Abstract Interpretation of Decision Tree Ensemble Classifiers

Francesco Ranzato, Marco Zanella
Dipartimento di Matematica, University of Padova, Italy
{ranzato, mzanella}@math.unipd.it

Abstract

We study the problem of formally and automatically verifying robustness properties of decision tree ensemble classifiers such as random forests and gradient boosted decision tree models. A recent stream of works showed how abstract interpretation, which is ubiquitously used in static program analysis, can be successfully deployed to formally verify (deep) neural networks. In this work we push forward this line of research by designing a general and principled abstract interpretation-based framework for the formal verification of robustness and stability properties of decision tree ensemble models. Our abstract interpretation-based method may induce complete robustness checks of standard adversarial perturbations and output concrete adversarial attacks. We implemented our abstract verification technique in a tool called *silva*, which leverages an abstract domain of not necessarily closed real hyperrectangles and is instantiated to verify random forests and gradient boosted decision trees. Our experimental evaluation on the MNIST dataset shows that *silva* provides a precise and efficient tool which advances the current state of the art in tree ensembles verification.

1 Introduction

Adversarial machine learning (Goodfellow, McDaniel, and Papernot 2018; Kurakin, Goodfellow, and Bengio 2017) is a hot topic studying vulnerabilities of machine learning (ML) in adversarial scenarios. Adversarial examples have been found in diverse application fields of ML such as image classification, spam filtering, malware detection, and the current defense techniques include adversarial model training, input validation, testing and automatic verification of learning algorithms (Goodfellow, McDaniel, and Papernot 2018). Formal verification of ML classifiers started to be an active field of investigation, in particular for robustness properties of (deep) neural networks. A classifier is stable for some (typically very small) perturbation of its input samples which represents an adversarial attack when it assigns the same class to all the samples within that perturbation, so that impercetible malicious alterations of input objects should not deceive a stable classifier. Formal verification methods for neural networks may rely on a number of different techniques: linear approximation of functions (Weng et al. 2018; Zhang et al. 2018), semidefinite relaxations (Raghunathan, Steinhardt, and Liang 2018), logical SMT solvers (Huang et al. 2017; Katz et al. 2017), symbolic interval propagation (Wang et al. 2018a), abstract interpretation (Gehr et al. 2018; Singh et al. 2018; 2019) or hybrid synergistic approaches (Anderson et al. 2019; Wang et al. 2018b). Abstract interpretation (Cousot and Cousot 1977) is a *de facto* standard technique used since forty years for designing static analysers and verifiers of programming languages. Recently, abstract interpretation has been successfully applied for designing precise and scalable robustness verification tools of (deep) neural network models (Gehr et al. 2018; Singh et al. 2018; 2019). While all these verification techniques consider neural networks as ML model, in this work we focus on decision tree ensemble methods, such as random forests and gradient boosted decision tree models, which are widely applied in different fields having sensible adversarial scenarios, notably image classification, malware detection, intrusion detection and spam filtering.

Contributions. Following the aforementioned stream of works applying abstract interpretation for certifying ML models, we design a general abstract interpretation-based framework for the formal verification of stability properties of decision tree ensemble models. Our verification algorithm of ensembles of decision trees: (1) is domain agnostic, since it can be instantiated to any abstract domain which represents properties of real vectors, such as simple hyperrectangles of real intervals or more involved linear relations; (2) is firmly based on the basic soundness principle of abstract interpretation and correspondingly rely on sound approximations of split functions used in decision tree classifiers; (3) under certain assumptions may induce complete robustness checks against adversarial perturbations; (4) is able to output concrete adversarial samples. Our formal verification methodology has been implemented in C in a tool called *silva* (Latin for *forest* and acronym of *Silvarum Interpretatione Lator Valens Analysis*) which leverages an abstract domain of possibly open real hyperrectangles and has been applied to random forests and gradient boosted decision trees. Our experimental evaluation on the standard MNIST dataset

and on the Sensorless dataset used by (Chen et al. 2019a) shows that *silva* provides a precise and efficient tool both for deriving the robustness metrics of forest classifiers and for performing a complete check of the robustness to adversarial perturbations of input samples. Our evaluation also compared the performance of *silva* against a recent robustness verification tool of tree ensembles (Törnblom and Nadjm-Tehrani 2019b), and this showed that *silva* improves on the current state of the art. The following three images display:

on the left, an original image O from the test set of MNIST, correctly classified as 7 by a random forest using 100 decision trees with maximum depth 100; in the middle, an image A which is automatically generated by *silva* as adversarial attack of O for a perturbation ± 1 of the brightness values of its pixels; on the right, a gray/black/white image showing which pixels change from O to A where gray means unchanged, black $+1$, white -1. The image A is an adversarial example because it is classified as either 2 or 7 (i.e., there is a tie between the scores of 2 and 7) by the same 100×100 random forest which classifies O as 7. When *silva* infers that an input sample O is not robust for a given input perturbation $\mathcal{P}(O)$, it also outputs a symbolic representation in its underlying abstract domain of a whole set of adversarial attacks in $\mathcal{P}(O)$, so that the above attack image A has been randomly selected in $\mathcal{P}(O)$.

Related Work. It is only recently that adversarial attacks and robustness of tree ensemble classifiers started to be a subject of investigation (Andriushchenko and Hein 2019; Calzavara et al. 2019; Calzavara, Lucchese, and Tolomei 2019; Chen et al. 2019a; 2019b; Einziger et al. 2019; Kantchelian, Tygar, and Joseph 2016; Törnblom and Nadjm-Tehrani 2018; Törnblom and Nadjm-Tehrani 2019b; Törnblom and Nadjm-Tehrani 2019a). The most related work is the robustness verification tool of tree ensembles by Törnblom and Nadjm-Tehrani (2019b) called VoTE. This is an abstraction-refinement procedure which iteratively refines a partition of the input space where each block of the partition is an hyperrectangle. The main differences between *silva* and VoTE can be summarized as follows: (i) VoTE is not designed as a principled abstract interpretation of decision tree ensembles, in particular it is not parametric on a generic abstract domain but it is specifically tailored for partitions whose blocks are hyperrectangles; (ii) the soundness and completeness properties of the verification algorithm of VoTE are not formally proved; (iii) VoTE does not output counterexamples to the robustness of some input sample. By contrast, *silva* is a robustness verification algorithm firmly based on the principles of abstract interpretation (Cousot and Cousot 1977) which: (1) is fully parametric on a given abstract domain (thus generalizing to so-called relational abstract domains) and its abstract elements and operations (in particular, bottom and meet); (2) is endowed with a formal soundness and completeness proof, in particular the conditions guaranteeing a sound/complete verification check are explicitly given; (3) outputs counterexamples to robustness when they exist; (4) can therefore benefit of all the well-established methods of abstract interpretation used in static program analysis, e.g., advanced relational abstract domains and completeness reasoning (Rival and Yi 2020).

2 Background

Classifiers. Given an input space $X \subseteq \mathbb{R}^d$ of feature vectors and a finite set of classification labels $\mathcal{L} = \{\ell_1, ..., \ell_m\}$, a classifier is a (total) function $C : X \to \wp_+(\mathcal{L})$, where $\wp_+(\mathcal{L})$ denotes the set of nonempty subsets of $\mathcal{L}$, which assigns at least one label to every input in X. The application of C on a set $Y \subseteq X$ of inputs is simply denoted by $C(Y) \triangleq \bigcup_{\mathbf{x}\in Y} C(\mathbf{x})$. ML classifiers typically output the set of labels having maximal score for some function $score : X \to \mathbb{R}^{|\mathcal{L}|}$, i.e. $C(\mathbf{x}) = \{\ell_i \in \mathcal{L} \mid \forall j.\ score(\mathbf{x})_i \geq score(\mathbf{x})_j\}$. Score functions are the output of a learning algorithm on some training dataset $D \subseteq X \times \mathcal{L}$ which explores a hypothesis space of functions and returns a best fit for the training set D according to some learning principle.

