security assessment of both standard and robust decision tree models.

Keywords: adversarial machine learning, decision trees, robustness

1. Introduction

Machine Learning (ML) is becoming more and more popular nowadays, in particular for classification tasks, yet it is acknowledged to be susceptible to different types of attacks. A number of research papers showed that classifiers trained using standard ML algorithms cannot be deployed in security-sensitive settings, because they are easily fooled in practice and their performance undergoes significant downgrade when their inputs are subject to adversarial manipulations, imperceptible to human experts [1, 2]. This motivated the development of new performance measures like *robustness*, which generalize traditional measures like accuracy to account for the threats of adversarial manipulations at test time [3, 4]. Specifically, given an input $\vec{x}$ and its correct class y, robustness requires the classifier to predict the class y also for all the adversarial manipulations $A(\vec{x})$, rather than just for the original input $\vec{x}$.

Robustness is certainly an intuitive and desirable property to estimate the performance of classifiers deployed in adversarial settings, yet it is sub-optimal because it is strongly dependent on the choice of a specific input $\vec{x}$. While the performance of classifiers must indeed be empirically estimated on a set of correctly labeled inputs (test set), such inputs are normally assumed to be sampled from an underlying data distribution and robustness tells nothing about unsampled data. In other words, a robustness proof for $\vec{x}$ does not provide any guarantee about any other input $\vec{z}$ close to $\vec{x}$ which could have been sampled in place of it. This is concerning, because the actual inputs of the classifier at test time will be different samples drawn from the same distribution of $\vec{x}$, which are not covered by standard security assessments based on robustness. This problem has been independently acknowledged in very recent work on *global robustness* properties [5, 6], which advocates the need for verification techniques establishing robustness guarantees on all the possible inputs provided to the classifier. Unfortunately, these efforts are still at an early stage and there is no uniform "one size fits all" definition of global robustness that can be readily used to verify the security of classifiers (see Section 6 for a discussion). Our work falls in the same research line of global

robustness properties, yet it takes a different direction to better complement the extensive amount of work on robustness verification.

In particular, we here propose a generalization of robustness, called *resilience*, designed to make the security assessment of classifiers more reliable. Resilience generalizes traditional robustness guarantees from a specific test set to all the other possible test sets which could have been sampled in place of it, i.e., which are close to it given an appropriate definition of neighborhood. Resilience thus provides a more conservative account of the security of classifiers than robustness, while retaining its intuitive flavour. Most importantly, the connection between resilience and robustness allows one to leverage traditional tools for robustness verification as the first step of a resilience verification pipeline, thus integrating with significant research efforts spent on robustness verification.

Contributions. In the present paper we make the following contributions:

1. We criticize the traditional robustness measure used to estimate the security of classifiers against evasion attacks and we propose an improved measure called resilience. We then discuss how resilience can be estimated by combining an arbitrary robustness verification technique with a *data-independent stability analysis*, which identifies a subset of the feature space where the classifier does not change its predictions despite adversarial manipulations at test time. The analysis is data-independent because it is based on the classifier alone, rather than on a specific test set (contrary to robustness). We finally present a simple technique to turn any classifier into a globally robust classifier (in the sense of [6]) by leveraging such data-independent stability analysis, thus clarifying the connections with recent work in the area (Section 3).
2. We propose a data-independent stability analysis for decision trees and decision tree ensembles, a popular class of ML models [7]. The stability analysis is based on *symbolic attacks*, i.e., symbolic representations of a set of instances along with their (relevant) adversarial manipulations, which support the analysis of tree-based classifiers independently of a specific test set. Our analysis is proved sound and can be readily leveraged to establish both robustness and resilience proofs for tree-based classifiers (Section 4).
3. We implement our data-independent stability analysis[1] and we experimentally assess its effectiveness on public datasets, by estimating the robustness and resilience of both standard and robust tree models trained using a state-of-the-art adversarial ML algorithm (Section 5).

Our experimental evaluation shows that resilience verification is both useful and feasible in practice, yielding a more reliable security assessment of classifiers deployed in adversarial settings. In particular, our experiments show that robustness can be significantly affected by the choice of a specific test set, hence it may give a false sense of security, while resilience is effective at discriminating between secure models and models which turned out to be robust just *by accident*, i.e., thanks to a lucky, specific sampling of the test set. We thus recommend the use of resilience for the security verification of ML models deployed in adversarial scenarios.

2. Background

We introduce here the key technical ingredients required to appreciate this work.

2.1. Decision Trees for Classification

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 $f : \mathcal{X} \mapsto \mathcal{Y}$.

Supervised learning algorithms automatically learn a *classifier* $g : \mathcal{X} \mapsto \mathcal{Y}$ 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

---

[1]We make our analyzer available for download at `https://github.com/FedericoMarcuzzi/resilience-verification`
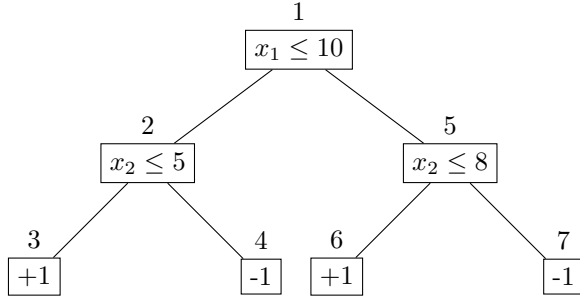
Figure 1: Example of decision tree

as possible. The performance of classifiers is estimated on a *test set* of correctly labeled instances $\mathcal{D}_{test} = \{(\vec{z}_i, f(\vec{z}_i))\}_i$, normally disjoint from the training set. For example, the *accuracy* measure a counts the percentage of correct predictions out of all the predictions performed on the test set:

$$a(g, \mathcal{D}_{test}) = \frac{|\{(\vec{z}_i, f(\vec{z}_i)) \in \mathcal{D}_{test} \mid g(\vec{z}_i) = f(\vec{z}_i)\}|}{|\mathcal{D}_{test}|}$$

Classifiers can take different shapes, drawn from different sets of hypotheses: in this work, we focus on traditional *binary decision trees*, which are the most common and popular version of such models.

Decision trees can be inductively defined as follows: a decision tree t is either a leaf $\lambda(y)$ for some label $y \in \mathcal{Y}$ or an internal node $\sigma(f, v, t_l, t_r)$, where $f \in \{1, \dots, d\}$ identifies a feature, $v \in \mathbb{R}$ is a threshold for the feature, and t_l, t_r are decision trees (left and right respectively). At test time, the instance $\vec{x}$ traverses the tree t until it reaches a leaf $\lambda(y)$, which returns the prediction y, denoted by $t(\vec{x}) = y$. Specifically, 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. Figure 1 represents an example decision tree of depth 2, which assigns the label $+1$ to the instance $\langle 12, 7 \rangle$ and the label -1 to the instance $\langle 8, 6 \rangle$.

Decision trees are normally combined into an *ensemble* $T = \{t_1, \dots, t_n\}$: in this case, the ensemble prediction $T(\vec{x})$ is computed by combining together the individual tree predictions $t_i(\vec{x})$, e.g., by performing majority voting on the individually predicted classes.

2.2. Stability and Robustness

Stability and robustness are classic definitions considered in prior literature on adversarial ML [4]. They are used to reason about the security of classifiers against *evasion attacks*, i.e., malicious manipulations of instances at test time aimed at forcing mispredictions. For example, an evasion attack might slightly modify malware to make it look like benign software to the classifier. We write $A(\vec{x})$ to represent the set of all the adversarial manipulations of the instance $\vec{x}$, corresponding to the possible evasion attack attempts against $\vec{x}$. Stability requires the classifier to make the same prediction for all the possible evasion attack attempts.

Definition 1 (Stability [4])**.** The classifier g is *stable* on the instance $\vec{x}$ if and only if, for every adversarial manipulation $\vec{z} \in A(\vec{x})$, we have $g(\vec{z}) = g(\vec{x})$.

Stability is useful for the security certification of classifiers: if the classifier g is stable on the instance $\vec{x}$, no adversarial manipulation $\vec{z} \in A(\vec{x})$ can be assigned a label different from the classifier prediction $g(\vec{x})$, hence no evasion attack is possible. However, stability does not capture whether a classifier is useful in practice: for example, a trivial classifier which always predicts a constant class is stable on all instances. Hence, the actual property of interest for classifiers deployed in adversarial settings is robustness, which additionally requires the classifier to perform correct predictions.

Definition 2 (Robustness [4])**.** The classifier g is *robust* on the instance $\vec{x}$ if and only if $g(\vec{x}) = f(\vec{x})$ and g is stable on $\vec{x}$.
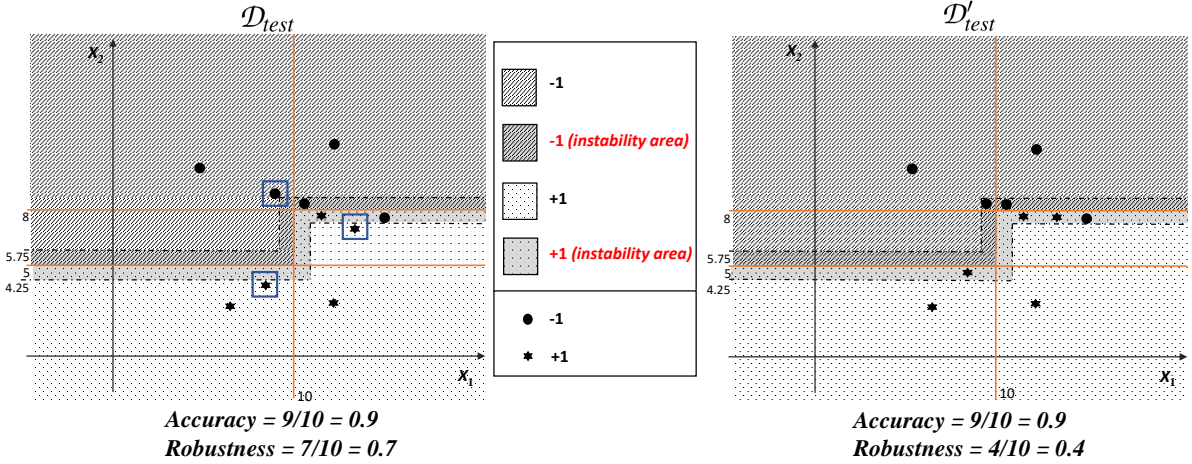
Figure 2: Robustness is not robust: slightly different test sets lead to very different values of robustness

Like accuracy, robustness is also normally quantified over a test set $\mathcal{D}_{test} = \{(\vec{z}_i, f(\vec{z}_i))\}_i$. Formally, we define the robustness measure r as follows:

$$r(g, \mathcal{D}_{test}) = \frac{|\{(\vec{z}_i, f(\vec{z}_i)) \in \mathcal{D}_{test} \mid g \text{ is robust on } \vec{z}_i\}|}{|\mathcal{D}_{test}|}$$

Note that robustness represents a lower bound to accuracy, i.e., $r(g, \mathcal{D}_{test}) \leq a(g, \mathcal{D}_{test})$ for every g and $\mathcal{D}_{test}$.

3. Resilience

We now discuss important shortcomings in the traditional robustness measure and we propose a generalization of robustness, called *resilience*, which is designed to mitigate those. We then explain how resilience can be verified in practice and we further elaborate on its design by discussing its connections with a recent definition of global robustness [6].

3.1. Robustness vs. Resilience

A key problem of robustness is its strong *data-dependence*, i.e., robustness is quantified on a specific test set $\mathcal{D}_{test}$. Hence, it is possible that even tiny differences between two test sets $\mathcal{D}_{test}, \mathcal{D}'_{test}$ might lead to quite different values of robustness. This might give a false sense of security, because a classifier having 0.7 robustness on $\mathcal{D}_{test}$ might only have 0.4 robustness on $\mathcal{D}'_{test}$, although both $\mathcal{D}_{test}$ and $\mathcal{D}'_{test}$ are representative of the same data distribution of the test instances. We show this problem of robustness in Figure 2 for the same tree classifier of Figure 1, in a two-dimensional space. We assume here an L_1-norm attacker such that $A(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid \|\vec{z} - \vec{x}\|_1 \leq 0.75\}$, leading to the highlighted *instability areas* around the decision boundaries of the tree. The figure shows that the same classifier provides very different robustness measures on the two close test sets $\mathcal{D}_{test}$ and $\mathcal{D}'_{test}$, because more instances of $\mathcal{D}'_{test}$ fall in the instability area. Hence, the adoption of $\mathcal{D}_{test}$ over $\mathcal{D}'_{test}$ for robustness computation might give a false sense of security.

