An Abstraction-Refinement Approach to Formal Verification of Tree Ensembles

John Törnblom and Simin Nadjm-Tehrani

Dept. of Computer and Information Science
Linköping University, Linköping, Sweden
{john.tornblom,simin.nadjm-tehrani}@liu.se

Abstract. Recent advances in machine learning are now being considered for integration in safety-critical systems such as vehicles, medical equipment and critical infrastructure. However, organizations in these domains are currently unable to provide convincing arguments that systems integrating machine learning technologies are safe to operate in their intended environments.
In this paper, we present a formal verification method for tree ensembles that leverage an abstraction-refinement approach to counteract combinatorial explosion. We implemented the method as an extension to a tool named VoTE, and demonstrate its applicability by verifying the robustness against perturbations in random forests and gradient boosting machines in two case studies. Our abstraction-refinement based extension to VoTE improves the performance by several orders of magnitude, scaling to tree ensembles with up to 50 trees with depth 10, trained on high-dimensional data.

Keywords: Formal verification · Decision trees · Tree ensembles

1 Introduction

Machine learning technologies have enabled great progress in many domains in recent years, e.g. computer vision, anomaly detection, and automatic control. Manufactures of safety-critical systems such as vehicles, medical equipment and critical infrastructure are now considering integrating these advances in their products. However, safety-critical systems are often subject to strict regulations, and as such, require convincing arguments that the systems are safe to operate in their intended environments. Current industry standards often rely on software testing and human experts capable of identifying circumstance under which the software should (not) be tested. Unfortunately, these methods are often unsuitable when machine learning technologies have been used to develop software artifacts subject to verification.

Complementing software testing that relies on human experts who comprehend the internal structure of the software under test, formal verification techniques offer additional evidence for correctness. Most research is so far focused on formally verifying neural networks (see e.g. the survey by Liu et al. [9]), but there are other learning models that may be more appropriate when verifiability is important e.g. random forests [1], and gradient boosting machines [6].

Recent work by Törnblom and Nadjm-Tehrani [13] demonstrates that formal verification of tree ensembles trained on low-dimensional data is practical. However, the proposed method struggles with combinatorial explosion when tree ensembles are trained on high-dimensional data. In this paper, we address these shortcomings by extending that work with an abstraction-refinement approach that counteracts combinatorial explosion, and thus enables formal verification of tree ensembles trained on high-dimensional data. The contributions of this paper are as follows.

- A formal abstraction-refinement based verification method tailored specifically for tree-based ensembles.
- A realization[1] of the method, implemented as an extension to the toolsuite VoTE [13].
- Application of the method in two case-studies from current literature.

The rest of this paper is structured as follows. Section 2 presents a background on tree-based ensembles and the toolsuite VoTE which our implementation is based upon. Section 3 presents our abstraction-refinement technique, and how we realized it in VoTE. Section 4 presents applications of the method on two case studies; a collision detection problem, and a digit recognition problem. Section 5 discusses related works on verification of tree-based ensembles. Finally, Sec. 6 concludes the paper and summarizes the lessons learned.

2 Background

In this section, we present the required background on tree-based ensembles and the toolsuite VoTE. We also provide a definition of the classifier robustness property which we will verify in case studies in Sec. 4.

2.1 Decision Trees

In machine learning, decision trees are used as predictive models to capture statistical properties of a system of interest.

Definition 1 (Decision Tree). *A decision tree implements a prediction function $t : X^n \to \mathbb{R}^m$ that maps disjoint sets of points $X_i \subset X^n$ to a single output point $\bar{y}_i \in \mathbb{R}^m$, i.e.*

$$t(\bar{x}) = \begin{cases} (y_{1,1}, \dots, y_{1,m}) & \bar{x} \in X_1 \\ \vdots & \\ (y_{k,1}, \dots, y_{k,m}) & \bar{x} \in X_k, \end{cases} \tag{1}$$

where k is the number of disjoint sets and $X^n = \bigcup_{i=1}^{k} X_i$.