Decision Trees and Tree Ensembles. Decision trees are used both for classification and regression tasks. A decision tree t is a finite branching structure where: (i) each internal node n (including a unique root) has a finite set of successor nodes $Succ_t(n)$ and is endowed with a decision rule, also called split condition, $split_n : X \to Succ_t(n)$ on input vectors, so that the branches outgoing from n represent the possible outputs of $split_n$; (ii) the set of leaves of t is denoted by $\textsc{leaves}(t)$ and each leaf stores some information on labels which is used for a final decision/classification, so that a path from the root to a leaf represents a classification rule. Although decision rules in general can be of any type, the most common decision trees (Breiman et al. 1984) use Boolean conditions which are univariate hard splits of the form $\mathbf{x}_i \leq k$ for some attribute $\mathbf{x}_i$ of its input vector $\mathbf{x}$, hence defining binary decision trees. Leaves typically store tuples in $\mathbb{R}^{|\mathcal{L}|}$ which play the role of output for a decision tree, which therefore induces a score function $t : X \to \mathbb{R}^{|\mathcal{L}|}$ for input samples in X, also denoted by $score_t(\mathbf{x})$. In a decision tree application $t(\mathbf{x})$, by following the path π of nodes in t from the root to a leaf denoted by $l = leaf(t(\mathbf{x}))$, one may collect the constraints which satisfy the split conditions encountered in every internal node of π and decorate the leaf l with the conjunction of these constraints, denoted by $constr_t(l)$, which is assumed to be logically consistent. For univariate hard splits, where the possible constraints of internal nodes are $\mathbf{x}_i \leq k$ or $\mathbf{x}_i > k$, $constr_t(l)$ therefore determines a not necessarily closed hyperrectangle (also called box), i.e., $box_t(l) \triangleq \{\mathbf{x} \in \mathbb{R}^d \mid constr_t(l) \models \mathbf{x}\}$. Also, the leaves of a decision tree t induce a partition of the input space X, where each $l \in \textsc{leaves}(t)$ induces a block $block_t(l) \triangleq \{\mathbf{x} \in X \mid leaf(t(\mathbf{x})) = l\}$, so that the number of these nonempty blocks is less than or equal to the number of leaves of t and $block_t(l) \subseteq box_t(l)$.

Ensemble methods allow to combine several constituent learning models to improve their accuracy and generalizability. For score-based classifiers, every classifier is fed with

the same input and computes its own score function, so that scores from multiple classifiers are then combined into a single score typically by applying some voting mechanism. The two most common voting schemes are: (Max): each constituent classifier contributes with a label-indexed tuple of either 1 (for labels with maximal score) or 0 (non maximal scores), and then votes are added; (Average): each constituent classifier computes a label-indexed tuple of scores in $[0, 1]$, and then the average score for each label is computed.

Tree ensembles or forests are ensemble methods for decision tree classifiers. Several ways of training a forest have been designed and investigated, where random forests (RFs) (Breiman 2001) and gradient boosted decision trees (GBDTs) (Friedman 2001) are the most successful. The experiment evaluation of our tool *silva* will focus on RFs and GBDTs, although our verification technique is directly applicable to any type of decision tree ensembles.

Abstract Interpretation. A numerical abstract domain is a tuple $\langle \mathcal{A}, \sqsubseteq^{\mathcal{A}}, \gamma \rangle$ where $\langle \mathcal{A}, \sqsubseteq^{\mathcal{A}} \rangle$ is at least a preordered set of abstract values and the concretization function $\gamma^{\mathcal{A}} : \mathcal{A} \to \wp(\mathbb{R}^d)$ monotonically preserves the ordering relation, namely, $a_1 \sqsubseteq^{\mathcal{A}} a_2$ implies $\gamma^{\mathcal{A}}(a_1) \subseteq \gamma^{\mathcal{A}}(a_2)$. The intuition is that an abstract domain $\mathcal{A}$ defines a symbolic abstract representation of sets ranging in the concrete domain $\wp(\mathbb{R}^d)$: well-known examples of numerical abstract domains include intervals, zonotopes, octagons, octahedra and polyhedra (see the survey (Miné 2017) and the book (Rival and Yi 2020)). Sometimes we denote an abstract domain by $\mathcal{A}_d$ to highlight the dimension $d \in \mathbb{N}$ in the concrete domain $\wp(\mathbb{R}^d)$. Given a concrete k-ary operation on vectors $f : (\mathbb{R}^d)^k \to \mathbb{R}^d$, for some $k \in \mathbb{N}$, an abstract function $f^{\mathcal{A}} : \mathcal{A}^k \to \mathcal{A}$ is called a *sound* (or *correct*) (over-)approximation of f when for all $(a_1, ..., a_k) \in \mathcal{A}^k$, $\{f(\mathbf{x_1}, ..., \mathbf{x_k}) \mid \forall i.\ \mathbf{x_i} \in \gamma^{\mathcal{A}}(a_i)\} \subseteq \gamma^{\mathcal{A}}(f^{\mathcal{A}}(a_1, ..., a_k))$ holds, while $f^{\mathcal{A}}$ is defined to be *complete* (or *exact*) when an equality holds. In words, this means that soundness holds when $f^{\mathcal{A}}(a_1, ..., a_k)$ never misses a concrete computation of f on some input $(\mathbf{x_1}, ..., \mathbf{x_k})$ which is abstractly represented by $(a_1, ..., a_k)$, while completeness implies that each abstract computation $f^{\mathcal{A}}(a_1, ..., a_k)$ is precisely a symbolic abstract representation of the set of concrete computations of f on all the concrete inputs that are abstractly represented by $(a_1, ..., a_k)$.

We will use an abstract domain of not necessarily closed real hyperrectangles $\mathcal{H}$, which is commonly called interval or box abstraction in abstract interpretation. The domain of closed/open bounded/unbounded real intervals is denoted by $\mathcal{I}$ and $[\![l, u]\!]$ denotes an interval in $\mathcal{I}$ where the lower and upper bounds $l, u \in \mathbb{R} \cup \{+\infty, -\infty\}$ are such that $l \leq u$ and $[\![$ may be either (or [and analogously for $]\!]$, e.g. $(-\infty, k], (k, +\infty) \in \mathcal{I}$. The concretization map $\gamma^{\mathcal{I}} : \mathcal{I} \to \wp(\mathbb{R})$ for intervals has an obvious definition. An abstract element of $\mathcal{H} \triangleq \mathcal{I}^d$ is a d-dimensional vector of components in $\mathcal{I}$ and $\gamma^{\mathcal{H}} : \mathcal{H} \to \wp(\mathbb{R}^d)$ is simply the d-th product of $\gamma^{\mathcal{I}}$, that is, $\gamma^{\mathcal{H}}(I_1, ..., I_d) \triangleq \{\mathbf{x} \in \mathbb{R}^d \mid \forall j.\ \mathbf{x}_j \in \gamma^{\mathcal{I}}(I_j)\}$. Let us also recall that $\langle \mathcal{H}, \sqsubseteq^{\mathcal{H}} \rangle$ is a complete lattice for the standard componentwise ordering: $(I_1, ..., I_d) \sqsubseteq^{\mathcal{H}} (I'_1, ..., I'_d)$ iff for all j, $\gamma^{\mathcal{I}}(I_j) \subseteq \gamma^{\mathcal{I}}(I'_j)$.

3 Stability of Classifiers

Accuracy on a test set T is the *de facto* standard metric for assessing a classification model C, that is, the ratio between the number of correctly classified samples and the size of T. However, it is a growing and widely held belief (Goodfellow, McDaniel, and Papernot 2018) that accuracy is not enough in ML, because the robustness properties of C may well affect its safety and generalizability. Robustness is here generalized to a notion of *stability* of C on a given input.

Definition 3.1 (Stability). A classifier C is *stable* on a neighborhood $N \subseteq X$ of an input sample $\mathbf{x} \in N$, denoted by $\textsc{IsStable}(C, N, \mathbf{x})$, when $C(N) = C(\mathbf{x})$. □

Thus, adversarial regions or perturbations of an input sample play the role of neighborhoods. The standard approach to robustness consists in using a distance function $\delta : X \times X \to [0, +\infty)$ and a threshold $\epsilon \in [0, +\infty)$ which define a perturbation $\mathcal{P}_{\delta,\epsilon}(\mathbf{x}) \triangleq \{\mathbf{x}' \in X \mid \delta(\mathbf{x}, \mathbf{x}') \leq \epsilon\}$. L_p norms on $\mathbb{R}^d$ have been investigated and in particular the most common distance function is induced by the L_∞ maximum norm (Carlini and Wagner 2017): $\mathcal{P}_{\infty,\epsilon}(\mathbf{x}) \triangleq \{\mathbf{x}' \in X \mid \max(|\mathbf{x}_1 - \mathbf{x}'_1|, ..., |\mathbf{x}_d - \mathbf{x}'_d|) \leq \epsilon\}$.

