Tree Ensemble Property Verification from A Testing Perspective

Bohao Wang[1,2], Zhe Hou[3], Gelin Zhang[2], Jianqi Shi[2], and Yanhong Huang* [1,2]

[1]Shanghai Key Laboratory of Trustworthy Computing, Shanghai, China
[2]National Trusted Embedded Software Engineering Technology Research Center,
East China Normal University, Shanghai, China,
{bohao.wang, gelin.zhang}@ntesec.ecnu.edu.cn,{yhhuang, jqshi}@sei.ecnu.edu.cn
[3]Griffith University, Australia, z.hou@griffith.edu.au

***Abstract*—With the development of artificial intelligence, machine learning algorithms are currently being used in more and more fields, such as autonomous driving, medical diagnosis, etc. In recent years, much research focuses on property verification of machine learning models. As one of the machine learning models, the tree ensemble model's structure is amicable to formal verification, but large models still prove hard to verify due to the combinatorial path explosion. This paper presents a violation-driven, sound but incomplete method from a testing perspective. We generate an explanation model of the original model and verify it formally. After a narrowed search space is obtained, we verify the original model by a testing-based method. A counterexample is then proof that the original model violates the property. We elaborate our method through a case study in detail. And we have developed our method into a tool called TEPV (Tree Ensemble Property Verification) and tested it on datasets of various sizes. The experiment demonstrates that our approach is scalable and works well on large tree ensemble models.**

***Index Terms*—Tree Ensemble, Property Verification, Testing**

I. Introduction

Nowadays, artificial intelligence utilizing machine learning algorithms has achieved a lot of success in many fields, such as face recognition, autonomous driving, medical diagnostics, etc. With the application of these technologies in more and more fields, people doubt whether they can meet certain properties, such as security, robustness, fairness etc., since deep learning models are almost a black-box, and the tree ensemble models are also too complex. In recent years, deep learning has developed rapidly, and now there has been much research on the verification of neural network [1], [2]. This paper is mainly focused on whether the tree ensemble models satisfy specific interesting properties.

Suppose you have a random forest model that tells you whether or not two cars will collide based on a variety of features. Traffic safety experts suggest that when the distance between two cars is less than 5 meters and both cars' speed is greater than 120km/h, the two cars are bound to collide. The prediction of a random forest model may be correct 99% of the time, but it may violate this property in some input spaces, so we still need to verify the random forest model.

As background work, we tried to develop and implement a general, sound and complete verification algorithm for random forest. Our method divided the input space of the random forest into different disjoint sets. Suppose the random forest has 10 trees, each tree has 32 leaf nodes (depth is 5), and each leaf node corresponds to a branch. We join each tree branch with one branch of all other trees to represent the disjoint sets of the input space divided by the random forest model. When the input space is divided into disjoint sets, the specific region of input space that does not satisfy the property will be found by a property checking algorithm. The method mentioned above faces combinatorial path explosions, which means it is not scalable and can only handle random forests where the sum of the number and depth are no more 15 in our experiment. Random forest models on this scale are toys that do not work in real life. Similar results are obtained in recent work on verifying tree ensembles [3].

Consequently, we take a step back and sacrifice completeness for a more feasible approach. The proposed approach in this paper performs verification of tree ensembles from a testing perspective. We first transform the original tree ensemble model into a relatively simple model, which is referred to as an "explanation model". The explanation model has a highly similar predictive behaviour compared to the original model [4]. Then we verify the explanation model against a property. Our hypothesis is that when the explanation model violates the property in a certain search space, it is highly likely that the original model also violates the property in the same search space. Then we can narrow down the search space using the explanation model and verify the original model in a much smaller input space by a test-based method. If a counterexample is found, then it is proved that the original model violates the property.

The contributions of this paper include:

- We propose a method for tree ensemble property verification, which alleviates combinatorial path explosion.
- We have developed a tool called TEPV that can perform property checking on tree ensemble models.

*Corresponding Author

DOI reference number: 10.18293/SEKE2021-087

- We validate our approach on datasets of different sizes, and the experiment demonstrates that our approach is scalable and works well on large tree ensemble models.

The rest of this paper is organized as follows: Section II gives the background knowledge of this paper. Section III details the proposed method. Section IV demonstrates a case study and experiment of our approach. We discuss related work in Section V. Finally, we conclude and give some future work in section VI.

II. Preliminaries

This section presents the required background knowledge, including decision trees, ensemble of decision trees, and the explanation tree ensemble model. We also give the definition of the properties used in this paper.