[1] Published at https://github.com/john-tornblom/VoTE/releases/tag/v0.2.1

The n-dimensional input domain X^n includes elements $\bar{x}$ as tuples in which each element x_i captures some feature of the system of interest as an input variable. Each internal node in the tree is associated with a decision function that separates points in the input space from each other, and the leaves define output values. The tree structure is evaluated in a top-down manner, where decision functions determine which path to take towards the leaves. When a leaf is hit, the output $\bar{y} \in \mathbb{R}^m$ associated with the leaf is emitted.

In general, decision functions are defined by non-linear combinations of several input variables at each internal node. In this paper, we only consider binary trees with linear decision functions with one input variable, which Irsoy et al. call univariate hard decision trees [7]. Although it has been demonstrated that non-linear [7] and multivariate decision trees [14] can be useful, state-of-the-art implementations of tree-based ensembles typically use univariate hard decision trees, e.g. scikit-learn [10] and CatBoost [11].

2.2 Random Forests

Decision trees are known to suffer from a phenomenon called overfitting. Models suffering from this phenomenon can be fitted so tightly to their training data that their performance on unseen data is reduced the more you train them. To counteract this issue with decision trees, Breiman [1] proposes random forests.

Definition 2 (Random Forest). *A random forest $f : X^n \to \mathbb{R}^m$ is an ensemble of B decision trees that produces outputs by averaging the values emitted by each individual tree, i.e.*

$$f(\bar{x}) = \frac{1}{B}\sum_{b=1}^{B} t_b(\bar{x}), \tag{2}$$

where t_b is the b-th tree in the ensemble.

To reduce correlation between trees, each tree is trained on a random subset of the training data, using potentially overlapping random subsets of the input variables.

2.3 Gradient Boosting Machines

Similarly, Freidman [6] introduces a machine learning model called gradient boosting machine that uses several decision trees to implement a prediction function. Unlike random forests, these trees are trained in a sequential manner. Each consecutive tree compensates for errors made by previous trees by estimating the gradient of errors (using gradient decent, hence the name). In a learning context, this is conceptually very different from random forests, but during prediction, these two models have many things in common.

Definition 3 (Gradient Boosting Machine). *A gradient boosting machine $f : X^n \to \mathbb{R}^m$ is an ensemble of B additive decision trees, i.e.*

$$f(\bar{x}) = \sum_{b=1}^{B} t_b(\bar{x}), \tag{3}$$

where t_b is the b-th tree in the ensemble.

2.4 Classifiers

Decision trees and tree ensembles may be used as classifiers. A classifier is a function that categorizes samples from an input domain into one or more classes and assigns each sample a label unique to its class. In this paper, we only consider functions that map each point from an input domain to exactly one class.

Definition 4 (Classifier). *Let $f(\bar{x}) = (y_1, \ldots, y_m)$ represent a model trained to predict the probability y_i associated with a class i within disjoint regions in the input domain, where m is the number of classes. A classifier $f_c(\bar{x})$ may then be defined as*

$$f_c(\bar{x}) = \operatorname*{argmax}_i y_i. \tag{4}$$

A random forest typically infers probabilities by capturing the number of times a particular class has been observed within some hyperrectangle in the input domain of a tree during training. Training a gradient boosting machine to predict class membership probabilities is somewhat different, and depends on the characteristics of the used learning algorithm, often involving post-processing the sum of all trees. For example, when training multiclass classifiers in CatBoost [11], individual trees emit values from a logarithmic domain that are summed up, and finally transformed and normalized into probabilities using the softmax function, i.e.

$$\text{softmax}(y_1, \ldots, y_m) = \frac{(e^{y_1}, \ldots, e^{y_m})}{\sum\limits_{i=1}^{m} y_i}. \tag{5}$$

2.5 Classifier Robustness

Bruneau et al. [2] describe robustness as the ability of a system of interest to withstand a given level of stress without suffering degradation or loss of function. In the context of machine learning, such a description includes a classifier's ability to maintain decisiveness in its predictions despite noisy or adversarial input. Formally, such an equivalence relationship between input and output may be defined as follows.