To mitigate this problem of robustness, we propose *resilience*, a new security measure that explicitly assumes that test instances are sampled from a given data distribution, hence each instance $\vec{x}$ is just a possible sample drawn from a set of neighbours $N(\vec{x})$. For example, $N(\vec{x})$ may contain all the instances which are within a maximum L_∞-distance from $\vec{x}$, as we assume in our experiments. In the example of Figure 2, we define $N(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid \|\vec{z} - \vec{x}\|_\infty \leq 0.50\}$ and the neighborhood of $\vec{x}$ is thus graphically represented by a small box around $\vec{x}$. Indeed, the figure shows these boxes only for the instances of $\mathcal{D}_{test}$

4

whose neighborhood $N(\vec{x})$ intersects the instability areas of the tree and, therefore, other instances in their neighborhood might suffer from evasion attacks.

Resilience avoids the shortcomings of robustness illustrated in Figure 2, as it generalizes the idea of robustness to all the test sets which could have been sampled within neighborhoods of the original test set. Formally, resilience requires the classifier to be robust on test instances, while remaining stable on their neighborhoods, for which the correct class labels are unknown.

Definition 3 (Resilience). The classifier g is *resilient* on the instance $\vec{x}$ if and only if g is robust on $\vec{x}$ and g is stable on all the instances $\vec{z} \in N(\vec{x})$.

Back to our example, the resilience of both the test sets of Figure 2 turns out to be 0.4, i.e., the measured robustness of $\mathcal{D}'_{test}$, the unlucky test set shown in the right part of the figure.

Note that resilience generalizes beyond the test set by means of the $N(\vec{x})$ component, which extends the stability guarantees of robustness to an uncountable set of neighbours not included in the test set. Still, similarly to robustness, we can quantify resilience on a test set of correctly labelled instances like in traditional ML pipelines by defining a resilience measure R as follows:

$$R(g, \mathcal{D}_{test}) = \frac{|\{(\vec{z}_i, f(\vec{z}_i)) \in \mathcal{D}_{test} \mid g \text{ is resilient on } \vec{z}\}|}{|\mathcal{D}_{test}|}$$

Observe that resilience provides a lower bound to robustness as claimed, i.e., $R(g, \mathcal{D}_{test}) \leq r(g, \mathcal{D}_{test})$ for every g and $\mathcal{D}_{test}$.

Finally, we acknowledge that resilience is still a data-dependent property, however the dependence from the test set is much weaker for resilience than for robustness, thanks to the introduction of the $N(\vec{x})$ component. Indeed, data dependence is not necessarily a problem per se, because a high quality test set is required to empirically assess the performance of ML models anyway, however an excessive dependence on a given test set might provide a false sense of security against evasion attacks. Our claims of the superiority of resilience over robustness are not just theoretical, since they are confirmed by the experimental evaluation in Section 5.

3.2. Resilience Verification

Traditional robustness verification approaches cannot be readily applied to prove resilience. In particular, note that robustness verification takes as input an instance $\vec{x}$ and attempts to assess its stability, while resilience verification requires proving stability for an uncountable set of instances $N(\vec{x})$.

Nevertheless, it is possible to estimate resilience by combining robustness verification with a *data-independent* stability analysis. In particular, assume one has a technique to identify the set $X_S = \{\vec{x} \in \mathcal{X} \mid g \text{ is stable on } \vec{x}\}$, then g is resilient on $\vec{x}$ if and only if g is robust on $\vec{x}$ and $N(\vec{x}) \subseteq X_S$. Note that X_S in Figure 2 corresponds to the whole area except the instability ones. Hence, resilience verification reduces to robustness verification with the additional condition $N(\vec{x}) \subseteq X_S$, i.e., we also have to check that $N(\vec{x})$ does not intersect the instability area. This allows one to take advantage of existing robustness verifiers also to quantify resilience, provided that the stable part of the feature space X_S has been computed. Note that computing X_S is not trivial, in particular for tree ensembles, because a given instance traverses the ensemble reaching a set of leaves, and any of such reachable set of leaves corresponds to a sub-space which should be evaluated for inclusion in X_S. Clearly, the number of these sub-spaces grows exponentially with the number of trees in the ensemble. In order to compute X_S, and therefore compute resilience, we resort to an approximated approach: computing a tractable under-approximation of X_S for decision trees and decision tree ensembles is a key contribution of the present paper.

In particular, the static analysis in Section 4 allows one to compute a set $X'_S \subseteq X_S$, i.e., a sound under-approximation of the portion of the feature space where a tree-based classifier is stable. The proposed computation of X'_S depends on the classifier alone and not on the test data at hand. In fact, we can exploit the generality of X'_S to efficiently compute lower bounds for the robustness and resilience measures as follows.

Given that if $\vec{x} \in X'_S$ and $g(\vec{x}) = f(\vec{x})$, then g is robust on $\vec{x}$, we can define a lower bound $\hat{r}$ on robustness as:

$$\hat{r}(g, \mathcal{D}_{test}) = \frac{|\{(\vec{z}_i, f(\vec{z}_i)) \in \mathcal{D}_{test} \mid \vec{z} \in X'_S \wedge g(\vec{z}) = f(\vec{z})\}|}{|\mathcal{D}_{test}|}$$

Similarly, given that if g is robust on $\vec{x}$ and $N(\vec{x}) \subseteq X'_S$, then g is also resilient on $\vec{x}$, we can introduce a lower bound $\hat{R}$ on resilience as:

$$\hat{R}(g, \mathcal{D}_{test}) = \frac{|\{(\vec{z}_i, f(\vec{z}_i)) \in \mathcal{D}_{test} \mid N(\vec{z}) \subseteq X'_S \wedge g \text{ robust on } \vec{z}\}|}{|\mathcal{D}_{test}|}$$

Note that the pre-computation of X'_S makes robustness verification straightforward for all the instances falling in X'_S, and that the computation of the resilience estimate $\hat{R}$ can potentially exploit any robustness verification method, which allows one to leverage existing work in the area. Moreover, we may use the accuracy of $\hat{r}$ with respect to the true robustness r (established using an existing robustness verifier) as a proxy for the accuracy of X'_S, which allows us to assess the quality of the under-approximation $\hat{R}$ and advocate its adoption to mitigate the false sense of security provided by r.

3.3. Resilience vs. Global Robustness

Recent work proposed a technique to train *globally robust* neural networks, which provide robustness guarantees for all the possible inputs, rather than just for the inputs in the test set [6]. The idea of globally robust neural networks, generalizable to arbitrary classifiers, is that the set of labels $\mathcal{Y}$ is extended with a special class $\perp$, used to denote that no reliable prediction is possible, because the instance is too close to the decision boundary of the classifier and thus potentially susceptible to evasion attacks. Global robustness requires that any two instances which are close enough to each other are either assigned the same prediction, or at least one of them is flagged as $\perp$. This property is intuitive and desirable, however, contrary to resilience, it cannot be used to verify the security of existing classifiers which have not been trained to return the $\perp$ label.

Rather, note that the proposed approach to resilience verification based on a data-independent stability analysis can be readily applied to transform any classifier into a globally robust classifier. In particular, given any classifier g and a subset of the feature space $X'_S \subseteq \mathcal{X}$ where g is stable, one can define a globally robust classifier g' as follows:

$$g'(\vec{x}) = \begin{cases} g(\vec{x}) & \text{if } \vec{x} \in X'_S \\ \perp & \text{otherwise} \end{cases}$$

Observe that the robustness estimate $\hat{r}$ previously defined provides a (local) robustness measure of the globally robust classifier g' obtained through the previous construction.

4. Data-Independent Stability Analysis

We present here a data-independent stability analysis for decision trees and decision tree ensembles, which allows one to compute the two measures $\hat{r}$ and $\hat{R}$ defined in Section 3.2, thus providing conservative estimates of robustness and (most importantly) resilience. The analysis is proved sound, i.e., we show that the portions of the feature space which are marked as stable by the analysis may only contain instances where the classifier is indeed stable. Proofs are given in Appendix Appendix A.

4.1. Preliminaries

Our analysis leverages *intervals* of real numbers. Given $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$ with $a \leq b$, we use standard notation for intervals, using parentheses to represent open bounds and brackets to represent closed bounds. This leads to four possible types of intervals: (a, b), $[a, b)$, $(a, b]$ and $[a, b]$. We use I, J to range over intervals. A *hyper-rectangle* $H \subseteq \mathcal{X}$ is represented as a vector of intervals $\langle I_1, \ldots, I_d \rangle$ over $\mathbb{R}$.

Given a decision tree t, we define its possible predictions over the hyper-rectangle H as $t(H) = \{y \in \mathcal{Y} \mid \exists \vec{x} \in H : t(\vec{x}) = y\}$. Note that computing $t(H)$ is straightforward by means of a recursive tree traversal.

Specifically, if $t = \lambda(y)$, then $t(H) = \{y\}$. If instead $t = \sigma(f, v, t_l, t_r)$, we define $t(H)$ with $H = \langle I_1, \ldots, I_d \rangle$ as follows:

$$t(H) = \begin{cases} t_l(H) & \text{if } I_f \cap (v, +\infty) = \emptyset \\ t_r(H) & \text{if } I_f \cap (-\infty, v] = \emptyset \\ t_l(H) \cup t_r(H) & \text{otherwise} \end{cases}$$

Finally, we extend predictions over hyper-rectangles from trees to tree ensembles, noted $T(H)$. The actual definition of $T(H)$ depends on the approach used by T to aggregate the individual tree predictions to produce the ensemble prediction. For example, in the case of majority voting, we may let:

$$T(H) = \begin{cases} \{y\} & \text{if } |\{t_i \in T \mid t_i(H) = \{y\}\}| > |T|/2 \\ \mathcal{Y} & \text{otherwise} \end{cases}$$

For soundness, we require $\{y \in \mathcal{Y} \mid \exists \vec{x} \in H : T(\vec{x}) = y\} \subseteq T(H)$, i.e., the set of the predictions $T(H)$ includes all the predictions that may be assigned to an instance in H. Proving that this requirement holds for the definition above is straightforward.

In the following, given two intervals, we define their sum $I + J$ as the interval whose lower bound is the sum of the lower bounds and whose upper bound is the sum of the upper bounds; we require the bounds to be closed if and only if both the added bounds are closed. For example, $[1, 3] + (4, 6] = (5, 9]$. Moreover, given two hyper-rectangles, we define their sum $H + H'$ as the pointwise sum of their components (intervals).

4.2. Threat Model

We assume each feature $f \in \{1, \ldots, d\}$ can be manipulated by paying a *cost* $c_f \in \mathbb{N}$, which allows the attacker to add a perturbation $\delta \in \mathbb{R}$ arbitrarily drawn from an interval I_f^{atk}. We characterize the attacker's power in terms of a *budget* b, which determines the maximum aggregate cost that the attacker can pay to manipulate features. We encode robust features which cannot be manipulated by setting $[0, 0]$ as their perturbation interval.

Definition 4 (Adversarial Manipulations). Given an instance $\vec{x} \in \mathcal{X}$, we define the set of the *adversarial manipulations*, noted $A(\vec{x})$, as the set of the instances $\vec{z}$ such that there exists a set of features $F \subseteq \{1, \ldots, d\}$ such that:

1. For all $f \in F$, we have $z_f = x_f + \delta$ for some $\delta \in I_f^{atk}$.
2. For all $f \notin F$, we have $z_f = x_f$.
3. We have $\sum_{f \in F} c_f \leq b$.

This threat model is inspired by traditional distance-based attackers from the adversarial ML literature and it is expressive enough to model a wide range of attacks, including those based on the L_0-norm and the L_∞-norm which are traditionally used in prior work [8, 9]. In particular, note that:

- L_0-norm attackers can be modeled by assuming that each feature can be perturbed in the interval $[-\infty, +\infty]$ by paying cost 1 and by setting the attacker's budget to the maximum L_0-distance assumed for the attack.

- L_∞-norm attackers can be modeled by assuming that each feature can be perturbed in the interval $[-\delta, +\delta]$ by paying cost 1, where δ is the maximum L_∞-distance assumed for the attack, and by setting the attacker's budget to the total number of features.