A. Decision Trees With a Logical Foundation

We adopt the definitions of decision tree described in [4]. In supervised learning, a structured dataset for classification is defined as set of *instances* of the form $(\vec{x}, y)$ where $\vec{x} = [x_1, ..., x_n]$, $n \in \mathbb{N}$, is an input vector called *features* and y is an outcome value often called the *label*. We denote by X the feature space and Y the outcome space.

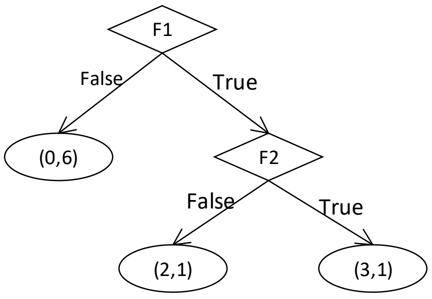

Fig. 1. An example decision tree.

A *decision tree* is composed of internal nodes (diamonds in Figure 1) and terminal nodes called leaves (ovals in Figure 1). Each internal node is associated with a logical formula over a feature. Each leaf node contains a set of instances, which yield a *vote distribution* of the form $(n_1, \cdots, n_m)$ where m is the number of classes and n_i $(1 \leq i \leq m)$ is the number of instances of the corresponding class. For example, in Figure 1, the leftmost leaf node $(0, 6)$ indicates that there are 0 `class1` instances and 6 `class2` instances. Without loss of generality, we focus on binary trees, in which internal nodes have two successors respectively called the left and right child nodes. By convention, the instances that satisfy the logical formula of an internal node go to the right child node, and those that do not satisfy go to the left child node. For example, in Figure 1, let I be the set of training instances associated with the root node, $I_1 \subset I$ be the subset that satisfies the formula F_1, then I_1 will be the set of instances associated with the right child node (with formula F_2), and $I_2 = I \setminus I_1$ will be the set of instances associated with the left child (leaf) node.

Given a decision tree, any input vector (or instance) is associated with a single leaf. A decision tree is, therefore, a compact representation of a function of the form $t : X \to \mathbb{N}^m$, where m is the number of classes. The output of a decision tree is a distribution of *votes* for each class. To obtain an outcome in Y, we take the class with the most votes.

B. Random Forest

We adopt the definitions of Cui et al. [5]. Let an ensemble be a set of decision trees of size T. It gives the weighted sum of the trees as follows:

$$E(x) = \sum_{i=1}^{T} w_i \cdot t_i(x) \tag{1}$$

where E is the function for the ensemble, w_i and t_i are respectively the weight and function for each tree. The summation aggregates the weighted votes from each tree and obtains the final votes for each class. Thus, the ensemble is also a function of the signature $E : X \to \mathbb{N}^m$ and requires a voting mechanism to obtain the outcome. We mainly focus on the ensemble trees by bagging. Each decision tree is trained using a subset of the dataset that is sampled uniformly with replacement. The remaining instances form the out-of-bag (OOB) set. When selecting the best formula at each decision node in a tree, only a subset of the features are considered. This is commonly found in algorithms such as Random Forest [6]. Bagging grows large trees with low bias, and the ensemble reduces variance.

C. Explanation Tree Ensemble

In our previous work [4], we extract logical formulas from an original tree ensemble model $\mathcal{M}_o$ to synthesize an explanation model $\mathcal{M}_e$. $\mathcal{M}_e$ is an approximation of $\mathcal{M}_o$, and it contains a set of decision rules. A decision rule is a tuple (F, s, w), where F is a classical logic formula, s is the *signature*, which is a normalized vote distribution between classes, and w is the *weight*, which indicates the importance of the rule. There are four parameters that determine the generation of the $\mathcal{M}_e$, which are θ, ϕ, ψ and k. θ determines the complexity of the decision rules, ψ determines the signature s of each decision rule. Both ϕ and k determine the number of the decision rule in $\mathcal{M}_e$. Refer to [4] for the details.

The explanation model $\mathcal{M}_e$ can make predictions based on the following method: given a data instance x, find all the decision rules in $\mathcal{M}_e$ whose logical formula F is satisfied by x, then multiply the signature of those rules by their corresponding weight, and add them up to get a tuple of vote distributions. The class with the largest value is the output. The above procedure is denoted as $\mathcal{M}_e(x) = c$, where c is a class. Figure 2 gives an example of how $\mathcal{M}_e$ makes a prediction.