Definition 5 (Robustness against Perturbations). *Let $f_c : X^n \to L$ be the classifier subject to verification, $X_l \subset X^n$ a set of samples with label $l \in L$ where robustness against perturbations is desirable, $\epsilon \in \mathbb{R}_{\geq 0}$ a robustness margin, and $\Delta = \{\delta \in \mathbb{R} : -\epsilon < \delta < \epsilon\}$ perturbations. We denote by $\bar{\delta}$ a tuple of perturbations, i.e. an n-tuple of elements drawn from Δ. The classifier is robust against perturbations with respect to X_l and Δ iff*

$$\forall \bar{x} \in X_l,\ \ \forall \bar{\delta} \in \Delta^n, f_c(\bar{x}) = f_c(\bar{x} + \bar{\delta}) = l. \tag{6}$$

Note that this definition does not capture all possible input perturbations. Depending on the application, other equivalence relationships such as axial rotations may also be of interest, but are out of scope for this paper.

2.6 Verifier of Tree Ensembles

VoTE (Verifier of Tree Ensembles) [13] is a toolsuite for formally verifying that tree ensembles comply with requirements. The toolsuite implements two techniques, one approximate but conservative technique that bounds the output of a tree ensemble, and one precise and exhaustive technique that computes and enumerates equivalence classes in a tree ensemble, i.e. sets of points in the input domain that yield the same output tuple. The approximate technique has been used to verify e.g. that probabilities computed in a classifier are in the range $[0, 1]$, and the precise technique can be used to verify robustness.

As shown in Fig. 1, the toolsuite consists of two components, VoTE Core and VoTE Property Checker. VoTE Core is instantiated from the ensemble subject to verification, $f : X^n \to \mathbb{R}^m$. It takes as input a hyperrectangle defining X^n, and emits all equivalence classes in f, i.e. sets of points in the input space that yield the same output. These equivalence classes are then checked for compliance against a property $\mathbb{P}$ by a VoTE Property Checker.

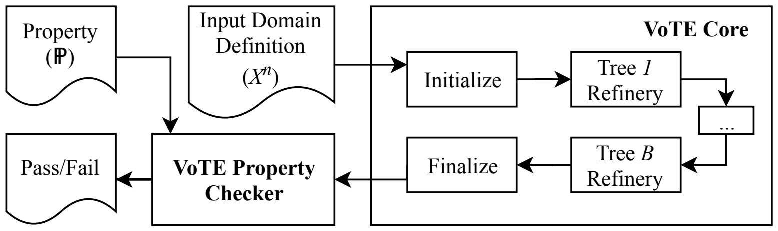

Fig. 1. The design of VoTE.

3 Abstractions and Refinements

In this section, we present our abstraction-refinement based verification approach that combines the two verification techniques mentioned in Sec. 2.6. The basic idea is to abstract multiple input-output mappings of a system subject to verification using the approximate technique, and then iteratively refine them using the precise technique.

3.1 Terminology

Requirements on systems considered in this paper may be expressed in terms of input-output mappings, expressions which we call *mapping specifications.*

Definition 6 (Mapping Specification). *Let X^n be the n-dimensional input domain of a system subject to specification, and $\mathbb{R}^m$ its m-dimensional output range. A mapping specification $\mathbb{P}$ is a set of pairs $(\bar{x}, \bar{y})$ where $\bar{x} \in X^n$ and $\bar{y} \in \mathbb{R}^m$, that specifies the expected input-output mappings of the system. More specifically, we expect that any implementation of the system maps $\bar{x}$ to $\bar{y}$.*

Verification of software with respect to a mapping specification may be carried out by means of exhaustive testing if the specification has a small enough cardinality. For large specifications, abstraction techniques may be used. Generally, an abstraction is a description that omits information that is irrelevant to the problem at hand. For example, classifier requirements are often only concerned with the most probable class in a prediction, in which case numerical probabilities and the order of less probable classes are irrelevant. To capture several input-output mappings with a single data structure, we use *abstract mappings.*

Definition 7 (Abstract Mapping). *An abstract mapping of a function* $f : X^n \to \mathbb{R}^m$ *is a pair of sets* (X_i, Y_a) *where* $X_i \subseteq X^n$ *denotes a precise input region, and* $Y_a \subseteq \mathbb{R}^m$ *is a conservative approximation of the output of* f *with respect to* X_i*, i.e.* $Y_a \supseteq \{f(\bar{x}) : \bar{x} \in X_i\}$.