Stability of a classifier C on a test set $T \subseteq X$ for a perturbation $\mathcal{P}$ is defined by the ratio $\frac{|\{\mathbf{x} \in T \mid \textsc{IsStable}(C, \mathcal{P}(\mathbf{x}), \mathbf{x})\}|}{|T|}$. If $\textsc{IsCorrect}(C, \mathbf{x})$ denotes that C predicts the correct label $\ell_{\mathbf{x}}$ of a labeled input sample $(\mathbf{x}, \ell_{\mathbf{x}})$, then the following metrics combine stability with accuracy w.r.t. a perturbation $\mathcal{P}$:

robustness: $$\frac{|\{\mathbf{x} \in T \mid \textsc{IsCorrect}(C, \mathbf{x}) \wedge \textsc{IsStable}(C, \mathcal{P}(\mathbf{x}), \mathbf{x})\}|}{|T|}$$

fragility: $$\frac{|\{\mathbf{x} \in T \mid \textsc{IsCorrect}(C, \mathbf{x}) \wedge \neg\, \textsc{IsStable}(C, \mathcal{P}(\mathbf{x}), \mathbf{x})\}|}{|T|}$$

vulnerability: $$\frac{|\{\mathbf{x} \in T \mid \neg\, \textsc{IsCorrect}(C, \mathbf{x}) \wedge \textsc{IsStable}(C, \mathcal{P}(\mathbf{x}), \mathbf{x})\}|}{|T|}$$

breakage: $$\frac{|\{\mathbf{x} \in T \mid \neg\, \textsc{IsCorrect}(C, \mathbf{x}) \wedge \neg\, \textsc{IsStable}(C, \mathcal{P}(\mathbf{x}), \mathbf{x})\}|}{|T|}$$

Of course, a direct verification of stability is not feasible, as this would require to apply the classifier to an infinite number of vectors, and even by restricting to finite samples, let us say, in $\{0, 1\}^d$, this number is exponential in d.

4 Stability Verification Framework

Abstract Interpretation of Classifiers. Given a classifier $C : X \to \wp_+(\mathcal{L})$ for an input space $X \in \wp(\mathbb{R}^d)$, the concretization function $\gamma^{\mathcal{A}} : \mathcal{A} \to \wp(\mathbb{R}^d)$ of an abstract domain $\mathcal{A}$ is specialized to the space X simply by defining $\gamma^{\mathcal{A}_X} : \mathcal{A} \to \wp(X)$ as $\gamma^{\mathcal{A}_X}(a) \triangleq \gamma^{\mathcal{A}}(a) \cap X$. In order to simplify the notation, in the following $\gamma^{\mathcal{A}_X}$ will be simply denoted by $\gamma^{\mathcal{A}}$, which is therefore meant as a function in $\mathcal{A} \to \wp(X)$. A *sound abstract interpretation* of C on a numerical abstract domain $\mathcal{A}$ is a computable function $C^{\mathcal{A}} : \mathcal{A} \to \wp_+(\mathcal{L})$ which over-approximates the labels predicted by C, i.e., for all $a \in \mathcal{A}$, $C(\gamma^{\mathcal{A}}(a)) \subseteq C^{\mathcal{A}}(a)$ holds. An abstract classifier $C^{\mathcal{A}}$ is designed by relying on a sound abstract score function $score^{\mathcal{A}} : \mathcal{A}_d \to \mathcal{A}_{|\mathcal{L}|}$, where soundness here means that for all $a \in \mathcal{A}_d$,

$$\{score(\mathbf{x}) \in \mathbb{R}^{|\mathcal{L}|} \mid \mathbf{x} \in \gamma^{\mathcal{A}_d}(a)\} \subseteq \gamma^{\mathcal{A}_{|\mathcal{L}|}}(score^{\mathcal{A}}(a)).$$

A simple method for defining a sound $C^{\mathcal{A}}$ by leveraging a sound $score^{\mathcal{A}}$ is:

$$C^{\mathcal{A}}(a) \triangleq \{\ell_i \in \mathcal{L} \mid \forall j \neq i.\ \sup\{\mathbf{s}_i \mid \mathbf{s} \in \gamma^{\mathcal{A}_{|\mathcal{L}|}}(score^{\mathcal{A}}(a))\} \geq \inf\{\mathbf{s}_j \mid \mathbf{s} \in \gamma^{\mathcal{A}_{|\mathcal{L}|}}(score^{\mathcal{A}}(a))\}\} \quad (1)$$

Hence, a label ℓ_i is not included in $C^{\mathcal{A}}(a)$ when there exists a different label ℓ_j dominating ℓ_i, namely, the minimum abstract score of ℓ_j is strictly greater than the maximum abstract score of ℓ_i. For example, for $|\mathcal{L}| = 4$ and the domain of boxes $\mathcal{H}$, if $score^{\mathcal{H}}(a) = (\ell_1/[4,6], \ell_2/[0,2], \ell_3/[5,7], \ell_4/[1,4])$ then $C^{\mathcal{A}}(a) = \{\ell_1, \ell_3\}$.

Lemma 4.1. *$C^{\mathcal{A}}(a)$ defined by* (1) *is sound.*

Sound abstract classifiers induce a correct stability check as follows.

Theorem 4.2. *Let $C^{\mathcal{A}}$ be a sound abstraction of C on $\mathcal{A}$. If $\mathbf{x} \in X$, $a \in \mathcal{A}$ and $\mathcal{P}(\mathbf{x}) \subseteq X$ is a perturbation of $\mathbf{x}$ such that $\mathcal{P}(\mathbf{x}) \subseteq \gamma^{\mathcal{A}}(a)$ then $C^{\mathcal{A}}(a) = C(\mathbf{x}) \Rightarrow$ IsStable$(C, \mathcal{P}(\mathbf{x}), \mathbf{x})$.*

Of course, the converse in general does not hold, since $C^{\mathcal{A}}(a)$ may well include spurious labels introduced by the approximation of $\mathcal{A}$, so that it may happen that $C^{\mathcal{A}}(a) \supsetneq C(\mathbf{x})$ although IsStable$(C, \mathcal{P}(\mathbf{x}), \mathbf{x})$ still holds. We will show that under certain conditions it is possible to achieve a *complete stability verification* method: $C^{\mathcal{A}}(a) = C(\mathbf{x}) \Leftrightarrow$ IsStable$(C, \mathcal{P}(\mathbf{x}), \mathbf{x})$.