By using this threat model, we are able to design a relatively general analysis technique that readily applies to at least these two popular classes of attackers from the literature. These attackers are popular, because they are simple mathematical models of two representative classes of threats: strong corruptions of a small number of features (L_0) and weak corruptions of all the features (L_∞).

4.3. Stability Analysis for Decision Trees

Our analysis operates by annotating each node of the decision tree with a set of *symbolic attacks*, which represent a set of instances along with their *relevant* adversarial manipulations. A first insight of our analysis is that most adversarial manipulations are not relevant for the stability analysis of decision trees, because such classifiers operate by means of thresholds, hence only attacks which allow for traversing some threshold might lead to instability [10]. Formally, a symbolic attack s has the shape $\langle I_1^{pre}, \ldots, I_d^{pre} \rangle \triangleright \langle I_1^{post}, \ldots, I_d^{post} \rangle_k$, where each I_i^{pre}, I_j^{post} is an interval on $\mathbb{R}$ and $k \in \mathbb{N}$. Intuitively, s identifies the set of instances located within the hyper-rectangle $\langle I_1^{pre}, \ldots, I_d^{pre} \rangle$, called the *pre-image* of the symbolic attack, along with their adversarial manipulations located within the hyper-rectangle $\langle I_1^{post}, \ldots, I_d^{post} \rangle$, called the *post-image* of the symbolic attack; the *cost* of such adversarial manipulations is bounded above by k. We make use of symbolic attacks to identify which nodes of the decision tree can be traversed by a set of instances as the result of adversarial manipulations against them; we require that when $k = 0$, the pre-image and the post-image coincide, i.e., the symbolic attack captures the case where no adversarial manipulation has taken place. We write $s.pre$, $s.post$ and $s.cost$ to project the pre-image, the post-image and the cost of s respectively.

Before presenting the formal details, we present the analysis at work on the decision tree of Figure 1 built on a feature space with two features. We assume an attacker who can manipulate at most one feature by adding a perturbation $\delta \in [-1, 1]$. Formally, this is represented by having $I_1^{atk} = I_2^{atk} = [-1, 1]$, $c_1 = c_2 = 1$ and $b = 1$. The result of the analysis in terms of node annotations are shown in Table 1. Node 1 is the root of the tree, hence the analysis cannot conclude anything about instances traversing the node, i.e., the symbolic attack in the node annotation models that all instances in the feature space traverse the root, no matter what the attacker does. Node 2, instead, is more interesting, because the analysis captures two cases via two symbolic attacks: an instance $\vec{x}$ might traverse the node either because $x_1 \leq 10$ and the attacker does nothing, or because $x_1 \in (10, 11]$ is adversarially manipulated into the interval $(9, 10]$. Note that, in the second case, the symbolic attack is assigned cost 1, which allows us to track that no further manipulation is possible because the budget has been exhausted. Even more interesting is the case of node 3, where we have three possibilities. In particular, an instance $\vec{x}$ might traverse the node in the following cases: (*i*) $x_1 \leq 10$ and $x_2 \leq 5$, hence no adversarial manipulation is needed, (*ii*) $x_1 \leq 10$ and $x_2 \in (5, 6]$ is manipulated into $(4, 5]$, or (*iii*) $x_1 \in (10, 11]$ is manipulated into $(9, 10]$ and $x_2 \leq 5$. We do not have a case where both features are manipulated, because this would exceed the attacker's budget. A similar reasoning applies to the other nodes in the tree. Once the tree has been annotated, it is possible to check stability by inspecting the annotations in its leaves: we discuss this aspect of the analysis later in the section.

Algorithm 1 describes the annotation function for decision trees. We assume each node of the tree is enriched with an attribute *sym*, used to store a set of symbolic attacks. The call ANNOTATE(t, S) annotates the root of t with the set of symbolic attacks S passed as a parameter (line 2), then uses S and the threshold information in the root to generate the annotations for the roots of the left and right sub-trees (lines 6-8); finally, the process goes down recursively (lines 9-10). When the annotation function is initially invoked on the root of the decision tree to analyze, we set $S = \{\langle (-\infty, +\infty)^d \rangle \triangleright \langle (-\infty, +\infty)^d \rangle_0\}$ as explained in the example.

The key part of the node annotation logic is implemented by the auxiliary functions REFINELEFT and REFINERIGHT, defined in Algorithm 2. Given a symbolic attack s, a feature f and the associated threshold v from an internal node of the decision tree, the call REFINELEFT(s, f, v) uses s to generate a new set of symbolic attacks S for the root of the left sub-tree (initially empty). Lines 7-13 account for the case where some instances in the post-image of s already fall in the left sub-tree, i.e., the attacker does not need to spend budget to manipulate the feature f so as to push some instances in the pre-image into the left sub-tree. In this case, S is extended with a refined variant of s, where we track that the feature f must belong to the interval $(-\infty, v]$ for the instances in the post-image (line 8). Also the pre-image of s is refined in the left sub-tree: if f was not attacked, we know that the feature f must belong to the interval $(-\infty, v]$ for the instances in the pre-image as well (lines 9-10); otherwise, we still know that the attack could not push instances beyond the maximum negative perturbation $\delta_l < 0$, hence the feature f must belong to the interval $(-\infty, v - \min(0, \delta_l)]$ for the instances in the pre-image (lines 11-12). Lines 14-17, instead, cover the case where some instances in the pre-image are close enough to the threshold v to be pushed into the

Table 1: Analysis results for the decision tree of Figure 1

1	$\langle(-\infty,+\infty),(-\infty,+\infty)\rangle \rhd \langle(-\infty,+\infty),(-\infty,+\infty)\rangle_0$
2	$\langle(-\infty,10],(-\infty,+\infty)\rangle \rhd \langle(-\infty,10],(-\infty,+\infty)\rangle_0$
	$\langle(10,11],(-\infty,+\infty)\rangle \rhd \langle(9,10],(-\infty,+\infty)\rangle_1$
3	$\langle(-\infty,10],(-\infty,5]\rangle \rhd \langle(-\infty,10],(-\infty,5]\rangle_0$
	$\langle(-\infty,10],(5,6]\rangle \rhd \langle(-\infty,10],(4,5]\rangle_1$
	$\langle(10,11],(-\infty,5]\rangle \rhd \langle(9,10],(-\infty,5]\rangle_1$
4	$\langle(-\infty,10],(5,+\infty)\rangle \rhd \langle(-\infty,10],(5,+\infty)\rangle_0$
	$\langle(-\infty,10],(4,5]\rangle \rhd \langle(-\infty,10],(5,6]\rangle_1$
	$\langle(10,11],(5,+\infty)\rangle \rhd \langle(9,10],(5,+\infty)\rangle_1$
5	$\langle(10,+\infty),(-\infty,+\infty)\rangle \rhd \langle(10,+\infty),(-\infty,+\infty)\rangle_0$
	$\langle(9,10],(-\infty,+\infty)\rangle \rhd \langle(10,11],(-\infty,+\infty)\rangle_1$
6	$\langle(10,+\infty),(-\infty,8]\rangle \rhd \langle(10,+\infty),(-\infty,8]\rangle_0$
	$\langle(10,+\infty),(8,9]\rangle \rhd \langle(10,+\infty),(7,8]\rangle_1$
	$\langle(9,10],(-\infty,8]\rangle \rhd \langle(10,11],(-\infty,8]\rangle_1$
7	$\langle(10,+\infty),(8,+\infty)\rangle \rhd \langle(10,+\infty),(8,+\infty)\rangle_0$
	$\langle(10,+\infty),(7,8]\rangle \rhd \langle(10,+\infty),(8,9]\rangle_1$
	$\langle(9,10],(8,+\infty)\rangle \rhd \langle(10,11],(8,+\infty)\rangle_1$

Algorithm 1 Decision tree annotation

1: **function** ANNOTATE(t, S)
2:      $t.sym \leftarrow S$
3:      **if** $t = \sigma(f, v, t_l, t_r)$ **then**
4:          $S_l \leftarrow \emptyset$
5:          $S_r \leftarrow \emptyset$
6:          **for** $s \in S$ **do**
7:              $S_l \leftarrow S_l \cup \text{REFINELEFT}(s, f, v)$
8:              $S_r \leftarrow S_r \cup \text{REFINERIGHT}(s, f, v)$
9:          $t_l \leftarrow \text{ANNOTATE}(t_l, S_l)$
10:         $t_r \leftarrow \text{ANNOTATE}(t_r, S_r)$
11:      **return** t

---

left sub-tree as the result of adversarial manipulations. In this case, provided that the attacker still has enough budget to spend, S is extended with a refined variant of s, where we update both the post-image and the pre-image to reflect their proximity to the threshold v. More precisely, given the maximum negative perturbation $\delta_l < 0$, we track that the feature f must belong to the interval $(v + \delta_l, v]$ for the instances in the post-image and to the interval $(v, v - \delta_l]$ for the instances in the pre-image, otherwise crossing the threshold would not be possible. The REFINERIGHT function performs an analogous reasoning for the right sub-tree, hence we omit a detailed explanation.

The stability analysis for decision trees is finally shown in Algorithm 3. The call ANALYZE(t) leverages the results of the tree annotation function to return a set of symbolic attacks U, identifying the portions of the feature space where the decision tree t may be unstable. The function operates by looking for two leaves with different class predictions such that: (i) the first leaf contains a symbolic attack s of cost 0, i.e., no adversarial manipulation was performed on the pre-image of s, (ii) the second leaf contains a symbolic attack s' of cost greater than 0, i.e., the attacker manipulated the pre-image of s', and (iii) the pre-images of the two symbolic attacks s, s' partially overlap, i.e., there exist some instances which might fall into a leaf with a different class than the original prediction due to adversarial manipulations (lines 4-9). In this case, the intersection of the pre-images identifies a portion of the feature space where the tree may be unstable and the post-image of s' is refined to capture that the adversarial manipulations cannot push instances

Algorithm 2 Refinement for the left and right sub-trees

1: **function** REFINELEFT(s, f, v)
2:    $S \leftarrow \emptyset$
3:    $\langle I_1^{pre}, \ldots, I_d^{pre} \rangle \leftarrow s.pre$
4:    $\langle I_1^{post}, \ldots, I_d^{post} \rangle \leftarrow s.post$
5:    $k \leftarrow s.cost$
6:    $\langle \delta_l, \delta_r \rangle \leftarrow I_f^{atk}$
7:    **if** $I_f^{post} \cap (-\infty, v] \neq \emptyset$ **then**
8:        $J_f^{post} \leftarrow I_f^{post} \cap (-\infty, v]$
9:        **if** $I_f^{pre} = I_f^{post}$ **then**
10:           $J_f^{pre} \leftarrow I_f^{pre} \cap (-\infty, v]$
11:       **else**
12:           $J_f^{pre} \leftarrow I_f^{pre} \cap (-\infty, v - \min(0, \delta_l)]$
13:       $S \leftarrow S \cup \{\langle I_1^{pre}, \ldots, I_{f-1}^{pre}, J_f^{pre}, I_{f+1}^{pre}, \ldots I_d^{pre} \rangle \triangleright \langle I_1^{post}, \ldots, I_{f-1}^{post}, J_f^{post}, I_{f+1}^{post}, \ldots, I_d^{post} \rangle_k\}$
14:   **if** $I_f^{pre} = I_f^{post} \wedge \delta_l < 0 \wedge I_f^{pre} \cap (v, v - \delta_l] \neq \emptyset \wedge k + c_f \leq b$ **then**
15:       $J_f^{post} \leftarrow I_f^{post} \cap (v + \delta_l, v]$
16:       $J_f^{pre} \leftarrow I_f^{pre} \cap (v, v - \delta_l]$
17:       $S \leftarrow S \cup \{\langle I_1^{pre}, \ldots, I_{f-1}^{pre}, J_f^{pre}, I_{f+1}^{pre}, \ldots I_d^{pre} \rangle \triangleright \langle I_1^{post}, \ldots, I_{f-1}^{post}, J_f^{post}, I_{f+1}^{post}, \ldots, I_d^{post} \rangle_{k+c_f}\}$
18:   **return** S
19:
20: **function** REFINERIGHT(s, f, v)
21:    $S \leftarrow \emptyset$
22:    $\langle I_1^{pre}, \ldots, I_d^{pre} \rangle \leftarrow s.pre$
23:    $\langle I_1^{post}, \ldots, I_d^{post} \rangle \leftarrow s.post$
24:    $k \leftarrow s.cost$
25:    $\langle \delta_l, \delta_r \rangle \leftarrow I_f^{atk}$
26:    **if** $I_f^{post} \cap (v, +\infty) \neq \emptyset$ **then**
27:        $J_f^{post} \leftarrow I_f^{post} \cap (v, +\infty)$
28:        **if** $I_f^{pre} = I_f^{post}$ **then**
29:            $J_f^{pre} \leftarrow I_f^{pre} \cap (v, +\infty)$
30:        **else**
31:            $J_f^{pre} \leftarrow I_f^{pre} \cap (v - \max(0, \delta_r), +\infty)$
32:        $S \leftarrow S \cup \{\langle I_1^{pre}, \ldots, I_{f-1}^{pre}, J_f^{pre}, I_{f+1}^{pre}, \ldots I_d^{pre} \rangle \triangleright \langle I_1^{post}, \ldots, I_{f-1}^{post}, J_f^{post}, I_{f+1}^{post}, \ldots, I_d^{post} \rangle_k\}$
33:    **if** $I_f^{pre} = I_f^{post} \wedge \delta_r > 0 \wedge I_f^{pre} \cap (v - \delta_r, v] \neq \emptyset \wedge k + c_f \leq b$ **then**
34:        $J_f^{post} \leftarrow I_f^{post} \cap (v, v + \delta_r]$
35:        $J_f^{pre} \leftarrow I_f^{pre} \cap (v - \delta_r, v]$
36:        $S \leftarrow S \cup \{\langle I_1^{pre}, \ldots, I_{f-1}^{pre}, J_f^{pre}, I_{f+1}^{pre}, \ldots I_d^{pre} \rangle \triangleright \langle I_1^{post}, \ldots, I_{f-1}^{post}, J_f^{post}, I_{f+1}^{post}, \ldots, I_d^{post} \rangle_{k+c_f}\}$
37:    **return** S