Our goal is to systematically construct abstract mappings from an implementation of a system and then reason about the implementation's compliance with a mapping specification using a *mapping checker.*

Definition 8 (Mapping Checker). *Let* $\mathbb{P}$ *be a mapping specification,* (X_i, Y_a) *an abstract mapping of the tree ensemble* f *subject to verification, such that* $X_i \subseteq \{\bar{x} : (\bar{x}, \bar{y}) \in \mathbb{P}\}$*, and* $M_a = X_i \times Y_a$*. A mapping checker* C *checks the correctness of* f *with respect to* $\mathbb{P}$ *using* M_a *as follows:*

$$C(M_a) = \begin{cases} \textit{Pass} & M_a \subseteq \mathbb{P} \\ \textit{Fail} & M_a \nsubseteq \mathbb{P} \wedge Y_a \cap \{\bar{y} : (\bar{x}, \bar{y}) \in \mathbb{P}\} = \emptyset \\ \textit{Unsure} & \textit{otherwise.} \end{cases} \tag{7}$$

A mapping checker is unsure whenever an abstract mapping used together with the function provides an output set which is neither compliant with $\mathbb{P}$, nor falls completely outside $\mathbb{P}$. In that case, we call an abstraction *inconclusive* whenever the checker returns "Unsure". The abstract mapping must then be refined (as described in Sec. 3.2) to determine compliance with the mapping specification.

Example 1 (Robustness Checker). Let $f : X^n \to \mathbb{R}^m$ be a tree ensemble trained to predict probabilities associated with a classifier that shall assign the label l to samples in a set X_l, and M_a an abstract mapping of f according to Definition 8. A mapping checker for this verification problem may then be implemented as

$$C(M_a) = \begin{cases} \text{Pass} & \{l\} = L_a \\ \text{Fail} & l \notin L_a \\ \text{Unsure} & \text{otherwise,} \end{cases} \tag{8}$$

where $L_a = \{\text{argmax}\, \bar{y} : (\bar{x}, \bar{y}) \in M_a\}$.

3.2 Abstraction-Refinement Loop

Our formal verification approach may be described as an iterative process as illustrated by Fig. 2. Starting with an initialization step, an initial input region capturing the entire input domain is created. Next follows an abstraction step that, given an input region X_i, produces an output approximation Y_a from a set of trees T, thus forming an abstract mapping (X_i, Y_a). Next, the abstract mapping is evaluated by a mapping checker. If the abstract mapping is conclusive, the process is terminated and the final outcome is reported, i.e. "Pass" or "Fail".

If the abstract mapping is inconclusive, a refinement step removes an arbitrary tree t from T. The input region X_i is then split into k disjoint subsets $X_{i_1}, \ldots, X_{i_k}$ according to the decision functions in t, where k is the number of leaves in t. The succeeding iteration then produces abstract mappings from these subsets, i.e. $(X_{i_1}, Y_{a_1}), \ldots, (X_{i_k}, Y_{a_k})$, which again are evaluated by the mapping checker. When $T = \emptyset$, the abstraction-refinement loop is identical to the precise technique mentioned in Sec. 2.6, and all abstract mappings capture exactly one output tuple each (thus conclusive).

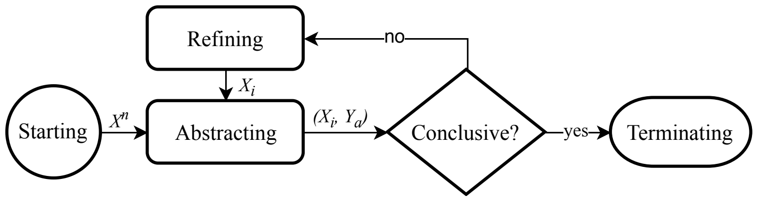

Fig. 2. Flowchart of our abstraction-refinement loop.

3.3 Implementation

We realize the abstraction-refinement loop in the toolsuite VoTE by extending its previous pipeline architecture with alternating abstraction and refining components, as illustrated by Fig. 3.

The first processing element in the pipeline constructs and initializes a hyperrectangle that captures the entire input domain. The final processing element executes a post-processing algorithm that is specific to a particular model. In the case of a random forest for example, the post-processing algorithm divides the sum of all tree outputs with the number of trees in the random forest.