Abstraction of Scores for Tree Ensembles. Let us now consider a forest classifier $F : X \to \wp_+(\mathcal{L})$ which combines a set of decision trees and where, by an abuse of notation, F also denotes this set of trees. For the sake of clarity, in the following we focus on the average score function Avg, although our method can be easily adapated to the Max score and to different voting schemes. Given an abstract domain $\mathcal{A}$, for all $t \in F$ we assume that each leaf $l \in$ Leaves(t) stores a triple $\langle score_t(l), constr_t(l), constr_t^{\mathcal{A}}(l)\rangle$ where: (i) $score_t(l) \in \mathbb{R}^{|\mathcal{L}|}$ is the score of l in t; (ii) $constr_t(l)$ is the logical constraint of l; (iii) $constr_t^{\mathcal{A}}(l) \in \mathcal{A}$ is a sound abstract constraint such that for all samples $\mathbf{x} \in X$, $constr_t(l) \models \mathbf{x} \Rightarrow \mathbf{x} \in \gamma^{\mathcal{A}}(constr_t^{\mathcal{A}}(l))$. For example, with the abstract domain $\mathcal{H}$, a leaf could store the following information:

$$\langle(\ell_1/0.3, \ell_2/0.5),\ \mathbf{x}_1 > 4.1 \wedge \mathbf{x}_3 \leq 2.02,\ \mathbf{x}_1 \in [0, +\infty), \mathbf{x}_2 \in (-\infty, +\infty), \mathbf{x}_3 \in (0, 3]\rangle.$$

As recalled in Section 2, the average score function for a forest F is $\text{Score}_F(\mathbf{x}) \triangleq \frac{1}{|F|} \cdot \sum_{t \in F} score_t(\mathbf{x}) \in \mathbb{R}^{|\mathcal{L}|}$. We then define an abstract average score $\text{Score}_F^{\mathcal{A}} : \mathcal{A}_d \to \mathcal{A}_{|\mathcal{L}|}$ for the forest F as follows:

$$\text{Score}_F^{\mathcal{A}}(a) \triangleq \frac{1}{|F|} \cdot^{\mathcal{A}} \sum\nolimits_{t \in F}^{\mathcal{A}} score_t^{\mathcal{A}}(a) \quad (2)$$

where: (i) for all $t \in F$, $score_t^{\mathcal{A}} : \mathcal{A}_d \to \mathcal{A}_{|\mathcal{L}|}$ is required to be a sound abstraction of $score_t : X \to \mathbb{R}^{|\mathcal{L}|}$; (ii) $\cdot^{\mathcal{A}}$ and $\sum^{\mathcal{A}}$ are, resp., sound approximations of the standard scalar multiplication and vector sum in $\mathbb{R}^{|\mathcal{L}|}$.

Theorem 4.3. $\text{Score}_F^{\mathcal{A}}$ *is a sound abstraction of* Score_F.

The general definition (1) of abstract classifier is instantiated to the abstract score function $\text{Score}_F^{\mathcal{A}}$ defined by (2), thus obtaining, by Lemma 4.1, a sound abstract forest classifier:

$$F^{\mathcal{A}} : \mathcal{A} \to \wp_+(\mathcal{L}) \quad (3)$$

Abstraction of Reachable Leaves. We assume that the abstract domain $\mathcal{A}$ is endowed with a bottom $\bot_{\mathcal{A}} \in \mathcal{A}$ and with a sound approximation $\sqcap^{\mathcal{A}} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ of set intersection such that: (i) $\gamma^{\mathcal{A}}(\bot_{\mathcal{A}}) = \varnothing$; (ii) for all $a, a' \in \mathcal{A}$, $\gamma^{\mathcal{A}}(a) \cap \gamma^{\mathcal{A}}(a') \subseteq \gamma(a \sqcap^{\mathcal{A}} a')$.

Given a tree $t \in F$ and a set of input samples $Y \subseteq X$, let $\text{Reach}_t(Y)$ denote the set of leaves of t reached by some sample in Y, i.e., $\text{Reach}_t(Y) \triangleq \{leaf(t(\mathbf{x})) \in \text{Leaves}(t) \mid \mathbf{x} \in Y\}$. Our stability verification algorithm needs to approximate Reach_t by an abstract function $\text{Reach}_t^{\mathcal{A}} : \mathcal{A} \to \wp(\text{Leaves}(t))$ which is defined by relying on the abstract intersection and bottom in $\mathcal{A}$ and the abstract logical constraint stored by leaves:

$$\text{Reach}_t^{\mathcal{A}}(a) \triangleq \{l \in \text{Leaves}(t) \mid a \sqcap^{\mathcal{A}} constr_t^{\mathcal{A}}(l) \neq \bot_{\mathcal{A}}\} \quad (4)$$

Lemma 4.4. $\text{Reach}_t^{\mathcal{A}}$ *is a sound abstraction of* Reach_t.

Let us notice that if $\text{Reach}_t^{\mathcal{A}}(a) = \{l\}$, for some leaf l, then all the input samples represented by the abstract value a follow the same path in t, meaning that we have a complete reachability information with no loss of precision.

Completeness of Tree Decision Rules. Our verification algorithm relies on an abstract domain $\mathcal{A}$ which is required to be complete for the tree decision rules of internal nodes. For simplicity, let us consider the most common case of a Boolean decision rule $split : X \to \{tt, ff\}$ (a generalization to decision rules with multiple outputs in $\{o_1, ..., o_k\}$ is straightforward). In this case, completeness means that $\mathcal{A}$ provides a complete abstraction of $split$ for both true and false outputs, namely, we assume two abstract functions $split_{tt}^{\mathcal{A}}, split_{ff}^{\mathcal{A}} : \mathcal{A} \to \mathcal{A}$ such that for all $a \in A$, $\{\mathbf{x} \in \gamma^{\mathcal{A}}(a) \mid split(\mathbf{x}) = tt\} = \gamma^{\mathcal{A}}(split_{tt}^{\mathcal{A}}(a))$ and $\{\mathbf{x} \in \gamma^{\mathcal{A}}(a) \mid split(\mathbf{x}) = ff\} = \gamma^{\mathcal{A}}(split_{ff}^{\mathcal{A}}(a))$.

It turns out that the abstract domain of hyperrectangles $\mathcal{H}$ is complete for all the univariate hard splits which are most commonly used in RFs and GBDTs. This is a consequence of the fact that a univariate split such as $\mathbf{x}_i \leq k$ or $\mathbf{x}_i > k$ exactly defines a hyperrectangle in $\mathcal{H}$, the domain $\mathcal{H}$ is a complete lattice and its meet $\sqcap^{\mathcal{H}}$ corresponds to intersecting hyperrectangles, meaning that $\sqcap^{\mathcal{H}}$ is complete for the intersection: for all $h, h' \in \mathcal{H}$, $\gamma^{\mathcal{H}}(h \sqcap^{\mathcal{H}} h') = \gamma^{\mathcal{H}}(h) \cap \gamma^{\mathcal{H}}(h')$. Thus, for a decision rule such as $\mathbf{x}_i \leq k$, the corresponding complete abstract functions $split_{tt}^{\mathcal{H}}, split_{ff}^{\mathcal{H}} : \mathcal{H} \to \mathcal{H}$ are simply given by $split_{tt}^{\mathcal{H}}(I_1, ..., I_i, ..., I_d) \triangleq (I_1, ..., I_i \sqcap^{\mathcal{I}} (-\infty, k], ..., I_d)$ and $split_{ff}^{\mathcal{H}}(I_1, ..., I_i, ..., I_d) \triangleq (I_1, ..., I_i \sqcap^{\mathcal{I}} (k, +\infty), ..., I_d)$, where $\sqcap^{\mathcal{I}}$ is the meet of intervals.

It is worth remarking that abstract interpretation includes a very wide range of abstract domains to be used for different purposes (Miné 2017; Rival and Yi 2020). This allows us to satisfy the above completeness assumption for different

types of decision rules by using suitable abstract domains. For example, the use of more general omnivariate decision trees has been investigated by (Yildiz and Alpaydin 2001), where decision rules are given by a linear combination of attributes of an input vector $\mathbf{x}$. Then, in omnivariate trees, for decision rules of type $\pm\mathbf{x}_i \pm \mathbf{x}_j \leq k$, involving at most two features, it turns out that the octagon abstract domain is complete for them, while for more general decision rules of type $\sum_i \mathbf{x}_i \leq k$, possibly involving all the components of $\mathbf{x}$, one may resort to the octahedron abstract domain, which is complete.