D. Properties of Interest

In this paper, we consider a general class of properties, which is defined below.

Definition 1: (Properties) Let $f : X^n \to \mathbb{R}^m$ be the function to be verified. A *property* is of the form $\mathbb{P} : Constraint \to$

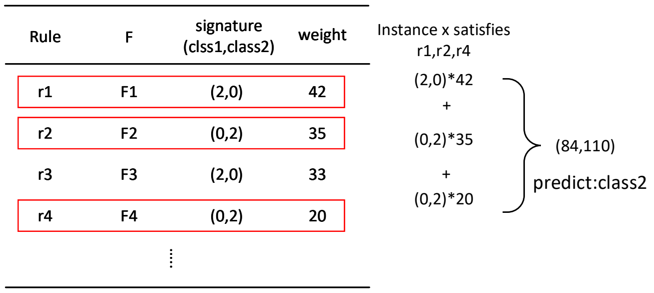

Fig. 2. An illustration of how $\mathcal{M}_e$ makes a prediction for an instance x.

$Target$, where $Constraint$ is the boundary of input features and $Target$ is the target label, i.e.,

$$Constraint ::= \bigwedge \alpha_i \leqslant x_i \leqslant \beta_i, \forall i \in \{1, ..., n\} \tag{2}$$

$$Target ::= (y = c_i), i \in \{1, ..., m\} \tag{3}$$

where $\alpha_i, \beta_i \in \mathbb{R}$ and c_i is the ith class of all m classes. If $\alpha_i = -\infty$ and $\beta_i = +\infty$, then x_i has no constraint. That is, $-\infty \le x_i \le +\infty \equiv \top$.

III. Methodology

This section details our tree ensemble property checking approach. The overview of our method is given in Figure 3. There are three stages in our method. First, we generate an appropriate explanation model $\mathcal{M}_e$, which has high fidelity. Here, *fidelity* refers to the degree of similarity between the predictions of $\mathcal{M}_e$ and $\mathcal{M}_o$ on unseen data [7]. The second stage is the ExModel Checker part. In this step, our approach checks whether the property $\mathbb{P}$ is violated by $\mathcal{M}_e$. If the ExModel Checker finds that $\mathbb{P}$ is violated, the process will continue. Finally, OrModel Checker will verify $\mathcal{M}_o$ against the property $\mathbb{P}$. If $\mathcal{M}_o$ violates $\mathbb{P}$, we will output a decision rule explanation and a counterexample. The counterexample is an instance that $\mathbb{P}$ is not satisfied by $\mathcal{M}_o$, and the decision rule explanation is a narrowed-down search space.

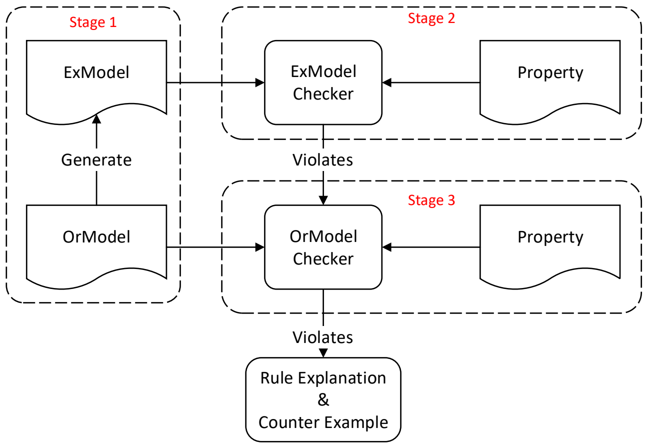

Fig. 3. An overview of the proposed method.

A. STAGE 1: Generate Explanation Model

We adopt our previous work [4] to generate the explanation model $\mathcal{M}_e$. The process of generating explanation model is expressed as the following formula:

$$explain(\mathcal{M}_o, \theta, \phi, \psi, k) = \mathcal{M}_e \tag{4}$$

A major difference compared to the previous work is that the $\mathcal{M}_e$ we generate here does not need to follow the criteria of *the explanation should be concise and small*. Instead, we only follow one criterion: *the classification behavior of the explanation model $\mathcal{M}_e$ should be as similar as possible to the original model $\mathcal{M}_o$.*