In between, there is an alternating sequence of abstraction and refinery elements. An abstraction element takes as input a hyperrectangle capturing X_i, and computes a hyperrectangle Y_a (using the approximate technique mentioned in Sec. 2.6) that captures all values from all possible path combinations in a set of trees. The first abstraction element in the pipeline contains $B - 1$ trees, while the succeeding one contains $B - 2$ trees, and so on. If the abstraction (X_i, Y_a) is conclusive, no further refinement is necessary, and the outcome from the mapping checker is reported. If the abstraction is inconclusive, X_i is split into smaller input hyperrectangles by the succeeding refinery, and each new input hyperrectangle is transmitted to the succeeding abstraction element.

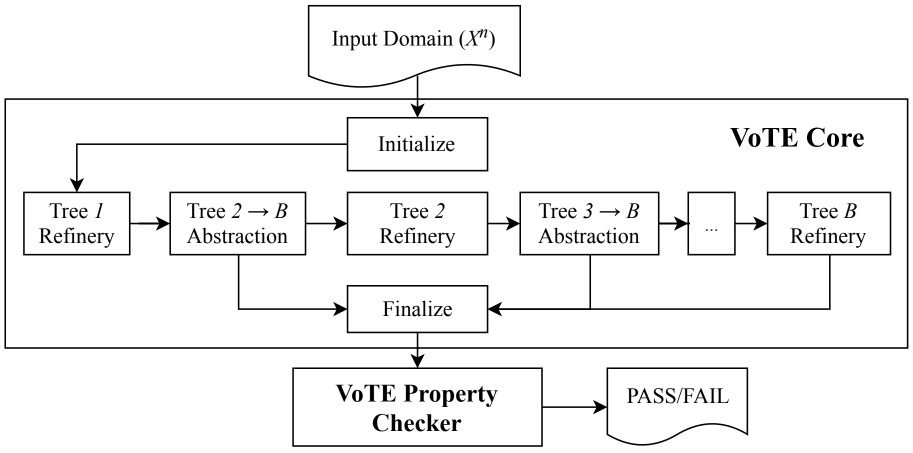

Fig. 3. Design of the abstraction-refinement extension to VoTE.

4 Case Studies

In this section, we evaluate our abstraction-refinement approach by verifying the robustness property in two case studies. Each case study defines a training set and a test set, and we used scikit-learn [10] to train random forests, and CatBoost [11] to train gradient boosting machines. Experiments were conducted on a machine with an Intel Core i5 2500K CPU and 16GB RAM. We also used a GeForce GTX 1050 GPU to speed up training of gradient boosting machines. For both case studies, we compare the outcome of the evaluation with an earlier method [13] as a baseline (VoTE without the abstraction-refinement loop).

4.1 Vehicle Collision Detection

In this case study, we verified tree ensembles trained to detect collisions between two moving vehicles traveling along curved trajectories. We used the same dataset used in an earlier study [13], which contains 30,000 training samples and 3,000 test samples generated by a simulation tool from Ehlers [4]. All samples are given in normalized form (position, speed, and direction are in the range $[0, 1]$, and rotation speed in the range $[-1, 1]$).

To keep comparability with the baseline, we defined input regions surrounding each sample in the test set with the robustness margin $\epsilon = 0.05$, which amounts to a 5% change since the data is normalized. Table 1 lists random forests (RF) and gradient boosting machines (GB) included in the experiment with their maximum tree depth d, number of trees B, accuracy on the test set (Accuracy), the percentage of samples from the test set where there were no misclassifications within the robustness region (Robustness), the elapsed time during verification (Time), and the elapsed time when using the baseline (Baseline).

Table 1. Performance impact of our abstraction-refinement approach in the vehicle collision detection case study.