Stability Verification Algorithm. We design a *complete verification* algorithm for the stability of a forest F by relying on the completeness of $\mathcal{A}$ for the decision rules of the trees in F. We assume that $\mathcal{A}$ is endowed with a top $\top_{\mathcal{A}} \in \mathcal{A}$ representing the lack of information, namely, $\gamma^{\mathcal{A}}(\top_{\mathcal{A}}) = \mathbb{R}^d$. For all $t \in F$ and $l \in \text{LEAVES}(t)$, the abstract constraint $constr_t^{\mathcal{A}}(l)$ is obtained by composing the abstract decision rules $split_{n_i}^{\mathcal{A}}$ along the nodes n_i of the path π from the root of t to the leaf l, i.e., if $\pi = n_1 \rightarrow ... \rightarrow n_k \rightarrow l$ and $split_{n_1}^{\mathcal{A}}, ..., split_{n_k}^{\mathcal{A}}$ are the abstract split conditions for the internal nodes $n_1, ..., n_k$ of π then $constr_t^{\mathcal{A}}(l) \triangleq split_{n_k}^{\mathcal{A}}(... split_{n_1}^{\mathcal{A}}(\top_A))$. By the completeness assumption on the split conditions, it turns out that for all $t \in F$, $l \in \text{LEAVES}(t)$ and $\mathbf{x} \in X$, $constr_t(l) \models \mathbf{x} \Leftrightarrow \mathbf{x} \in \gamma^{\mathcal{A}}(constr_t^{\mathcal{A}}(l))$ holds. This means that the logical constraint of the leaf l representing the conjunction of all the split conditions along the path π is symbolically represented by an abstract value in $\mathcal{A}$ with no loss of precision.

Given an abstract value $a \in \mathcal{A}$, an abstract partition P of a, denoted by $P \in \text{Par}^{\mathcal{A}}(a)$, is defined to be a nonempty set $P \in \wp(\mathcal{A})$ such that $\{\gamma^{\mathcal{A}}(b) \mid b \in P\}$ is a standard partition of the set $\gamma^{\mathcal{A}}(a)$ (i.e., a set of nonempty subsets of $\gamma^{\mathcal{A}}(a)$ which are pairwise disjoint and whose union is $\gamma^{\mathcal{A}}(a)$). The notion of partition refinement is standard: given $P, P' \in \text{Par}^{\mathcal{A}}(a)$, P is a refinement of P', denoted by $P \preceq P'$, when the standard partition $\{\gamma^{\mathcal{A}}(b) \mid b \in P\}$ is a refinement of $\{\gamma^{\mathcal{A}}(b') \mid b' \in P'\}$, i.e., for all $b \in P$, there exists $b' \in P'$ such that $\gamma^{\mathcal{A}}(b) \subseteq \gamma^{\mathcal{A}}(b')$.

Algorithm 1 describes in pseudocode our stability verification methodology. The procedure IsStable$^{\mathcal{A}}$ takes as input a forest F, an abstract value $a \in \mathcal{A}$ and a sample $\mathbf{x} \in X$ such that a abstracts a perturbation $\mathcal{P}(\mathbf{x})$ of $\mathbf{x}$, i.e., $\mathcal{P}(\mathbf{x}) \subseteq \gamma^{\mathcal{A}}(a)$. The possible outputs are True and False, where False means instability and this is output together with an abstract representation of a set of input counterexamples in $\mathcal{P}(\mathbf{x})$ to the stability of F on $\mathbf{x}$.

The procedure IsStable$^{\mathcal{A}}$ maintains a set $L \in \wp(\mathcal{L})$ (line 15), which is an over-approximation of the labels computed by $F(\gamma^{\mathcal{A}}(a))$, and a partition $P \in \text{Par}^{\mathcal{A}}(a)$ (line 16) of the input abstract value $a \in \mathcal{A}$. The set of labels L is iteratively refined by removing from L the labels that are inferred to be outside of $F(\gamma^{\mathcal{A}}(a))$, while the partition P is refined by using the abstract constraints $constr_t^{\mathcal{A}}(l)$ of the reachable leaves l of some $t \in F$. Each tree $t \in F$ may contribute to refine both L and P and the procedure IsStable$^{\mathcal{A}}$ will scan (for-loop at lines 17-23) the trees in F until a True/False is eventually output (in the worst case, at line 24, all the trees of F will be processed). The current tree $t \in F$ of the for-loop is explored by the function Refine which may either infer instability or refine both L and P. At the exit of a call Refine(t, P), $L' \in \wp(\mathcal{L})$ and $P' \in \wp(\mathcal{A})$ will satisfy the invariant conditions at lines 17 and 22: L' will be a refinement of L and still an over-approximation of $F(\gamma^{\mathcal{A}}(a))$, while P' will be a refinement of the partition P. In a function call Refine(t, P), the abstract elements b of the partition P are iteratively processed in order to compute the set of labels $L_b = F^{\mathcal{A}}(b)$ as defined by the abstract classifier in (3), which is an over-approximation of $F(\gamma^{\mathcal{A}}(b))$. If L_b is disjoint with $F(\mathbf{x})$ (line 6) then F surely classifies all the samples in $\gamma^{\mathcal{A}}(b)$ differently from $\mathbf{x}$, meaning that F is unstable on the whole set $\gamma^{\mathcal{A}}(b)$. In this case, the procedure IsStable outputs b, which is a symbolic representation of these counterexamples (the adversarial image depicted in Section 1 has been obtained from one such output). If, instead, $F^{\mathcal{A}}(b) = F(\mathbf{x})$ holds (line 10) then no sample in $\gamma^{\mathcal{A}}(b)$ can be currently inferred to be a counterexample to the stability of F, meaning that the abstract element b does not need to be refined, so that b is just added to P' at line 10. Finally, in the remaining case (line 11) we have that $F^{\mathcal{A}}(b) \cap F(\mathbf{x}) \neq \emptyset$ and $F^{\mathcal{A}}(b) \neq F(\mathbf{x})$, meaning that no conclusion on the stability on $\gamma^{\mathcal{A}}(b)$ can be correctly inferred. In this case, Refine computes $\text{REACH}_t^{\mathcal{A}}(b)$ which provides an over-approximation of the leaves reachable by $\gamma^{\mathcal{A}}(b)$ and then refines (the block represented by) b by adding to P' the corresponding set of abstract values $constr_t^{\mathcal{A}}(l) \in \mathcal{A}$ for all the leaves $l \in \text{REACH}_t^{\mathcal{A}}(b)$. If Refine$(t, P)$ does not output False then the set L' is filled at line 9 with all the labels in $F^{\mathcal{A}}(b)$: at the exit, this L'

Algorithm 1 Stability Verification of Forests

```
1:  function Refine(t ∈ F, P ∈ Par^A(a))
2:      ℘(L) ∋ L' ← ∅
3:      ℘(A) ∋ P' ← ∅
4:      for all b ∈ P do
5:          ℘(L) ∋ L_b ← F^A(b)
6:          if L_b ∩ L_x = ∅ then output False
7:                          ▷ Invariant: ∀x' ∈ γ^A(b). F(x') ≠ F(x)
8:          else
9:              L' ← L' ∪ L_b
10:             if L_b = L_x then P' ← P' ∪ {b}
11:             else P' ← P' ∪ {constr_t^A(l) ∈ A | l ∈ REACH_t^A(b)}
12:     return ⟨L', P'⟩
13: procedure IsStable^A(F ∈ Forest, a ∈ A, x ∈ X)
14:     ℘(L) ∋ L_x ← F(x)
15:     ℘(L) ∋ L ← L
16:     Par^A(a) ∋ P ← {a}
17:     for all t ∈ F do                ▷ Invariant: L ⊇ F(γ^A(a))
18:         if L = L_x then output True
19:             ▷ Invariant: ∀b ∈ P.∀x' ∈ γ^A(b). F(x') ⊆ F(x)
20:         else
21:             ⟨L', P'⟩ ← Refine(t, P)
22:                 ▷ Invariant: L' ⊆ L ∧ P' ∈ Par^A(a) ∧ P' ⪯ P
23:             L ← L'; P ← P'
24:     if L = L_x then output True else output False
```

is a refinement of the previous set L, because it has been computed by using the blocks of a refined partition. Hence, a function call $\text{REFINE}(t, P)$ either proves instability or refines both L and P.

Theorem 4.5. *Let us assume that $\mathcal{A}$ is complete for all the decision rules of trees in F. Then, for all $\mathbf{x} \in X$ and $a \in \mathcal{A}$,* $\text{ISSTABLE}^{\mathcal{A}}(F, a, \mathbf{x}) \Leftrightarrow \text{ISSTABLE}(F, \gamma^{\mathcal{A}}(a), \mathbf{x})$.