Algorithm 3 Stability analysis for decision trees

1: **function** ANALYZE(t)
2:     $t \leftarrow$ ANNOTATE($t, \{\langle(-\infty, +\infty)^d\rangle \rhd \langle(-\infty, +\infty)^d\rangle_0\}$)
3:     $U \leftarrow \emptyset$
4:     **for** $\lambda(y) \in t$ **do**
5:         **for** $s \in \{\hat{s} \in \lambda(y).sym \mid \hat{s}.cost = 0\}$ **do**
6:             **for** $\lambda'(y') \in t$ **do**
7:                 **if** $y' \neq y$ **then**
8:                     **for** $s' \in \{\hat{s} \in \lambda'(y').sym \mid \hat{s}.cost > 0\}$ **do**
9:                         **if** $s.pre \cap s'.pre \neq \emptyset$ **then**
10:                            $s''.pre \leftarrow s.pre \cap s'.pre$
11:                            $s''.post \leftarrow s'.post \cap (s''.pre + \langle I_1^{atk}, \ldots, I_d^{atk}\rangle)$
12:                            $s''.cost \leftarrow s'.cost$
13:                            $U \leftarrow U \cup \{s''\}$
14:     **return** U

---

beyond the maximum allowed manipulation of the intersection of the two pre-images (lines 10-13). This is a conservative approximation, which accounts for all the possible adversarial manipulations.

To exemplify the output of the stability analysis, consider the node annotations in Table 1. The stability analysis returns the following symbolic attacks:

- For leaf 3: $\langle(-\infty, 10], (4, 5]\rangle \rhd \langle(-\infty, 10], (5, 6]\rangle_1$

- For leaf 4: $\langle(-\infty, 10], (5, 6]\rangle \rhd \langle(-\infty, 10], (4, 5]\rangle_1$ and $\langle(9, 10], (5, 8]\rangle \rhd \langle(10, 11], (4, 8]\rangle_1$

- For leaf 6: $\langle(10, +\infty), (7, 8]\rangle \rhd \langle(10, +\infty), (8, 9]\rangle_1$ and $\langle(10, 11], (5, 8]\rangle \rhd \langle(9, 10], (5, 9]\rangle_1$

- For leaf 7: $\langle(10, +\infty), (8, 9]\rangle \rhd \langle(10, +\infty), (7, 8]\rangle_1$

The pre-images of these symbolic attacks identify the portions of the feature space where the decision tree is unstable. It is interesting to observe that leaves 4 and 6 contribute two portions of the feature space where the tree may be unstable, while leaves 3 and 7 only contribute one. The reason for this is that leaves 4 and 6 partially overlap on the values allowed for the second feature, i.e., the interval $(5, 8]$. This means that it is possible to jump from leaf 4 to leaf 6 (and vice-versa) as the result of an attack targeting just the first feature, provided that the second feature falls into $(5, 8]$. Conversely, leaves 3 and 7 have no overlap on any of the two features, hence an attack which manipulates just one feature cannot induce a jump between these two leaves. As a final comment, notice that the post-images are not needed at this stage of the analysis: we just collect them because they support the analysis of tree ensembles in the next section.

The soundness theorem for our analysis is given below. The theorem states that all the instances $\vec{x}$ where t is unstable must fall in the pre-image of a symbolic attack contained in the set U returned by the call ANALYZE(t). In other words, the union of the pre-images of the symbolic attacks in U can only over-approximate the portion of the feature space where t is unstable, i.e., t must be stable on all the instances located outside such area. This allows us to compute the set X'_S where t is stable (cf. Section 3.2) by subtracting the union of the pre-images of U from the full feature space $\mathcal{X}$.

Theorem 1 (Soundness of Tree Analysis). *The call* ANALYZE(t) *returns a set of symbolic attacks U such that, for every instance $\vec{x} \in \mathcal{X}$ and every adversarial manipulation $\vec{z} \in A(\vec{x})$ such that $t(\vec{z}) \neq t(\vec{x})$, there exists $s \in U$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$.*

We conjecture that the analysis is not only sound, but also complete, i.e., for every $s \in U$ there exist $\vec{x} \in \mathcal{X}$ and $\vec{z} \in A(\vec{x})$ such that $\vec{x} \in s.pre$, $\vec{z} \in s.post$ and $t(\vec{z}) \neq t(\vec{x})$. However, we do not formally prove this result because it has limited practical value: in particular, single decision trees are very rarely used in practice and have to be combined in an ensemble to provide enough predictive power. Since the

---

Algorithm 4 Stability analysis for tree ensembles

1: **function** ANALYZE(T)
2:     $C \leftarrow \emptyset$
3:     **for** $t_i \in T$ **do**
4:         $C \leftarrow C \cup \text{ANALYZE}(t_i)$
5:     $E \leftarrow \emptyset$
6:     **while** stopping condition is not met **do**
7:         **for** $s \in C$ **do**
8:             **if** $\exists y : T(s.pre) = T(s.post) = \{y\}$ **then**
9:                 $C \leftarrow C \setminus \{s\}$
10:            **else**
11:                **if** $T(s.pre) \cap T(s.post) \neq \emptyset$ **then**
12:                    $C \leftarrow (C \setminus \{s\}) \cup \text{SPLIT}(s)$
13:                **else**
14:                    $C \leftarrow C \setminus \{s\}$
15:                    $E \leftarrow E \cup \{s\}$
16:        **return** $C \cup E$

---

ensemble analysis in the next section is sound but not complete, the conjectured completeness result would not generalize to practical cases.

4.4. Stability Analysis for Tree Ensembles

We now discuss how the stability analysis for decision trees can be generalized to tree ensembles by means of an iterative algorithm (Algorithm 4). The algorithm operates by refining a set of candidates C where the ensemble T may be unstable, initially set to the union of the symbolic attacks computed for the individual trees $t_i \in T$ (line 2-4). The algorithm also keeps track of an initially empty set of symbolic attacks E where the analysis has ended because no further refinement of them is possible (line 5).

Each iteration of the algorithm inspects all the candidates $s \in C$, distinguishing three cases (lines 6-15). If T predicts the same label y over both the pre-image and the post-image of s, then T is stable on that portion of the feature space and s is removed from C (lines 8-9). Otherwise, the algorithm checks whether the predictions performed by T over the pre-image and the post-image of s share some common elements. If this is the case, then T may be stable on a subset of the pre-image of s, yet this cannot be concluded at the current iteration; hence, s is refined by splitting it into a set of smaller symbolic attacks (lines 11-12). Any splitting criterion would work, as long as it satisfies the soundness condition defined in the theorem below. If instead the predictions performed by T over the pre-image and the post-image of s are disjoint, there is no way of proving that T is stable even on a subset of the pre-image of s, hence s is moved from C to E to avoid further refinements (lines 14-15).

The algorithm may implement an arbitrary stopping condition, e.g., C is empty or a maximum number of iterations has been performed, as we do in our implementation. Similarly to the stability analysis for decision trees, the algorithm eventually returns a set of symbolic attacks $C \cup E$, whose union of the pre-images over-approximates the portion of the feature space where T is unstable. The soundness of the analysis is formalized by the following theorem, which is the natural generalization to ensembles of Theorem 1.

Theorem 2 (Soundness of Tree Ensemble Analysis). *Assume the following hypotheses:*

- *$T(H)$ is sound, i.e., it satisfies the following: $\{y \in \mathcal{Y} \mid \exists \vec{x} \in H : T(\vec{x}) = y\} \subseteq T(H)$.*

- *The splitting procedure is sound, i.e., for all symbolic attacks s, if there exist $\vec{x} \in \mathcal{X}$ and $\vec{z} \in A(\vec{x})$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$, then there exists $s' \in \text{SPLIT}(s)$ such that $\vec{x} \in s'.pre$ and $\vec{z} \in s'.post$.*

The call ANALYZE(T) returns a set of symbolic attacks U such that, for every instance $\vec{x} \in \mathcal{X}$ and every adversarial manipulation $\vec{z} \in A(\vec{x})$ such that $T(\vec{z}) \neq T(\vec{x})$, there exists $s \in U$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$.

Note that the ensemble analysis is sound, but not complete for generic reasons, the most important one being that even the traditional robustness verification problem for tree ensembles is already NP-hard [11] and computing all the equivalence classes induced by an ensemble quickly becomes infeasible in practice [12]. We expect a complete analysis to be feasible for restricted settings, e.g., for specific attackers, however we leave the investigation of this problem to future work.

4.5. Implementation

We implemented our stability analysis for decision trees and decision tree ensembles (based on majority voting). The output of the stability analysis is used to compute lower bounds of robustness $\hat{r}$ and resilience $\hat{R}$ for a given model and test set, as discussed in Section 3.2.

We discuss below selected aspects of the implementation, which is a rather direct translation of our pseudo-code. A first point to note is that the set of candidates C is implemented by means of a min priority queue and only the top k symbolic attacks can be split at each loop iteration ($k = 0.05 \cdot |C|$ by default) to mitigate the growth of C. The priority queue is ordered according to the following criterion: each symbolic attack s is first assigned a pair (n_c, n_u), where n_c is a counter used to keep track of how many splits have been performed to produce s and n_u is the number of "undecided" trees t_i such that $|t_i(s.pre)| > 1$; pairs are then ordered according to the standard lexicographic order. In this heuristics, n_c acts as a penalization factor to ensure that the algorithm does not split the same symbolic attacks too many times, but rather tries to process all the symbolic attacks at least once even when the number of iterations is relatively small; symbolic attacks with the same value of n_c are split by prioritizing symbolic attacks with a small number of undecided trees n_u, because they are intuitively easier to certify and should be analyzed earlier.