We mentioned in Section II-C that different parameters generate different $\mathcal{M}_e$. In order to get an explanation model $\mathcal{M}_e$ with high fidelity, we also use the *linearly decreasing inertia weight particle swarm optimization* algorithm (LDIW-PSO) [8] to optimise the four parameters:θ, ϕ, ψ and k. Since our criteria are different from [4], we propose a new equation below as the fitness, i.e., the objective function to be optimised.

$$S_{opt} = \sum_{i}^{n} \Theta(\mathcal{M}_e(x_i), \mathcal{M}_o(x_i)) \tag{5}$$

where $\Theta(x, y)$ is a function that outputs 1 when $x = y$; otherwise outputs 0, x_i is the sample in the test set, and n is the number of samples in the test set. After optimization, we finally obtain $\mathcal{M}_e^{opt}$.

B. STAGE 2:Verify Explanation Model

Once we obtain $\mathcal{M}_e^{opt}$, we can use the ExModel Checker to verify $\mathcal{M}_e^{opt}$ against properties. The detail of Stage 2 is shown in Algorithm 1.

Algorithm 1 Stage 2: find violation of explanation model

```
1: Input: Property $\mathbb{P}$, Explanation Model $\mathcal{M}_e^{opt}$
2: Output: a set of decision rules set $\Phi$
3: $R \leftarrow \emptyset$
4: $\Phi \leftarrow \emptyset$
5: for all $r \in \mathcal{M}_e^{opt}$ do
6:     if $r \wedge \mathbb{P}$.constraint is satisfiable then
7:         $R \leftarrow R \cup \{r\}$
8:     end if
9: end for
10: for all $R' \subseteq R$ do
11:     if $R'$ is satisfiable and $\mathcal{M}_e^{opt}.predict(R') \neq \mathbb{P}.target$ then
12:         $\Phi \leftarrow \Phi \cup \{R'\}$
13:     end if
14: end for
15: return $\Phi$
```

Given an $\mathcal{M}_e^{opt}$ and a property $\mathbb{P}$, the first step of checking $\mathcal{M}_e^{opt}$ is rule selection, corresponding to the lines 4-8 in Algorithm 1. For simplicity, the notation of $\mathcal{M}_e^{opt}$ not only refers to the explanation model but also the decision rules in the explanation model. For the decision rules in $\mathcal{M}_e^{opt}$ that are compatible with the constraint of property $\mathbb{P}$, we collect them as R. After rule selection, we try to find the subset of R whose decision rules are satisfiable when combined. Assume there is a set R' that meets the requirements; we use $\mathcal{M}_e^{opt}$ to make a prediction based on R'. We mentioned in Section II-C that the explanation model would first select decision rules in $\mathcal{M}_e^{opt}$ are satisfied by x, and then make a specific prediction according to the selected decision rules. Here, the prediction of $\mathcal{M}_e^{opt}$ omits the steps for selecting decision rules, as these are done in lines 4 - 8 already. Instead, we directly give the prediction based on the set R'.

If the prediction result is not consistent with the target of the property $\mathbb{P}$, $\mathcal{M}_e^{opt}$ violates the property $\mathbb{P}$. Then, we will continue to verify the original model $\mathcal{M}_o$ using OrModel Checker based on the narrowed search space.

C. STAGE 3: Verify Original Model

Because $\mathcal{M}_e^{opt}$ and $\mathcal{M}_o$ have very similar predictive behavior, when we look at the same (narrowed-down) search space, if $\mathcal{M}_e^{opt}$ violates the property $\mathbb{P}$, it is very likely that $\mathcal{M}_o$ may also violate $\mathbb{P}$. We present the details of stage 3 in Algorithm 2.

Algorithm 2 Stage 3: find violation sample of original model.

```
1: Input: Property $\mathbb{P}$, Original Model $\mathcal{M}_o$, decision rules set $R'$ and the number of generated samples $n$
2: Output: $flag$, $c$ and $e$
3: $flag \leftarrow False$
4: $c \leftarrow None$
5: $e \leftarrow R' \wedge \mathbb{P}.constraint$
6: $samples \leftarrow$ generate $n$ samples based on $e$
7: for all $s \in samples$ do
8:     if $\mathcal{M}_o.predict(s) \neq \mathbb{P}.target$ then
9:         $flag \leftarrow True$
10:        $c \leftarrow s$
11:        return $flag, c, e$
12:    end if
13: end for
14: return $False, None, None$
```

Note that the constraint component of a property has the same structure as the logical formula of a decision rule, so they can be used in conjunction to limit the search space. More specifically, we combine the decision rules in R' and the property $\mathbb{P}$'s constraint and simplify them by merging subformulae on the same feature (Line 5 of Algorithm 2). At this point, the counterexample search space of property $\mathbb{P}$ is reduced significantly by the above simplified formula, which we call *decision rule explanation*. Then, a sample generator will generate many samples in the narrow search space.