Thus, if the abstract value a represents precisely an adversarial perturbation $\mathcal{P}(\mathbf{x})$ then $\text{ISSTABLE}^{\mathcal{A}}(F, a, \mathbf{x})$ provides a verification algorithm for the robustness of a correctly classified sample $\mathbf{x}$ under adversarial attacks in $\mathcal{P}(\mathbf{x})$. Let us also mention that when Algorithm 1 is used to derive the stability metrics defined in Section 2 to assess the robustness properties of some classifier F on some test set T, if T is large then it makes sense to set a timeout mechanism in $\text{ISSTABLE}^{\mathcal{A}}$ for the verification of a single test sample $\mathbf{x} \in T$ for efficiency reasons: in this case, the stability metrics will be given within a small approximation interval.

5 Experimental Evaluation

We implemented Algorithm 1 in a tool called *silva* whose source code in C (about 5K LOC) is available on GitHub (Ranzato and Zanella 2019).

The function REFINE of Algorithm 1 is independent of any criteria for selecting at line 4 the next abstract value $b \in P$ to be used as input for the abstract forest classifier $F^{\mathcal{A}}(b)$ and then possibly to be refined at line 11. If the forest F is actually stable then in each function call $\text{REFINE}(t, P)$ at line 21 all the abstract values of the partition P will be eventually processed, meaning that the ordering used for scanning P ultimately does not affect the efficiency of REFINE. However, if the forest F is instead unstable then it would be more efficient to detect a counterexample to its stability as soon as possible. Thus, our actual implementation of $\text{REFINE}(t, P)$ relies on a best-first search of the tree t, where the next abstract value $b \in P$ is chosen by maximizing a heuristic function which estimates the likelihood for an abstract value b to represent a counterexample (due to lack of space the details are here omitted and can be retrieved from the source code of *silva*).

We used *silva* for inferring stability, robustness, fragility, vulnerability and breakage (as defined in Section 3) of random forest classifiers on MNIST, which have been trained with different combinations of number of trees, maximum tree depth, splitting criteria (Gini and entropy impurity measures) and voting schemes (average and max). We compared the performance of *silva* against the robustness verification tool called VoTE (Törnblom and Nadjm-Tehrani 2019b), already discussed in Section 1, both for random forests and gradient boosted decision trees. In our experiments RFs have been trained by scikit-learn while CatBoost has been used for GBDTs. Let us recall that CatBoost uses complete oblivious decision trees, where the same splitting criterion is used across an entire level of a tree. Since all these forest classifiers rely on univariate hard splits of type $\mathbf{x}_i \leq k$, *silva* has been instantiated to the hyperrectangle abstract domain $\mathcal{H}$. Our experiments were run on a AMD Ryzen 7 1700X 3.0GHz CPU.

RF		Gini, max			Gini, average		
B	d	acc.%	stab.%	time(s)	acc.%	stab.%	time(s)
5	5	66.7	14.3	0.6	76.4	18.1	0.5
5	25	90.5	21.4	0.9	90.6	21.4	0.9
5	50	90.5	19.5	1.0	90.5	19.5	1.0
25	5	82.8	12.1	3.0	85.7	16.8	3.6
25	25	96.0	31.7	22.5	96.1	31.8	24.4
25	50	96.0	26.5	30.2	96.0	26.5	30.0
50	5	84.0	12.7	40.8	85.8	18.3	662.9
50	25	96.6	35.1 $\pm$1.7	965.4	96.6	35.2 $\pm$1.7	970.8
50	50	96.5	35.3 $\pm$2.1	1126.2	96.5	35.3 $\pm$2.1	1136.8

RF		entropy, max			entropy, average		
B	d	acc.%	stab.%	time(s)	acc.%	stab.%	time(s)
5	5	67.5	9.6	0.5	76.1	20.2	0.5
5	25	91.3	28.4	0.8	91.3	28.4	0.8
5	50	91.3	22.8	0.9	91.3	22.8	0.9
25	5	81.3	16.5	3.7	85.6	19.7	6.9
25	25	96.2	39.4	28.7	96.2	39.4	28.8
25	50	96.2	36.4	36.7	96.2	36.4	34.9
50	5	83.4	20.8	67.4	85.4	24.2	811.9
50	25	96.5	43.1 $\pm$1.5	863.9	96.5	43.1 $\pm$1.5	874.1
50	50	96.6	41.3 $\pm$1.4	824.5	96.6	41.3 $\pm$1.4	826.1

Table 1: Stability of different RFs on MNIST.

Setup. An automatic verifier Ver of robustness properties of a ML classifier C could be used for two main purposes:

P_1: to assess the robustness properties of C by inferring some stability metrics on a large test set T.

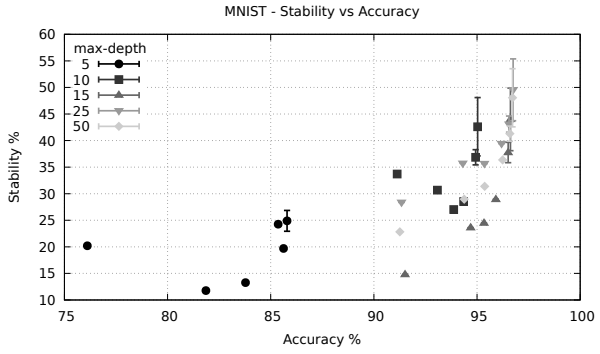
P_2: to check the robustness of C on an input sample $\mathbf{x} \in X$ against some adversarial perturbation $\mathcal{P}(\mathbf{x})$; when Ver infers that C is not robust on $\mathbf{x}$ then Ver should also output a (set of) counterexample(s) in $\mathcal{P}(\mathbf{x})$;

A complete verifier such as *silva* always outputs a TRUE/FALSE answer for each input sample, provided that computational time and memory constraints are met. When a complete verifier Ver is used for reaching a purpose of type P_1, it makes sense to add a timeout option to Ver to set a time limit for verifying the stability of a single sample $\mathbf{x} \in T$, so that if the timeout applies then the stability check on $\mathbf{x}$ is considered to be inconclusive. In our experiments using *silva* for evaluating the robustness properties of tree ensemble classifiers, we utilized a timeout per sample, so that the stability metrics may be given by a percentage ranging within an interval, thus taking into account inconclusive stability checks.

Let us recall that MNIST consists of 70000 gray scale pictures of hand-written digits (with $\mathcal{L} = \{0, ..., 9\}$), where each image of 28×28 pixels is encoded as a vector of 784 integer values in $[0, 255]$ representing the brightness of a pixel (0 black, 255 white). The standard training set of MNIST consists of 60000 samples, while its test set T includes the remaining 10000 samples. We considered the standard perturbation $\mathcal{P}_{\infty,\epsilon}$ (Carlini and Wagner 2017) with $\epsilon = 1$, meaning that adversarial attacks may brighten/darken each pixel up to a 0.5% magnitude. Hence, for each input image $\mathbf{x}$, its perturbation $\mathcal{P}_{\infty,1}(\mathbf{x})$ includes 3^{784} potential attacks.

Stability of RFs. Table 1 shows the accuracy and stability percentages and the total verification time on the whole test set of MNIST for different random forest classifiers trained by combining 4 parameters: number B of decision trees, maximum tree depth d, training criterion (Gini and entropy)

and voting scheme (max and average). In this experiment of type P_1 we set in *silva* a low timeout of 1 second per sample, so that the percentage of stability of some RFs with $B = 50$ is given as an interval which includes inconclusive stability checks. We first observe that as the forest grows in size, the difference between voting schemes becomes negligible, both in terms of time and stability. For smaller forests the average vote yields a slightly better stability. We also observe that forests trained using the entropy criterion are more stable than those trained using the Gini index. As expected, stability grows with the number B of trees. It is also worth noticing that for a given B, training criterion and voting scheme, the stability increases with the depth d up to a maximum $d = 25$ and then slowly starts to decrease. This is likely due to overfitting, so that deeper trees may improve the accuracy of a RF but, on the other hand, tend to reduce its generalizability. Based on this assessment, for the successive experiments we considered random forests trained with the entropy criterion and using the average score, which appears to be an optimal configuration for the stability metric.