A second relevant aspect to discuss is the implementation of the splitting function. Given the symbolic attack s, our implementation of $\text{SPLIT}(s)$ operates as follows: it first identifies a feature f and a threshold v such that v falls in the f-th component of $s.pre$; then, if $I_f^{atk} = [\delta_l, \delta_r]$, it splits $s.pre$ in (at most four) hyper-rectangles based on the thresholds $v+\delta_l$, v, $v+\delta_r$. For example given the interval (a, b), if v, $v+\delta_l$ and $v + \delta_r$ are inside (a, b), the $\text{SPLIT}(s)$ function divides the interval in the following four intervals: $(a, v + \delta_l]$, $(v + \delta_l, v]$, $(v, v + \delta_r]$ and $(v + \delta_r, b]$. Finally, it uses these intervals as the pre-images of the new symbolic attacks, computing the corresponding post-images by intersecting $s.post$ with the maximum perturbation applicable to the pre-images. It is easy to show that this implementation enjoys the required soundness condition for the splitting procedure.Finally, we note that Algorithm 4 can be parallelized by partitioning C across different threads and by joining the analysis results at the end. In particular, we first build the priority queue C and then we distribute it across threads using a round robin algorithm, which is useful to ensure that no thread is penalized by the prevalence of symbolic attacks which are expected to be hard to analyze. This scheme ensures a deeper exploration of C when the analysis terminates before convergence after a fixed number of iterations. Our implementation supports a configurable number of threads.

5. Experimental Evaluation

We finally report on the experimental evaluation of our resilience verification technique. We first discuss the setup and the research questions, then we present the key experiments and results.

5.1. Experimental Setup

We evaluate our proposal using four publicly available datasets from LIBSVM Data[2]. This source of datasets has been used in prior work on the robustness verification of tree-based models [9]. We selected instances of class 1 and 11 from the Sensorless dataset, since our analysis works with binary classification trees and forests. Datasets are divided into a training test $\mathcal{D}_{train}$ and a test set $\mathcal{D}_{test}$ by using a 80-20 splitting with stratified random sampling. We normalize each feature in the interval $[0, 1]$ and we train different types of classifiers on $\mathcal{D}_{train}$, i.e., decision trees of different depths and random forests including different numbers

---

[2]`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`: Diabetes, Cod-RNA, Breast Cancer and Sensorless.

Table 2: Datasets

Dataset	#Instances	#Features	Perturbation δ
diabetes	768	8	0.03
cod-rna	59,535	8	0.07
breast-cancer	683	10	0.15
sensorless (1 vs. 11)	10,638	48	0.20

of trees (of depth 3). We cover both standard models trained using the popular sklearn library[3] and robust models trained using a state-of-the art adversarial ML algorithm for decision trees, called TREANT [13].

For each classifier, we leverage the test set $\mathcal{D}_{test}$ to compute different measures: accuracy a, robustness r, its under-approximation $\hat{r}$, and the under-approximation of resilience $\hat{R}$. The robustness r is computed using an exact verification technique from related work [8], while $\hat{r}$ and $\hat{R}$ are under-approximations computed by our analysis (cf. Section 3.2). Similarly to the existing literature, for simplicity we consider a synthetic threat model where each feature can be adversarially corrupted by adding a perturbation δ, which is defined in Table 2 for the different datasets. The value of δ depends on the dataset, because different datasets are drawn from different distributions, hence attacks that are effective on models trained over a given dataset may be too strong or too weak for models trained on a different dataset [5]. The number of features that can be corrupted is defined by the attacker's budget b set in the experiments.

When estimating resilience, we assume the neighborhood $N(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid ||\vec{z} - \vec{x}||_\infty \leq \varepsilon\}$ for a given value of ε defined in the experiments. Observe that the actual resilience R is unknown and only the estimate $\hat{R}$ can be computed by our analysis. Our experiments are designed to investigate the following research questions:

1. May the theoretical shortcomings of robustness actually occur in practice?
2. Can we compute accurate resilience estimates by means of our data-independent stability analysis and are these estimates practically useful?
3. What is the impact of the parameter ε on the resilience estimates that we can compute?
4. What is the performance of our resilience verification technique in terms of running times and how is it affected by the attacker's budget b?

All the experiments are performed using 20 threads.

5.2. Shortcomings of Robustness

The first experiment we carry out motivates our study by showing that the relevant shortcomings of robustness identified on paper might also occur in practical scenarios. To do that, we use the original test set $\mathcal{D}_{test}$ of the different datasets to craft 100 synthetic test sets $\mathcal{D}_{test}^1, \ldots, \mathcal{D}_{test}^{100}$ obtained by replacing each instance $\vec{x}$ in $\mathcal{D}_{test}$ with a randomly sampled instance $\vec{z} \in N(\vec{x})$. We then compute the robustness of the trained classifiers over all the test sets $\mathcal{D}_{test}^i$, reporting the best and worst obtained results to understand to which extent a "lucky" sample of the data distribution might give a false sense of security. To ensure that the synthetic test sets are still representative of the same data distribution used for training, we only consider cases where the classifier roughly preserves the same accuracy computed on the original test set $\mathcal{D}_{test}$.

Table 3 presents the experimental results of our evaluation, assuming an attacker with budget $b = 1$. The table reports for the different datasets the worst robustness r_{min} and the best robustness r_{max} computed over all the generated test sets for different values of ε, as well as the corresponding values of accuracy, noted a_{min} and a_{max} respectively. The table also includes the accuracy a and the robustness r computed on the original test set $\mathcal{D}_{test}$. In the table, we mark in bold the cases where the gap between r_{min} and r_{max} is at least 0.04. The experiments are performed on ensembles with 7 decision trees.

---

[3] https://scikit-learn.org/stable/

Table 3: Shortcomings of robustness (for fixed $b = 1$)

Dataset	ε	Standard Models						Robust Models					
		a	a_{min}	a_{max}	r	r_{min}	r_{max}	a	a_{min}	a_{max}	r	r_{min}	r_{max}
diabetes	0.01	0.714	0.708	0.721	0.649	0.643	0.662	0.727	0.721	0.727	0.714	0.675	0.714
	0.02	0.714	0.708	0.714	0.649	0.630	0.662	0.727	0.714	0.740	0.714	**0.669**	**0.721**
	0.03	0.714	0.688	0.714	0.649	**0.636**	**0.682**	0.727	0.721	0.747	0.714	**0.669**	**0.727**
	0.04	0.714	0.688	0.727	0.649	**0.630**	**0.701**	0.727	0.708	0.747	0.714	**0.675**	**0.734**
cod-rna	0.01	0.775	0.774	0.775	0.686	0.676	0.690	0.750	0.748	0.753	0.714	0.710	0.721
	0.02	0.775	0.773	0.775	0.686	0.665	0.686	0.750	0.749	0.758	0.714	0.711	0.725
	0.03	0.775	0.773	0.775	0.686	0.657	0.686	0.750	0.750	0.760	0.714	0.705	0.723
	0.04	0.775	0.768	0.775	0.686	0.650	0.686	0.750	0.752	0.761	0.714	0.703	0.723
breast-cancer	0.05	0.948	0.948	0.948	0.926	0.926	0.941	0.970	0.933	0.963	0.956	**0.919**	**0.963**
	0.06	0.948	0.933	0.956	0.926	**0.911**	**0.956**	0.970	0.933	0.970	0.956	**0.911**	**0.963**
	0.07	0.948	0.941	0.956	0.926	**0.904**	**0.956**	0.970	0.926	0.963	0.956	**0.911**	**0.963**
	0.08	0.948	0.933	0.970	0.926	**0.904**	**0.963**	0.970	0.926	0.956	0.956	**0.904**	**0.956**
sensorless	0.03	1.000	1.000	1.000	0.985	0.985	0.996	1.000	0.999	1.000	0.999	0.995	0.999
	0.04	1.000	1.000	1.000	0.985	0.985	0.996	1.000	0.996	1.000	0.999	0.990	0.999
	0.05	1.000	1.000	1.000	0.985	0.981	0.993	1.000	0.993	1.000	0.999	0.979	0.999
	0.06	1.000	1.000	1.000	0.985	0.975	0.989	1.000	0.990	1.000	0.999	0.962	0.999

The results show that the size of the interval $[r_{min}, r_{max}]$ is significant in many cases and may reach 0.07 for the highest values of ε, while the size of the interval $[a_{min}, a_{max}]$ is relatively small in comparison, spanning at most 0.04. For example, the robustness of the standard model trained over the breast-cancer dataset suffers from a fluctuation of around 0.05 for $\varepsilon = 0.07$, while the corresponding accuracy fluctuates of just 0.01. Our experiments show that robustness is generally more sensitive to small amounts of noise than accuracy. Remarkably, this observation applies to both standard and robust models trained using TREANT. Robust models offer an improved robustness over standard models, however the interval $[r_{min}, r_{max}]$ may have a significant size also for them, i.e., roughly 0.06 in the worst case. This shows that a security evaluation based on robustness may give a false sense of security for both standard and robust models. We also remark that our experiment still provides a conservative account of the actual limitations of robustness, being based on just 100 synthetic test sets: the actual gap between r_{min} and r_{max} within the neighborhood N may be larger in practice.

5.3. Effectiveness of Resilience Verification

We now investigate the effectiveness of our resilience verification technique. To do that, we would like to show that our estimate $\hat{R}$ is an accurate under-approximation of the actual resilience R and that resilience significantly mitigates the shortcomings of robustness. Unfortunately, since the actual value of resilience is unknown, we can only provide a best-effort answer to the first point. Our evaluation is based on two independent experiments:

1. In the first one, we operate by comparing the similarity between the actual robustness r and its estimate $\hat{r}$ computed by our analysis. We consider the similarity between r and $\hat{r}$ as a proxy for the precision of the stability analysis underlying our resilience verification technique: the more r and $\hat{r}$ are close to each other, the more the stability analysis is effective at detecting the portions of the feature space where the classifier is stable, which suggests that also the estimate $\hat{R}$ is precise, being based on the same stability analysis.

2. In the second one, we refer to the experiment in the previous section and we observe that, if a classifier is not robust on the instance $\vec{z}$ belonging to some $\mathcal{D}_{test}^i$ with $i \in [1, 100]$, then there must exist $\vec{x}$ in $\mathcal{D}_{test}$ such that the classifier is not robust on at least one instance in $N(\vec{x})$ by construction. This allows one to construct an additional test set $\overline{\mathcal{D}_{test}}$, corresponding to the "most unlucky" sampling in the neighborhood of the original $\mathcal{D}_{test}$, i.e., the one with the lowest robustness $\bar{r}$. The measure $\bar{r}$ is

Table 4: Computed measures for different datasets and models (for fixed $b = 1$)

Dataset	ε	# Trees	Depth	Standard Models					Robust Models				
				a	r	$\hat{r}$	$\overline{r}$	$\hat{R}$	a	r	$\hat{r}$	$\overline{r}$	$\hat{R}$
diabetes	0.01	1	3	0.675	0.623	0.623	0.623	0.623	0.682	0.643	0.643	0.643	0.643
		1	5	0.721	0.636	0.636	0.617	0.617	0.675	0.636	0.636	0.630	0.630
		1	7	0.727	**0.610**	0.610	**0.539**	**0.539**	0.695	0.675	0.675	0.636	0.636
		5	3	0.708	0.662	0.643	0.656	0.636	0.727	**0.714**	0.701	**0.675**	**0.662**
		7	3	0.714	0.649	0.630	0.636	0.623	0.727	**0.714**	0.708	**0.675**	**0.662**
		9	3	0.747	0.656	0.630	0.623	0.617	0.753	**0.740**	0.727	**0.695**	**0.688**
cod-rna	0.01	1	3	0.774	0.683	0.683	0.638	0.637	0.750	0.714	0.714	0.698	0.698
		1	5	0.874	**0.433**	0.433	**0.334**	**0.330**	0.810	**0.703**	0.703	**0.643**	**0.641**
		1	7	0.804	**0.575**	0.575	**0.474**	**0.472**	0.810	**0.700**	0.700	**0.637**	**0.634**
		5	3	0.775	**0.686**	0.672	**0.639**	**0.621**	0.752	0.715	0.707	0.698	0.691
		7	3	0.775	**0.686**	0.666	**0.640**	**0.612**	0.750	0.714	0.713	0.698	0.697
		9	3	0.769	**0.677**	0.663	**0.625**	**0.605**	0.750	0.714	0.713	0.698	0.697
breast-cancer	0.05	1	3	0.948	0.904	0.904	0.874	0.874	0.948	**0.941**	0.941	**0.859**	**0.859**
		1	5	0.956	0.911	0.911	0.874	0.874	0.956	**0.941**	0.941	**0.793**	**0.793**
		1	7	0.948	0.904	0.904	0.867	0.867	0.956	**0.941**	0.941	**0.793**	**0.793**
		5	3	0.941	**0.926**	0.904	**0.904**	**0.867**	0.978	**0.948**	0.941	**0.889**	**0.881**
		7	3	0.948	0.926	0.911	0.926	0.881	0.970	**0.956**	0.941	**0.889**	**0.874**
		9	3	0.963	**0.941**	0.919	**0.933**	**0.889**	0.970	**0.956**	0.933	**0.889**	**0.867**
sensorless	0.03	1	3	0.953	**0.763**	0.763	**0.461**	**0.461**	1.000	0.501	0.501	0.501	0.501
		1	5	1.000	**0.933**	0.933	**0.401**	**0.399**	1.000	0.501	0.501	0.501	0.501
		1	7	1.000	**0.933**	0.933	**0.401**	**0.399**	1.000	0.999	0.999	0.993	0.993
		5	3	1.000	**0.978**	0.973	**0.918**	**0.912**	1.000	0.999	0.999	0.993	0.993
		7	3	1.000	**0.985**	0.977	**0.935**	**0.912**	1.000	0.999	0.999	0.993	0.993
		9	3	1.000	**0.992**	0.977	**0.942**	**0.905**	1.000	0.999	0.999	0.993	0.993

interesting, because it is based on an exact robustness verification technique: if $\overline{r}$ is close to $\hat{R}$, then we have a proof that most instances where the classifier is not considered resilient by our analysis are indeed insecure with respect to some evasion attacks.