Parameters		Accuracy (%)		Robustness (%)		Time (s)		Baseline (s)	
d	B	GB	RF	GB	RF	GB	RF	GB	RF
5	20	93.4	85.8	44.5	**65.6**	1	1	1	2
5	25	93.8	85.7	40.4	65.5	1	1	3	4
10	20	95.5	90.4	34.4	48.9	1	1	23	56
10	25	95.6	90.0	34.0	50.3	1	1	64	285
15	20	95.6	93.0	34.0	34.1	2	1	213	271
15	25	**96.0**	92.9	34.0	35.1	5	2	576	1637

When the tree depth was increased, accuracy increased, but robustness decreased. This suggests that the models were over-fitted with noiseless examples during training, and thus adding noisy examples to the training set may improve robustness. Compared to the baseline setup, our approach is several orders of magnitude faster.

4.2 Digit Recognition

The MNIST dataset [8] is a collection of hand-written digits commonly used to evaluate machine learning algorithms. The dataset contains 70,000 gray scale images with a resolution of 28x28 pixels at 8bpp. Each image is encoded as a tuple of 784 pixels, and the dataset was randomized and split into two subsets; a 85% training set, and a 15% test set.

We defined input regions surrounding each sample in the test set with the robustness margin $\epsilon = 1$, which amounts to a 0.5% lightning change per pixel in a 8bpp gray-scaled image. Due to scalability issues with the baseline setup, earlier work [13] had reduced the complexity of the high-dimensional problem by only considering all possible perturbations within a sliding window of 5x5 pixels. We apply the same complexity reduction technique in this case study to obtain comparable results.

Table 2 lists random forests (RF) and gradient boosting machines (GB) included in the experiment with their maximum tree depth d, number of trees B, accuracy of the test set (Accuracy), the percentage of samples from the test set where there were no misclassifications within the robustness region (Robustness), the elapsed time during verification (Time), and the elapsed time in our baseline setup (Baseline).

Our abstraction-refinement approach was particularly effective on random forests, demonstrating a speedup by several orders of magnitude. The baseline setup was unable to compute the robustness of large random forests within a reasonable amount of time, so we aborted long-running experiments after 7 hours (denoted by "-" entries in the table). With gradient boosting machines, the abstraction-refinement approach was consistently faster than the baseline setup, demonstrating speedup factors between 1.4–4.9 that increased with the size of the tree ensembles.

Table 2. Performance impact of our abstraction-refinement approach in the digit recognition case study where perturbations across a sliding window were considered.

Parameters		Accuracy (%)		Robustness (%)		Time (s)		Baseline (s)	
d	B	GB	RF	GB	RF	GB	RF	GB	RF
5	25	92.5	84.5	48.2	43.0	58	46	66	236
5	50	94.2	86.1	60.2	50.2	90	91	122	21041
5	75	94.4	85.9	60.8	54.7	127	137	191	-
10	25	94.7	94.2	66.0	74.8	63	55	107	1118
10	50	95.7	94.7	71.0	80.8	105	88	287	-
10	75	**95.9**	94.6	75.1	**82.2**	183	141	689	-

To explore the limitations of our approach, we reran the experiments without the baseline setup, and considered perturbations across the entire input domain (instead of sliding windows of 5x5 pixels). Table 3 lists the results in the same format as before.

Table 3. Accuracy, robustness, and elapsed verification time when using the abstraction-refinement approach in the digit recognition case study and considering perturbations across the entire input domain.

Parameters		Accuracy (%)		Robustness (%)		Time (s)	
d	B	GB	RF	GB	RF	GB	RF
5	25	92.5	84.5	8.5	13.6	70	7
5	50	94.2	86.1	12.1	14.2	316	851
5	75	94.4	85.9	9.7	-	13239	-
10	25	94.7	94.2	16.1	25.7	293	12
10	50	95.7	94.7	16.0	**31.4**	23292	7636
10	75	**95.9**	94.6	-	-	-	-

We note that the robustness of the learned system with respect to the larger set of possible perturbations is much lower (between 8–31%), which is somewhat expected. What is positive in the context is the fact that performing such analyses is at all possible considering the large search space (2^{784} possible perturbations).

During these experiments, we noticed that some images are harder to verify than others. In one of the more time consuming experiments, a single image accounted for 34% of the elapsed time. This suggests that evaluations of methods that verify robustness against perturbations need a significant amount of test samples to reveal the expected performance when collecting evidence for industrial-sized safety arguments.