The chart above depicts the relationship between stability and accuracy, where darker points represent random forests with lower depth d. The number B of trees of these RFs is not depicted in the chart and ranges in $\{5, 10, 15, 25, 50, 75\}$. The vertical bars represent intervals of stability due to inconclusive checks within the 1 second per sample timeout. We observe that RFs having the same maximum depth d tend to cluster together and that for RFs with $d \leq 10$ no clear interpretation of their stability can be derived, thus suggesting that RFs with $d \leq 10$ should not be considered for assessing the stability on MNIST. On the other hand, deeper forests with $d \geq 15$ tend to a vertical alignment, thus revealing a growing stability with comparable accuracy. Fixed a depth d, we may observe that both accuracy and stability grow with B. It is worth remarking that for a given accuracy, the highest stability is achieved by RFs with depth $d = 25$ rather than $d = 50$, thus hinting that to increase d beyond some threshold may not positively affect the accuracy while reducing the stability. This finding shows that by taking into account both accuracy and stability the overall effectiveness of a random forest model does not necessarily increases with its size. In particular, the 75×25 random forest turns out to be the most accurate and stable. The following table also displays the stability metrics defined in Section 3.

RF							
B	d	acc. %	stab. %	rob. %	frag. %	vuln. %	break. %
5	5	76.1	20.2	18.0	58.1	2.2	21.7
5	25	91.3	28.4	27.9	63.5	0.5	8.1
5	50	91.3	22.8	22.4	68.9	0.4	8.3
25	5	85.6	19.7	19.2	66.4	0.5	13.9
25	25	96.2	39.4	39.3	56.8	0.1	3.7
25	50	96.2	36.4	36.2	60.0	0.1	3.6
50	5	85.4	24.2	23.6	61.7	0.6	14.0
50	25	96.5	43.1 ±1.5	43.0 ±1.5	53.5 ±1.5	0.1	3.4
50	50	96.6	41.3 ±1.4	41.2 ±1.4	55.4 ±1.4	0.1	3.3

Here, it is worth remarking that: (i) stability and robustness are closely related; (ii) vulnerability decreases with the overall size $B \times d$ of the RF, and for RFs having size $\leq 50 \times 25$ we found a significant percentage of misclassified input samples ($\approx 0.4\%$) with adversarial attacks which are consistently classified with the same wrong label; (iii) breakage appears to converge towards $\approx 3.5\%$ for larger forests.

Verification Time per Sample. In a context of using *silva* as a complete verifier for checking whether a given input sample $\mathbf{x} \in X$ is robust against the adversarial perturbation $\mathcal{P}_{\infty,1}(\mathbf{x})$, the following table displays the average verification Time per Sample (TpS), the average verification Time for the Samples whose verification time is above the 90th percentile of the distribution (TpS10), the Maximum verification Time per Sample (MTpS). It is worth observing that the worst case verification time of *silva* never exceeds 1 minute and that the average verification time on the hardest input samples is always less than 5 seconds.

RF					GBDT				
B	d	TpS(s)	TpS10(s)	MTpS	B	d	TpS(s)	TpS10(s)	MTpS
25	5	0.00	0.00	0.04	50	10	0.00	0.00	0.01
25	10	0.00	0.01	0.60	75	5	0.00	0.00	0.02
50	5	0.07	0.66	34.69	75	10	0.00	0.01	0.32
50	10	0.44	4.36	42.25	100	10	0.01	0.15	32.99

Comparison with VoTE. The following table shows the results of the comparison of *silva* with VoTE, a recent robustness verifier (Törnblom and Nadjm-Tehrani 2019b), already discussed in Section 1. We replicated the experiments on the MNIST dataset as described in (Törnblom and Nadjm-Tehrani 2019b) and compared the results in terms of robustness and verification time (on our machine). Each experiment is run on RFs and GBDTs trained with the same parameters: RFs are trained using scikit-learn with Gini/average parameters, while GBDTs are trained by CatBoost with default learning rate and softmax voting scheme. We followed (Törnblom and Nadjm-Tehrani 2019b) by setting an overall timeout of 7 hours for the verification time of the full test set of MNIST.

RF		*silva*			VoTE		
B	d	acc.%	rob.%	time(s)	acc.%	rob.%	time(s)
25	5	85.7	16.3	4	84.5	13.6	8
25	10	94.1	24.2	20	94.1	25.7	11
50	5	85.8	17.8	576	86.1	14.2	833
50	10	94.4	30.2	4353	94.6	31.4	7704
75	5	86.2	19.8±0.4	8289	86.0	–	timeout

GBDT		*silva*			VoTE		
B	d	acc.%	rob.%	time(s)	acc.%	rob.%	time(s)
50	10	95.6	94.0	1	95.3	92.4	5
75	5	94.8	91.3	1	94.7	89.6	4
75	10	96.0	93.4	6	96.0	91.9	116
100	10	96.2	93.0	147	96.4	91.9	974
150	10	96.7	92.2±0.4	9459	96.7	–	timeout

The difference between *silva* and VoTE on the accuracies of the input RFs and GBDTs is negligible and due to

the randomness of the learning algorithm. The difference on the robustness ratios inferred by *silva* and VoTE may reach 3.6% (for the RF 50×5): the reasons why this happens are not clear and would require an in-depth inspection of the VoTE source code. Overall, it turns out that *silva* is faster than VoTE: on the larger forests verified by VoTE within 7 hours, *silva* achieves a speedup factor of ≈ 1.7 on the RF of size 50×10 and of ≈ 6.6 on the GBDT of size 100×10. The table also shows that by applying a timeout of 60 seconds per sample on the RF 75×5 and the GBDT 150×10, *silva* is able to output in at most 2.5 hours a rather precise estimate ($\pm 0.4\%$) of the robustness metric for these large RF and GBDT models where VoTE exceeds the 7 hours limit.

5.1 Sensorless Dataset

We also used *silva* on RFs and GBDTs trained on the Sensorless dataset from the UCI ML Repository, which has been recently used by Chen et al. (2019a) to test a robust learning method for GBDTs. Sensorless consists of 58509 vectors with 48 real features scaled in $[0, 1]$ and with 11 possible class labels, extracted as measures of electric current drive signals. The training set of Sensorless includes 48509 samples, while its test set T_{S} includes the remaining 10000 samples. We considered the perturbation $\mathcal{P}_{\infty,\epsilon}$ with $\epsilon = 0.01$, meaning that each component of an input vector can be perturbed $\pm 1\%$. We loosely followed GBDTs considered by (Chen et al. 2019a), which have been trained by XGBoost with $B \in [3, 30]$ and $d = 6$. The following tables summarize the results of our experiments obtained by running *silva* on a range of RFs and GBDTs trained on Sensorless, whose stability metrics and total verification times have been evaluated on the whole test set T_{S} (vulnerability is always $\leq 0.8\%$ and therefore omitted).

RF							
B	d	acc. %	stab. %	rob. %	frag. %	break. %	time(s)
20	5	89.8	27.9	27.7	62.1	10.0	2.89
20	10	99.6	22.9	22.9	76.7	0.4	19.85
20	15	99.9	20.3	20.3	79.5	0.1	26.23
20	20	99.8	17.9	17.9	82.0	0.2	21.01
25	5	89.5	31.3	31.0	58.5	10.2	6.23
25	10	99.6	24.4	24.4	75.1	0.4	62.90
25	15	99.9	22.2	22.2	77.6	0.1	116.50
25	20	99.9	18.9	18.9	80.8	0.1	85.35
30	5	90.0	33.6	33.1	57.0	9.4	19.68
30	10	99.7	26.1	26.1	73.6	0.3	213.37
30	15	99.9	23.8	23.8	76.0	0.1	357.25
30	20	99.9	20.2	20.2	79.6	0.1	532.34
35	5	89.8	35.2	34.4	55.4	9.4	48.05
35	10	99.7	26.2	26.2	73.4	0.3	832.80
35	15	99.9	23.9	23.9	75.9	0.1	1517.18
35	20	99.9	21.4	21.4	78.4	0.1	1459.36
GBDT							
15	5	97.0	19.4	18.9	78.0	2.6	6.57
15	6	98.0	29.8	29.5	68.5	1.7	8.77
15	7	98.1	17.4	17.3	80.8	1.8	12.58
20	5	97.7	15.6	15.3	82.5	2.0	36.01
20	6	98.5	29.3	29.0	69.5	1.2	109.04
20	7	98.5	14.5	14.4	84.1	1.5	108.14
25	5	98.1	15.1	14.9	83.3	1.6	142.14
25	6	98.8	22.8	22.6	76.1	1.0	835.84
25	7	98.9	14.8	14.7	84.1	1.1	8202.94
30	5	98.4	13.8	13.6	84.8	1.4	1031.43
30	6	98.9	20.2	20.1	78.8	0.9	6185.99
30	7	99.1	13.8	13.7	85.3	0.9	24699.30