Note that the second experiment does not just prove the precision of our approximated resilience verification technique, but it also gives a clear security interpretation of the benefits of resilience over robustness.

Table 4 shows the experimental results of our evaluation, assuming $b = 1$. The results highlight that the estimate $\hat{r}$ is a rather precise under-approximation of the actual robustness r: in particular, for individual decision trees $\hat{r}$ always coincides with r. As to the tree ensembles, the gap between the two measures increases, yet it is still quite small (roughly 0.02) for standard models and most often negligible for robust models trained using TREANT, hence we expect that also $\hat{R}$ is a reasonably accurate estimate of the actual resilience R. The table also shows that the gap between r and $\hat{R}$ may be quite significant, both for standard and robust models, often reaching a value of around 0.06 and even reaching 0.10 and more in some cases. We highlight in bold the cases where the gap between r and $\hat{R}$ is at least 0.05. Remarkably, it is apparent that the resilience estimate $\hat{R}$ provides a much more realistic security assessment than robustness, because the value of $\overline{r}$ is much closer to $\hat{R}$ than to r in the very large majority of cases. Since $\overline{r}$ captures effective evasion attacks against instances within close neighborhoods of the test set, this confirms that $\hat{R}$ is not overly conservative in practice.

To further substantiate our claims, Figure 3 shows how robustness is affected when varying the number of synthetic datasets used to construct $\overline{\mathcal{D}_{test}}$, which was built from 100 datasets in our experiments. In particular, the figure plots how the value of $\overline{r}$ changes when varying the number of synthetic datasets from zero to 100. This evaluation is conservative, because we consider the synthetic datasets $\mathcal{D}_{test}^i$ in decreasing order of robustness. The figure shows that less than 20 synthetic datasets are sufficient to have a decrease in
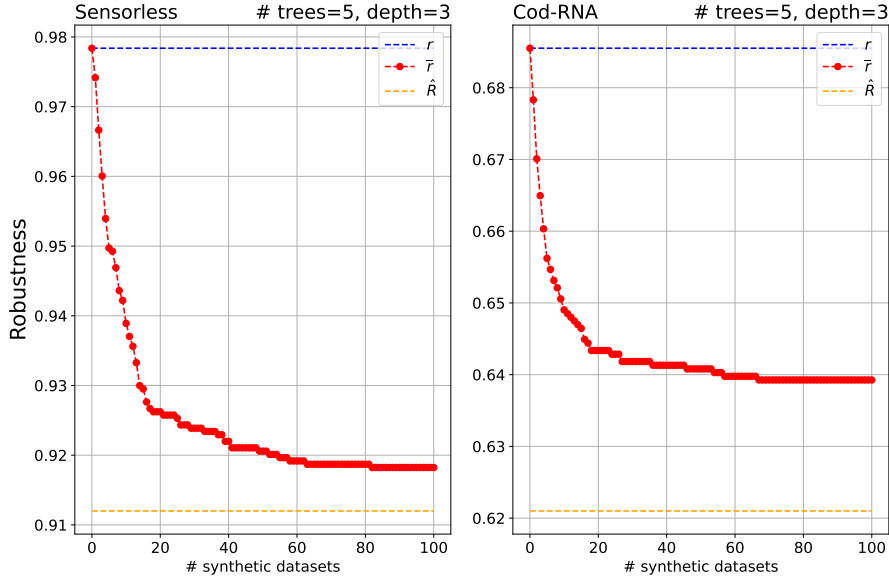
Figure 3: Value of $\overline{r}$ when varying the number of synthetic datasets used to construct $\overline{\mathcal{D}_{test}}$

model robustness of approximately 0.05 for sensorless and 0.04 for cod-rna, i.e., it is easy to find a neighbor $\vec{z} \in N(\vec{x})$ of the original instance $\vec{x}$ for which the model is not robust, hence robustness is not a good measure to assess security against evasion attacks. Moreover, even with a small number of synthetic datasets, the value of $\overline{r}$ is closer to our approximated resilience $\hat{R}$ rather than to the robustness r.

We finally assess the role of the parameter ε on our resilience verification technique. For the sake of readability, we only focus on models trained on the Diabetes dataset using the TREANT algorithm. In particular, we set $b = 1$ and we compute different resilience estimates for different values of ε. Of course, we expect resilience to decrease when increasing the value of ε, because the stability guarantees required on the classifier become more demanding. Still, it is interesting to understand whether the quality of the resilience estimate $\hat{R}$ is affected by the value of ε: to understand this, we compare $\hat{R}$ against $\overline{r}$, because we would like the two measures to be relatively close to each other. Figure 4 plots how the value of our resilience estimate $\hat{R}$ and $\overline{r}$ decrease when increasing ε from 0.01 to 0.05. The figure shows that $\hat{R}$ and $\overline{r}$ are consistently close to each other, with a maximum difference of 0.02. This shows that the computed resilience estimates always capture possible evasion attacks, i.e., the precision of our approximated analysis does not downgrade when increasing the value of ε.

5.4. Performance Evaluation

We finally investigate the performance of our resilience verification technique. For the sake of readability, we only focus on models trained on the Diabetes dataset using the TREANT algorithm. The decision tree analysis is based on a simple tree traversal, hence expected to be very efficient. Our experimental evaluation confirms this intuition: when varying the tree depth from 3 to 15, the analysis always terminates in less than one second.

The performance of the ensemble analysis is subtler to assess though, because that analysis is based on an iterative algorithm and its performance crucially depends on the number of iterations. We are interested in two aspects here:

1. Understanding how much the analysis time (up to convergence) changes when increasing the ensemble size.
2. Understanding how much the quality of the robustness and resilience estimates $\hat{r}, \hat{R}$ changes when increasing the number of iterations, while keeping the same ensemble size.
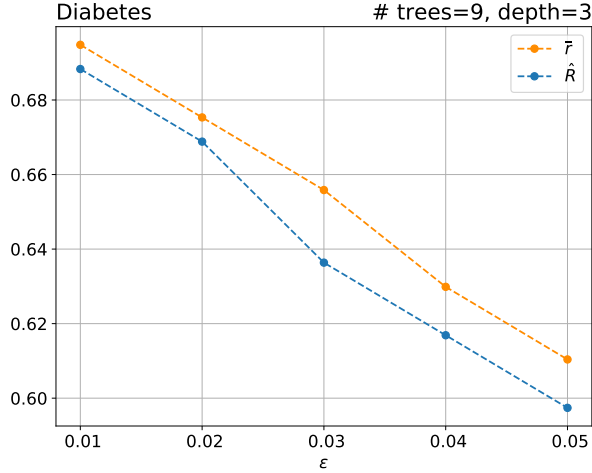
17

Figure 4: Resilience estimates for different values of ε
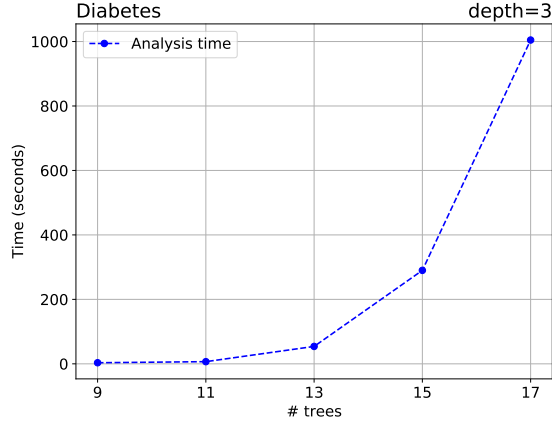

Figure 5: Analysis times when varying the size of the ensemble

The first point provides insights on the scalability of the analysis to increasingly larger models, while the second point allows one to understand whether it is possible to compute useful robustness and resilience estimates even when the analysis becomes intractable and the number of iterations is limited to forcefully stop the analysis before convergence. Indeed, our analysis was deliberately designed to support iterative refinements and parallelization to be equipped against the exponential complexity blowup underlying the verification of decision tree ensembles [12].

Figure 5 shows how the analysis time changes when increasing the size of the ensemble from 9 to 17. Small ensembles with 9 trees can be analyzed in a matter of seconds, while larger ensembles with 17 trees can be analyzed in around 16 minutes. We consider this result promising, because the stability analysis is data-independent, i.e., it can be computed just once and then applied to establish different properties on different test sets. We expect the analysis times to be further improvable by sacrificing a bit of precision, e.g., by aggregating together symbolic attacks which are close to each other. Our next experiment also provides positive results with respect to the scalability of the analysis to larger ensembles.

Figure 6 shows how the values of the estimates $\hat{r}$ and $\hat{R}$ computed on an ensemble of 17 trees change when increasing the number of analysis iterations. The figure shows a desirable trend, with a significant increase of the estimates of robustness and resilience in the first 120 iterations before reaching a plateau. This means that it is possible to establish reasonably accurate estimates of robustness and resilience even
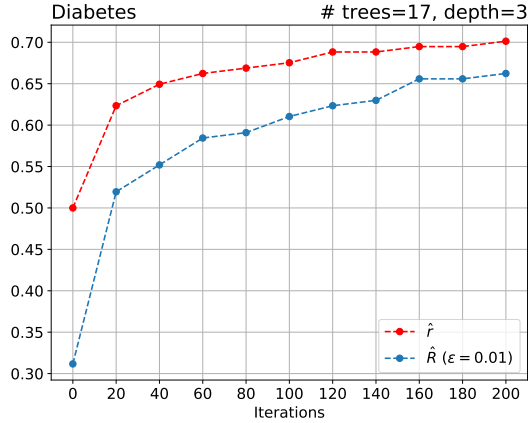
Figure 6: Robustness and resilience estimates when varying the number of analysis iterations
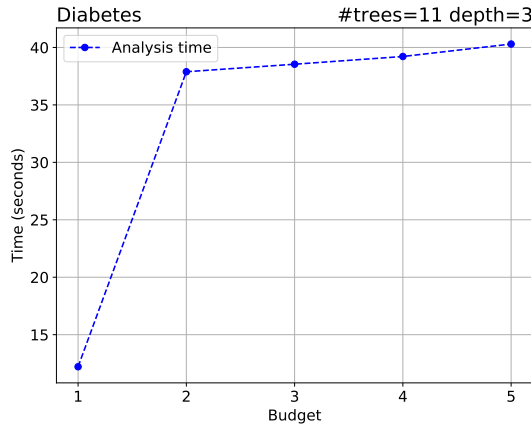

Figure 7: Analysis times when varying the attacker's budget

with a limited number of iterations of the analysis, which is again important to support scalability, because useful results can be established also before analysis convergence. Indeed, our analysis was designed to prioritize portions of the feature space which are intuitively easier to prove as stable (or not).

The last experiment we carry out assesses how the attacker's budget b may impact on the performance of resilience verification. In particular, we investigate how the analysis times change for different values of b, using an ensemble of 11 trees (trained with $b = 5$) analyzed up to convergence. The results of the experiment are shown in Figure 7. As we can see, the impact of the attacker's budget on the analysis times is much more limited than the impact of the size of the ensemble. The analysis times just range from around 12 seconds to around 40 seconds when varying the attacker's budget from 1 to 5.