5 Related Works

As mentioned earlier, this work is related to the work by Törnblom and Nadjm-Tehrani [13]. Specifically, in this paper we extend the tool VoTE with an abstraction-refinement scheme, and we use results from that paper as baselines in our evaluations.

Chen et al. [3] study the problem of training tree-based ensembles that are robust against adversarial attacks, and propose a technique to address the issue. They evaluate their technique by quantifying robustness against perturbations by means of testing, and demonstrate that their technique significantly improves robustness. In this paper, we take a formal approach that aims for a conclusive outcome compared to informal testing.

Recently, several researchers have pursued a formal approach to the verification of gradient boosting machines. Einziger et al. [5] verify the robustness of gradient boosting machines using an SMT solver. Similarly, Sato et al. [12] leverage an SMT solver, but address a regression problem in their case study, namely gradient boosting machines trained to predict continuous outputs. Due to significant differences in benchmarks and implementations of tree ensembles, we leave a systematic comparison between these three approaches for future works.

6 Conclusions and Future Works

Recent advances in machine learning are now being considered for integration in safety-critical systems. However, there is currently a lack of verification methods which yield convincing arguments that such systems are safe enough to operate.

In this paper, we presented an abstraction-refinement based approach to formal verification of tree-based machine learning models. We combined two verification techniques from related works [13], a conservative and fast approximation technique, and a precise and exhaustive technique. We realized the abstraction-refinement approach as an extension to the earlier toolsuite VoTE, and evaluated its performance impact on two case studies; a collision detection problem, and a digit recognition problem. Compared to previous work, our approach demonstrated speedups by several orders of magnitude.

In case studies addressed by this paper, we verified the robustness property using incomplete specifications. For example, the dataset in our digit recognition case study only contains 70,000 images, while the actual number of images that resemble a digit is enormous. The lack of complete formal specifications in applications where machine learning is useful is still an open research question, an issue we intend to address in future works.

As mentioned before, earlier work by Einziger et al. [5] and current work by Sato et al. [12] suggests that SMT solvers can verify gradient boosting machines. However, there are significant differences between test benches and implementations of tree ensembles used in their case studies, thus making a direct comparison difficult. Consequently, there is a need for plug-n-play benchmarks that can point towards fruitful future lines of research. Other potential lines of research based on this paper include a more strict formalization to enable formulating the decision procedure with soundness and completeness proofs, and a systematic analysis of abstraction and refinement criteria, e.g. the order in which to choose trees in the refinement steps.

Acknowledgements. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

1. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
2. Bruneau, M., Chang, S.E., Eguchi, R.T., Lee, G.C., O'Rourke, T.D., Reinhorn, A.M., Shinozuka, M., Tierney, K., Wallace, W.A., Von Winterfeldt, D.: A framework to quantitatively assess and enhance the seismic resilience of communities. Earthquake spectra **19**(4), 733–752 (2003)
3. Chen, H., Zhang, H., Boning, D., Hsieh, C.J.: Adversarial defense for tree-based models. Safe Machine Learning workshop at ICLR (2019)
4. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis (ATVA). Springer (2017)
5. Einziger, G., Goldstein, M., Sa'ar, Y., Segall, I.: Verifying robustness of gradient boosted models. In: AAAI Conference on Artificial Intelligence (2019)
6. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
7. Irsoy, O., Yildiz, O.T., Alpaydin, E.: Soft decision trees. In: International Conference on Pattern Recognition (ICPR). IEEE (2012)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
9. Liu, C., Arnon, T., Lazarus, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
10. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in Python. Journal of machine learning research **12**, 2825–2830 (2011)
11. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. In: Advances in Neural Information Processing Systems (NIPS) (2018)
12. Sato, N., Kuruma, H., Nakagawa, Y., Ogawa, H.: Formal verification of decision-tree ensemble model and detection of its violating-input-value ranges. arXiv preprint arXiv:1904.11753 (2019)
13. Törnblom, J., Nadjm-Tehrani, S.: Formal verification of input-output mappings of tree ensembles. arXiv preprint arXiv:1905.04194 (2019)
14. Wang, F., Wang, Q., Nie, F., Yu, W., Wang, R.: Efficient tree classifiers for large scale datasets. Neurocomputing **284** (2018)