RF					GBDT				
B	d	TpS(s)	TpS10(s)	MTpS	B	d	TpS(s)	TpS10(s)	MTpS
30	15	0.04	0.34	6.61	25	6	0.08	0.77	12.72
30	20	0.05	0.51	65.73	25	7	0.82	8.12	1354.88
35	15	0.15	1.47	68.11	30	6	0.62	5.94	93.74
35	20	0.15	1.41	68.42	30	7	2.47	24.43	828.23

From this set of experiments for the $\pm 1\%$ perturbation, the most significant observations are as follows: (i) RFs are more accurate, stable and faster to verify than GBDTs; (ii) stability of RFs decreases by increasing the depth d and increases by increasing B; stability of GBDTs decreases by increasing B and initially increases and reaches a maximum for $d = 6$ then decreases by increasing d; (iii) by combining accuracy and stability, the 35×10 RF appears to be an optimal model although the 20×15 RF already provides a very close performance, while the 20×6 and 25×6 GBDTs appears to be optimal; (iv) some input samples of GBDTs with $d = 7$ may be harder to verify and may reach a verification time of ≈ 23 minutes; it turned out that, resp., 99.79% and 99.05% of the samples in T_{S} for, resp., the 25×7 and 30×7 GBDTs are verified in less than 30s, while 0.08% and 0.26%, resp., required more than 300s (and less than, resp., 23m and 14m); in complete oblivious trees used in GBDTs trained by CatBoost the number of reachable leaves is significantly higher than in RFs and this could explain why some sporadic input samples of GBDTs are harder to verify.

6 Future Work

We believe that this work represents a step forward in applying formal verification methods to machine learning models, in particular a very well known program analysis technique such as abstract interpretation. As a benefit of this principled approach, we singled out the role of abstract interpretation for designing a complete verifier of robustness properties of decision tree classifiers. We envisage that more advanced techniques could be used for abstracting combinations of paths in different decision trees, as those successfully applied in static program analysis (e.g. trace partitioning (Rival and Yi 2020, Section 5.1)). We also plan to resort to abstract interpretation in order to train decision tree classifiers which are provably robust, namely, to apply abstract interpretation to training algorithms rather than to trained classification algorithms, similarly to the approach of (Mirman, Gehr, and Vechev 2018) for training provably robust neural networks.

Acknowledgments

The doctoral fellowship of Marco Zanella is funded by Fondazione Bruno Kessler (FBK), Trento, Italy. This work has been partially funded by the University of Padova, under the SID2018 project "Analysis of STatic Analyses (ASTA)" and by the Italian Ministry of Research MIUR, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)".

References

Anderson, G.; Pailoor, S.; Dillig, I.; and Chaudhuri, S. 2019. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proc. 40th ACM Conf. on Programming Language Design and Implementation (PLDI 2019)*, 731–744.

Andriushchenko, M., and Hein, M. 2019. Provably robust boosted decision stumps and trees against adversarial attacks. In *Proc. 33rd Annual Conf. on Neural Information Processing Systems (NeurIPS 2019)*.

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

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

Calzavara, S.; Lucchese, C.; Tolomei, G.; Abebe, S. A.; and Orlando, S. 2019. TREANT: Training evasion-aware decision trees. *Preprint arXiv:1907.01197*.

Calzavara, S.; Lucchese, C.; and Tolomei, G. 2019. Adversarial training of gradient-boosted decision trees. In *Proc. 28th ACM Int. Conf. on Information and Knowledge Management (CIKM 2019)*, 2429–2432.

Carlini, N., and Wagner, D. A. 2017. Towards evaluating the robustness of neural networks. In *Proc. of 2017 IEEE Symposium on Security and Privacy*, 39–57.

Chen, H.; Zhang, H.; Boning, D. S.; and Hsieh, C. 2019a. Robust decision trees against adversarial examples. In *Proc. 36th Int. Conf. on Machine Learning, (ICML 2019)*, 1122–1131.

Chen, H.; Zhang, H.; Si, S.; Li, Y.; Boning, D. S.; and Hsieh, C. 2019b. Robustness verification of tree-based models. In *Proc. 33rd Annual Conf. on Neural Information Processing Systems (NeurIPS 2019)*.

Cousot, P., and Cousot, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages (POPL 1977)*, 238–252. ACM.

Einziger, G.; Goldstein, M.; Sa'ar, Y.; and Segall, I. 2019. Verifying robustness of gradient boosted models. In *Proc. 33rd AAAI Conf. on Artificial Intelligence*, 2446–2453.

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

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. T. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Proc. 2018 IEEE Symposium on Security and Privacy*, 3–18.

Goodfellow, I.; McDaniel, P.; and Papernot, N. 2018. Making machine learning robust against adversarial inputs. *Commun. ACM* 61(7):56–66.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety verification of deep neural networks. In *Proc. Int. Conf. on Computer Aided Verification (CAV 2017)*, 3–29.

Kantchelian, A.; Tygar, J. D.; and Joseph, A. D. 2016. Evasion and hardening of tree ensemble classifiers. In *Proc. 33rd Int. Conf. on Machine Learning (ICML 2016)*, 2387–2396.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. Int. Conf. on Computer Aided Verification (CAV 2017)*, 97–117.

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2017. Adversarial machine learning at scale. In *Proc. 5th Int. Conf. on Learning Representations (ICLR 2017)*.

Miné, A. 2017. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages* 4(3-4):120–372.

Mirman, M.; Gehr, T.; and Vechev, M. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proc. Int. Conf. on Machine Learning (ICML 2018)*, 3575–3583.

Raghunathan, A.; Steinhardt, J.; and Liang, P. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Proc. Annual Conf. on Neural Information Processing Systems (NeurIPS 2018)*, 10900–10910.

Ranzato, F., and Zanella, M. 2019. *silva* GitHub Repository. https://github.com/abstract-machine-learning/silva.

Rival, X., and Yi, K. 2020. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. The MIT Press.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. T. 2018. Fast and effective robustness certification. In *Proc. Annual Conf. on Neural Information Processing Systems 2018 (NeurIPS 2018)*, 10825–10836.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3(POPL 2019):41:1–41:30.

Törnblom, J., and Nadjm-Tehrani, S. 2018. Formal verification of random forests in safety-critical applications. In *Proc. 6th Int. Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2018)*, 55–71. Springer.

Törnblom, J., and Nadjm-Tehrani, S. 2019a. An abstraction-refinement approach to formal verification of tree ensembles. In *Proc. 2nd Int. Workshop on Artificial Intelligence Safety Engineering, held with SAFECOMP*. Springer.

Törnblom, J., and Nadjm-Tehrani, S. 2019b. Formal verification of input-output mappings of tree ensembles. *Preprint arXiv:1905.04194*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient formal safety analysis of neural networks. In *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS 2018)*, 6369–6379.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal security analysis of neural networks using symbolic intervals. In *Proc. 27th USENIX Security Symposium*, 1599–1614.

Weng, T.; Zhang, H.; Chen, H.; Song, Z.; Hsieh, C.; Daniel, L.; Boning, D. S.; and Dhillon, I. S. 2018. Towards fast computation of certified robustness for ReLU networks. In *Proc. 35th Int. Conf. on Machine Learning, (ICML 2018)*, 5273–5282.

Yildiz, O. T., and Alpaydin, E. 2001. Omnivariate decision trees. *IEEE Trans. Neural Networks* 12(6):1539–1546.

Zhang, H.; Weng, T.; Chen, P.; Hsieh, C.; and Daniel, L. 2018. Efficient neural network robustness certification with general activation functions. In *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS 2018)*, 4944–4953.