5.5. Discussion

In the end, our experimental analysis yields positive results. We showed that resilience is *useful*, because robustness may give a false sense of security, which is largely mitigated by the use of resilience. We also proved that our under-approximated resilience verification technique is *precise*: we empirically showed that potentially large gaps between robustness and estimated resilience are motivated, because the estimated resilience is close to the robustness measured over the "most unlucky" sampling performed in a small neighborhood of the original test set. This confirms that our resilience estimates capture effective evasion attacks against plausible samplings of the same data distribution used to build the original test set.

19

Finally, we showed that resilience verification is *feasible* in practice, at least for the relatively small models and simple datasets considered in the present work. For larger models, we showed that the iterative refinement process supported by our analysis technique can be leveraged to obtain useful under-approximations of resilience even before analysis convergence. Moreover, since the soundness proofs of our analysis abstract from several implementation details, e.g., the splitting criterion for symbolic attacks, different heuristics may be tried out to further improve the analysis in terms of both precision and efficiency. We leave a more thorough investigation of this point to future work. On the negative side, preliminary experiments with large, complex datasets like MNIST (784 features) suggest that scalability can be an issue for a data-independent analysis like ours, which generalizes beyond individual instances and is thus inherently more challenging to perform. Similar issues have been independently observed in recent work on global robustness, e.g., training a globally robust ensemble on the Twitter Spam URL dataset takes three days [5], which is incomparably more expensive than training standard models. Improving the performance of algorithms falling in the new research line of data-independent security properties is an important avenue for future research.

6. Related Work

6.1. Global Robustness

Recent independent work in the area also acknowledged the limitations of robustness for the security verification of classifiers [5, 6]. Chen *et al.* defined a set of five new *global robustness* properties, i.e., universally-quantified statements over one or more inputs to the classifier and its corresponding outputs [5]. These properties also include a data-independent stability definition, that requires any two inputs differing just for the value of a fixed set of features to lead to "close" predictions, and the authors proposed a technique to verify this property for a custom type of rule-based classifiers generalizing decision tree ensembles. Although their work shares similarities with ours in terms of research goals, we note several important differences. First, our data-independent stability analysis allows one to identify a subset of the feature space where the classifier is stable, rather than verifying stability over the entire feature space; in other words, we can effectively identify sub-spaces where stability is globally guaranteed. This is more useful in practice, because stability over the entire feature space is often too strong, essentially requiring that the set of non-robust features is unused for classification. Indeed, contrary to resilience, their global robustness properties entirely abstract from the data distribution, which is a sound yet overly conservative choice in the ML setting. This claim is confirmed by the experimental evaluation in [5], which shows that stability cannot be verified for any model (standard or robust) besides those deliberately trained by the authors to enforce that property. Rather, we are able to use our resilience notion to perform practically useful security evaluations of existing ML models, while still overcoming the limitations of robustness confirmed by our experiments.

Leino *et al.*, instead, introduced globally-robust neural networks [6]. They proposed a technique to train neural networks with a special output $\perp$, designed to signal predictions performed on a subset of the feature space that is too close to the decision boundary, hence potentially subject to evasion attacks. Their notion of global robustness requires that any two "close" inputs must lead to the same output, unless $\perp$ is returned for at least one of the two inputs. Our data-independent stability analysis essentially captures the same notion, because its output could also be used to return $\perp$ on all the instances that do not fall on a stable subset of the feature space (see Section 3.3). However, observe that global robustness as defined in [6] cannot be used to reason about the security of traditional classifiers which do not use the $\perp$ label and their technical treatment is quite different from ours, because in this paper we operate on decision tree models rather than on neural networks.

6.2. Security of Decision Trees

The security certification of decision trees and decision tree ensembles has received an increasing amount of attention by the research community during the last years. The first seminal work on the topic is due to Kantchelian, Tygar and Joseph [11]. They showed that computing minimal adversarial perturbations for tree ensembles is NP-complete in general, and proposed a mixed-integer linear programming technique

for the task. This motivated additional work in the area by Chen *et al.* [9]. They investigated restricted fragments of the problem which are tractable in polynomial time and proposed an approximated, yet sound, approach to verify robustness against L_∞-norm attackers. In later work, Ranzato and Zanella proposed the use of abstract interpretation to mitigate the complexity of robustness verification by means of a sound over-approximation of the ensemble predictions, again assuming L_∞-norm attackers [4]. Calzavara, Ferrara and Lucchese showed that abstract interpretation could also be used to verify the robustness of decision trees against an expressive threat model, where the attacker is encoded as an arbitrary imperative program [14]. All of these approaches only prove robustness and cannot be directly used to prove resilience without a data-independent stability analysis, like the one proposed in the present paper.

A different line of work which is more directly comparable to ours is related to the VoTE checker by Törnblom and Nadjm-Tehrani [12]. Given a tree ensemble, VoTE computes the set of all the equivalence classes induced by the ensemble over the feature space. Once the equivalence classes have been computed, VoTE uses a property checker module to verify different properties on them. The idea of computing equivalence classes from the tree ensemble yields a data-independent analysis approach, however there are important differences with respect to our work. First, their analysis is not adversary-aware and the security implications of data-independence are not explored by the authors, since they just verify traditional robustness properties. We rather clarify the practical relevance of data-independence by introducing a new formal security notion called resilience and we design experiments to show its empirical value on real datasets. Moreover, computing all the equivalence classes of an ensemble is infeasible in general due to their combinatorial explosion, as also observed by the authors of VoTE. In a follow-up work, the same authors proposed an abstraction-refinement approach to mitigate this complexity problem [15]. However, contrary to our analysis, their extension is not proved sound, which is an important requirement for the analysis of classifiers deployed in adversarial settings.

Finally, we mention that several papers discussed new algorithms for training decision trees and decision tree ensembles that are robust to evasion attacks [10, 13, 16, 17, 18, 19]. These works are complementary to our verification technique, because they present training algorithms and not verification algorithms. Concretely, they discuss how to learn decision tree models with improved robustness properties, while we discuss how to verify the resilience properties of arbitrary decision tree models (both standard and robust). Our work motivated why resilience is superior to robustness in terms of practical security guarantees and showed that even robustly-trained are not necessarily resilient.

7. Conclusion

We criticized the traditional robustness measure used to assess the security of classifiers against evasion attacks and we proposed an improved measure called resilience, which provides additional assurances on unsampled data outside the test set. We then discussed how resilience can be estimated by combining traditional tools for robustness verification with a data-independent stability analysis, which does not depend on a specific test set. We finally proposed a formally sound data-independent stability analysis for decision trees and decision tree ensembles, which we evaluated on public datasets with positive results. By using our stability analysis, we managed to establish precise and practically useful estimates of resilience within a reasonable amount of time.

We see several interesting directions for future work. First, we would like to extend our analysis to gradient-boosted decision trees [20] and more sophisticated threat models, e.g., where adversarial manipulations are expressed in terms of Euclidean distances or via rewriting rules [14]. Moreover, we plan to design new training algorithms based on resilience minimization and a new boosting algorithm for robust tree ensembles based on our stability analysis. Indeed, since our analysis is able to identify the weak spots where the ensemble might be unstable, the boosting algorithm may train additional trees designed to provide stability on that part of the feature space. Finally, we would like to generalize our resilience verification approach to classifiers different than decision trees and decision tree ensembles, e.g., SVMs [21] and deep neural networks [22]. This would be a valuable, yet challenging research direction.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: Y. Bengio, Y. LeCun (Eds.), 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014. URL: `http://arxiv.org/abs/1312.6199`.

[2] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL: `http://arxiv.org/abs/1412.6572`.

[3] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018. URL: `https://openreview.net/forum?id=rJzIBfZAb`.

[4] F. Ranzato, M. Zanella, Abstract interpretation of decision tree ensemble classifiers, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 5478–5486. URL: `https://aaai.org/ojs/index.php/AAAI/article/view/5998`.

[5] Y. Chen, S. Wang, Y. Qin, X. Liao, S. Jana, D. A. Wagner, Learning security classifiers with verified global robustness properties, in: Y. Kim, J. Kim, G. Vigna, E. Shi (Eds.), CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, ACM, 2021, pp. 477–494. URL: `https://doi.org/10.1145/3460120.3484776`. doi:10.1145/3460120.3484776.

[6] K. Leino, Z. Wang, M. Fredrikson, Globally-robust neural networks, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 6212–6222. URL: `http://proceedings.mlr.press/v139/leino21a.html`.

[7] J. R. Quinlan, Induction of decision trees, Machine learning 1 (1986) 81–106.

[8] S. Calzavara, C. Lucchese, F. Marcuzzi, S. Orlando, Feature partitioning for robust tree ensembles and their certification in adversarial scenarios, EURASIP J. Inf. Secur. 2021 (2021) 12. URL: `https://doi.org/10.1186/s13635-021-00127-0`. doi:10.1186/s13635-021-00127-0.

[9] H. Chen, H. Zhang, S. Si, Y. Li, D. S. Boning, C. Hsieh, Robustness verification of tree-based models, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 12317–12328. URL: `https://proceedings.neurips.cc/paper/2019/hash/cd9508fdaa5c1390e9cc329001cf1459-Abstract.html`.

[10] S. Calzavara, C. Lucchese, G. Tolomei, Adversarial training of gradient-boosted decision trees, in: W. Zhu, D. Tao, X. Cheng, P. Cui, E. A. Rundensteiner, D. Carmel, Q. He, J. X. Yu (Eds.), Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019, ACM, 2019, pp. 2429–2432. URL: `https://doi.org/10.1145/3357384.3358149`. doi:10.1145/3357384.3358149.

[11] A. Kantchelian, J. D. Tygar, A. D. Joseph, Evasion and hardening of tree ensemble classifiers, in: M. Balcan, K. Q. Weinberger (Eds.), Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, 2016, pp. 2387–2396. URL: `http://proceedings.mlr.press/v48/kantchelian16.html`.

[12] J. Törnblom, S. Nadjm-Tehrani, Formal verification of input-output mappings of tree ensembles, Sci. Comput. Program. 194 (2020) 102450. URL: `https://doi.org/10.1016/j.scico.2020.102450`. doi:10.1016/j.scico.2020.102450.

[13] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, S. Orlando, Treant: training evasion-aware decision trees, Data Min. Knowl. Discov. 34 (2020) 1390–1420. URL: `https://doi.org/10.1007/s10618-020-00694-9`. doi:10.1007/s10618-020-00694-9.

[14] S. Calzavara, P. Ferrara, C. Lucchese, Certifying decision trees against evasion attacks by program analysis, in: L. Chen, N. Li, K. Liang, S. A. Schneider (Eds.), Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II, volume 12309 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 421–438. URL: `https://doi.org/10.1007/978-3-030-59013-0\_21`. doi:10.1007/978-3-030-59013-0_21.

[15] J. Törnblom, S. Nadjm-Tehrani, An abstraction-refinement approach to formal verification of tree ensembles, in: A. B. Romanovsky, E. Troubitsyna, I. Gashi, E. Schoitsch, F. Bitsch (Eds.), Computer Safety, Reliability, and Security - SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings, volume 11699 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 301–313. URL: `https://doi.org/10.1007/978-3-030-26250-1\_24`. doi:10.1007/978-3-030-26250-1_24.

[16] H. Chen, H. Zhang, D. S. Boning, C. Hsieh, Robust decision trees against adversarial examples, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of *Proceedings of Machine Learning Research*, PMLR, 2019, pp. 1122–1131. URL: `http://proceedings.mlr.press/v97/chen19m.html`.

[17] M. Andriushchenko, M. Hein, Provably robust boosted decision stumps and trees against adversarial attacks, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 12997–13008. URL: `https://proceedings.neurips.cc/paper/2019/hash/4206e38996fae4028a26d43b24f68d32-Abstract.html`.

[18] D. Vos, S. Verwer, Efficient training of robust decision trees against adversarial examples, in: M. Meila, T. Zhang (Eds.),

Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 10586–10595. URL: `http://proceedings.mlr.press/v139/vos21a.html`.

[19] F. Ranzato, M. Zanella, Genetic adversarial training of decision trees, in: F. Chicano, K. Krawiec (Eds.), GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021, ACM, 2021, pp. 358–367. URL: `https://doi.org/10.1145/3449639.3459286`. doi:10.1145/3449639.3459286.

[20] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of statistics (2001) 1189–1232.

[21] B. Biggio, B. Nelson, P. Laskov, Support vector machines under adversarial label noise, in: C. Hsu, W. S. Lee (Eds.), Proceedings of the 3rd Asian Conference on Machine Learning, ACML 2011, Taoyuan, Taiwan, November 13-15, 2011, volume 20 of *JMLR Proceedings*, JMLR.org, 2011, pp. 97–112. URL: `http://proceedings.mlr.press/v20/biggio11/biggio11.pdf`.

[22] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. T. Vechev, Fast and effective robustness certification, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 10825–10836. URL: `https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html`.

Appendix A. Proofs

We provide proofs of the formal results in the paper.

Appendix A.1. Proof of Theorem 1

The proof leverages a key technical lemma (Lemma 1) formalizing the soundness of the tree annotation function in Algorithm 1. More specifically, we prove that the tree annotation function always produces a *well-annotated* decision tree according to the following definition.

Definition 5 (Well-Annotated Decision Tree). The node n of the decision tree t is *well-annotated* by the set of symbolic attacks S if and only if, for every instance $\vec{x} \in \mathcal{X}$ and every $\vec{z} \in A(\vec{x})$ such that n is traversed in the prediction $t(\vec{z})$, S contains an element $\langle I_1^{pre}, \dots, I_d^{pre} \rangle \triangleright \langle I_1^{post}, \dots, I_d^{post} \rangle_k$ such that $\vec{x} \in \langle I_1^{pre}, \dots, I_d^{pre} \rangle$, $\vec{z} \in \langle I_1^{post}, \dots, I_d^{post} \rangle$ and k is the minimum cost to pay to make $\vec{x}$ traverse n. We say that the decision tree t is well-annotated if and only if all its nodes are well-annotated by the set of symbolic attacks stored in their *sym* attribute.

Lemma 1 (Soundness of Tree Annotation). *The call* ANNOTATE(t, S) *returns a well-annotated decision tree, provided that the root of t is well-annotated by S.*

Proof. The proof is by induction on the depth of the tree t. If the tree has depth 1, then it includes a single node, i.e., the root, and the conclusion follows by the assumption that the root of t is well-annotated by S. Otherwise, we have $t = \sigma(f, v, t_r, t_r)$ for some feature f, threshold v and sub-trees t_l, t_r. The function then computes two new sets of symbolic attacks S_l, S_r before invoking ANNOTATE(t_l, S_l) and ANNOTATE(t_r, S_r). Hence, the desired conclusion follows by inductive hypothesis, provided that we are able to show that the roots of t_l and t_r are well-annotated by S_l and S_r respectively. We just prove the former, since the latter uses an equivalent reasoning.

Pick any instance $\vec{x} \in \mathcal{X}$ and consider any $\vec{z} \in A(x)$, we observe that S must contain an element s such that $\vec{x} \in s.pre$, $\vec{z} \in s.post$ and $s.cost = 0$, because all instances must traverse the root. Assume $z_f \leq v$, we prove that REFINELEFT(s, f, v) returns a set of symbolic attacks $S'_l \subseteq S_l$ such that there exists $s' \in S'_l$ such that $\vec{x} \in s'.pre$, $\vec{z} \in s'.post$ and $s'.cost$ is the minimum cost to pay to make $\vec{x}$ traverse the left child of the root. Assume $s.pre = \langle I_1^{pre}, \dots, I_d^{pre} \rangle$, $s.post = \langle I_1^{post}, \dots, I_d^{post} \rangle$ and $I_f^{atk} = \langle \delta_l, \delta_r \rangle$, we discriminate four cases:

- If $I_f^{pre} = I_f^{post}$ and $x_f \leq v$, we leverage the observation that $\vec{z} \in s.post$ and $z_f \leq v$, hence the condition $I_f^{post} \cap (-\infty, v] \neq \emptyset$ at line 6 must be satisfied. In this case, S'_l must contain an s'' such that:

    - $s''.pre = \langle I_1^{pre}, \dots, I_{f-1}^{pre}, I_f^{pre} \cap (-\infty, v], I_{f+1}^{pre}, \dots, I_d^{pre} \rangle$
    - $s''.post = \langle I_1^{post}, \dots, I_{f-1}^{post}, I_f^{post} \cap (-\infty, v], I_{f+1}^{post}, \dots, I_d^{post} \rangle$

23

- $s''.cost = 0$

The conclusion follows by the observation that s'' satisfies the three required conditions on s'.

- If $I_f^{pre} = I_f^{post}$ and $x_f > v$, we leverage the observation that $\vec{z} \in A(x)$ and $z_f \leq v$. This implies that $\delta_l < 0$, $x_f \in (v, v - \delta_l]$ and $z_f \in (v + \delta_l, v]$; moreover, we must have $c_f \leq b$. By combining all this information and the observation that $s.cost = 0$, we conclude that the condition at line 14 must be satisfied. In this case, S_l' must contain an s'' such that:

  - $s''.pre = \langle I_1^{pre}, \ldots, I_{f-1}^{pre}, I_f^{pre} \cap (v, v - \delta_l], I_{f+1}^{pre}, \ldots, I_d^{pre} \rangle$
  - $s''.post = \langle I_1^{post}, \ldots, I_{f-1}^{post}, I_f^{post} \cap (v + \delta_l, v], I_{f+1}^{post}, \ldots, I_d^{post} \rangle$
  - $s''.cost = c_f$

  The conclusion follows by the observation that s'' satisfies the three required conditions on s'.

- If $I_f^{pre} \neq I_f^{post}$ and $x_f \leq v$, we leverage the observation that $\vec{z} \in s.post$ and $z_f \leq v$, hence the condition $I_f^{post} \cap (-\infty, v] \neq \emptyset$ at line 6 must be satisfied. In this case, S_l' must contain an s'' such that:

  - $s''.pre = \langle I_1^{pre}, \ldots, I_{f-1}^{pre}, I_f^{pre} \cap (-\infty, v - \min(0, \delta_l)], I_{f+1}^{pre}, \ldots, I_d^{pre} \rangle$
  - $s''.post = \langle I_1^{post}, \ldots, I_{f-1}^{post}, I_f^{post} \cap (-\infty, v], I_{f+1}^{post}, \ldots, I_d^{post} \rangle$
  - $s''.cost = k$

  The conclusion follows by the observation that s'' satisfies the three required conditions on s'.

- If $I_f^{pre} \neq I_f^{post}$ and $x_f > v$, we leverage the observation that $\vec{z} \in A(x)$ and $z_f \leq v$. This implies that $\delta_l < 0$, $x_f \in (v, v - \delta_l]$ and $z_f \in (v + \delta_l, v]$. We then observe that $\vec{z} \in s.post$ and $z_f \leq v$, hence the condition $I_f^{post} \cap (-\infty, v] \neq \emptyset$ at line 6 must be satisfied. In this case, S_l' must contain an s'' such that:

  - $s''.pre = \langle I_1^{pre}, \ldots, I_{f-1}^{pre}, I_f^{pre} \cap (-\infty, v - \min(0, \delta_l)], I_{f+1}^{pre}, \ldots, I_d^{pre} \rangle$
  - $s''.post = \langle I_1^{post}, \ldots, I_{f-1}^{post}, I_f^{post} \cap (-\infty, v], I_{f+1}^{post}, \ldots, I_d^{post} \rangle$
  - $s''.cost = k$

  The conclusion follows by the observation that s'' satisfies the three required conditions on s'.

$\square$

We now move back to the proof of the theorem. Consider an instance $\vec{x}$ and an adversarial perturbation $\vec{z} \in A(\vec{x})$ such that $t(\vec{z}) \neq t(\vec{x})$. This means that there exist two leaves $\lambda(y)$ and $\lambda'(y')$ with $y \neq y'$ such that $t(\vec{x}) = y$ and $t(\vec{z}) = y'$. By Lemma 1, t must be well-annotated after line 2, hence we can make the following observations by Definition 5:

1. Since $\vec{x} \in A(\vec{x})$, the leaf $\lambda(y)$ must contain a symbolic attack $s = \langle I_1^{pre}, \ldots, I_d^{pre} \rangle \rhd \langle I_1^{post}, \ldots, I_d^{post} \rangle_k$ such that $\vec{x} \in \langle I_1^{pre}, \ldots, I_d^{pre} \rangle$, $\vec{x} \in \langle I_1^{post}, \ldots, I_d^{post} \rangle$ and $k = 0$.
2. Since $\vec{z} \in A(\vec{x})$, the leaf $\lambda'(y')$ must contain a symbolic attack $s' = \langle J_1^{pre}, \ldots, J_d^{pre} \rangle \rhd \langle J_1^{post}, \ldots, J_d^{post} \rangle_{k'}$ such that $\vec{x} \in \langle J_1^{pre}, \ldots, J_d^{pre} \rangle$, $\vec{z} \in \langle J_1^{post}, \ldots, J_d^{post} \rangle$ and k' is the minimum cost to pay to make $\vec{x}$ traverse $\lambda'(y')$. This cost must be greater than 0, because $t(\vec{z}) \neq t(\vec{x})$ implies $\vec{z} \neq \vec{x}$.

This implies that line 9 is reachable and $s.pre \cap s'.pre \neq \emptyset$, hence a new symbolic attack s'' is added to the return value U at lines 10-13. Thus, we just need to show that s'' satisfies the conditions of the theorem:

- We have that $\vec{x} \in s''.pre = s.pre \cap s'.pre$, by points 1 and 2.

- We have that $\vec{z} \in s'.post$ by point 2. Moreover, since $\vec{z} \in A(\vec{x})$, we must have $\vec{z} \in \vec{x} + \langle I_1^{atk}, \ldots, I_d^{atk} \rangle$ by definition of adversarial manipulation. Since $\vec{x} \in s''.pre$ by the previous point, we get $\vec{z} \in s''.pre + \langle I_1^{atk}, \ldots, I_d^{atk} \rangle$, hence we conclude $\vec{z} \in s'.post \cap (s''.pre + \langle I_1^{atk}, \ldots, I_d^{atk} \rangle) = s''.post$ as desired.

24

Appendix A.2. Proof of Theorem 2

We prove the following invariant for the outer loop: for every instance $\vec{x} \in \mathcal{X}$ and every adversarial manipulation $z \in A(\vec{x})$ such that $T(\vec{z}) \neq T(\vec{x})$, there exists $s \in C \cup E$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$.

Let $T = \{t_1, \ldots, t_n\}$, consider an instance $\vec{x}$ and an adversarial manipulation $\vec{z} \in A(\vec{x})$ such that $T(\vec{z}) \neq T(\vec{x})$. We first prove the base case, i.e., we show that the invariant holds when no loop iteration has taken place. Initially, $C = \cup_i U_i$ where each U_i is computed by calling $\textsc{Analyze}(t_i)$. Since $T(\vec{z}) \neq T(\vec{x})$, there exists $t_i \in T$ such that $t_i(\vec{z}) \neq t_i(\vec{x})$. Hence, there exists $s \in U_i$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$ by Theorem 1. The conclusion then follows by definition of C.

Assume now the invariant holds up to a given iteration, we show it is preserved at the next iteration. By inductive hypothesis there exists $s \in C \cup E$ such that $\vec{x} \in s.pre$ and $\vec{z} \in s.post$. We distinguish two cases. If $s \in E$, then the conclusion is immediate because nothing is ever removed from E. If instead $s \in C$, we show that each iteration of the inner loop cannot break the outer loop invariant. In particular, assume some $s' \in C$ is processed by an iteration of the inner loop, leading to updated C' and E' respectively. We can distinguish the following cases at the end of the iteration:

- If $C' = C \setminus \{s'\}$ and $E' = E$, then there exists y such that $T(s.pre) = T(s.post) = \{y\}$. This implies that for all instances $\vec{w} \in s.pre \cup s.post$ we have $T(\vec{w}) = y$, thanks to the first soundness condition. Since $T(\vec{x}) \neq T(\vec{z})$, we have that either $\vec{x} \notin s.pre$ or $\vec{z} \notin s.post$, hence $s' \neq s$ and the loop invariant is preserved.

- If $C' = (C \setminus \{s'\}) \cup \textsc{Split}(s')$ and $E' = E$, the loop invariant is preserved by the second soundness condition.

- if $C' = C \setminus \{s'\}$ and $E' = E \cup \{s'\}$, then $C' \cup E' = C \cup E$ and thus the loop invariant is preserved.