We adopt a Generative Adversarial Network (GAN) [9] model as the sample generator. GAN has two parts: a discriminative network (the discriminator) and a generative network (the generator). The discriminator learns to distinguish the features of a given data instance. The generator, which learns to confuse the discriminator, can generate fake but plausible data via an adversarial learning process. GAN will generate samples with the same distribution as the training set, but we need samples within the narrow search space determined by e. So we add a filter after the GAN model to select samples that meet the requirements. The process mentioned above corresponds to Line 6 of Algorithm 2.

Next, we use $\mathcal{M}_o$ to make predictions from the generated samples, which are expected to have a high chance of violating $\mathbb{P}$. If the predicted result is indeed different from the target of the property, it means that $\mathcal{M}_o$ violates the property. The sample with the different predicted result is provided as the counterexample, and the final narrow search space is output as a decision rule explanation.

Noted that there may be a lot of R' that cause property violation of $\mathcal{M}_e^{opt}$ in the subset of R, which means a lot of decision rules sets will be emitted in stage2, but what we want to find is a R'' that cause property violation of both two models. Since our approach is violation-driven, once we find one appropriate R'', our method will stop and return the counterexample and decision rule explanation. Otherwise, the method will continue to find the violation.

IV. EXPERIMENT AND CASE STUDY

We implement our method in Python and develop a tool called TEPV (Tree Ensemble Property Verification). We use scikit-learn [10] to train random forest models and evaluate our tool TEPV through a case study and experiment. For random forest in our experiment, the number of trees is 100, and the depth of the tree is *unlimited*. For simplicity, we only consider binary classification datasets with numeric features. However, the proposed approach can easily be extended to multi-class datasets with both numeric and nominal features. We test our method on three datasets: diabetes, JM1 and MiniBooNE, all of which are available on OpenML [11]. Particle size and iteration period are set to 20 by default, which are two parameters in LDIW-PSO. Experiments were conducted on a machine with an Intel Core i9-7960X CPU and 32GB RAM.

TABLE I
THE CHANGE OF SEARCH SPACE.

	Constraint of the Property		Rule Explanation	
Feature	lower bound	upper bound	lower bound	upper bound
preg	7.51	10.65	7.51	9.5
plas	138.52	$+\infty$	157.5	$+\infty$
pres	99.02	101.23	99.02	101.23
insu	77.94	93.51	77.94	93.51
skin	$-\infty$	99.04	$-\infty$	99.04
mass	26.94	60.39	27.9	28.9
pedi	0.63	2.23	0.63	1.1
age	$-\infty$	$+\infty$	$-\infty$	27.5

A. Case Study 1:Pima Indians Diabetes Dataset

The Pima Indians Diabetes dataset [12] has 768 samples, 8 features, 2 classes. The features are as follows: the number of times pregnant (preg), plasma glucose concentration (plas), diastolic blood pressure (pres), 2hour serum insulin (insu), triceps skinfold thickness (skin), body mass index (mass), diabetes pedigree function (pedi) and age. Two classes are

TABLE II
ALL DECISION RULES OF $\mathcal{M}_e^{opt}$.

Rule	Logic Formula	Signature (N, P)	Weight	Rule	Logic Formula	Signature (N, P)	Weight
r1	$(plas \leq 132.0) \bigwedge (mass \leq 26.4)$	(2,0)	133	r13	$(preg \leq 9.5) \bigwedge (plas > 157.5) \bigwedge (mass > 27.9)$	(0,2)	60
r2	$(mass \leq 28.9)$	(2,0)	127	r14	$(plas > 154.5) \bigwedge (pres \leq 92.0) \bigwedge (pedi > 0.3)$	(0,2)	52
r3	$(plas \leq 123.0)$	(2,0)	124	r15	$(plas > 154.5) \bigwedge (mass > 29.9) \bigwedge (age \leq 49.5)$	(0,2)	47
r4	$(plas \leq 128.5) \bigwedge (mass \leq 30.9)$	(2,0)	122	r16	$(plas > 154.5) \bigwedge (pedi \leq 1.1)$	(0,2)	46
r5	$(plas \leq 130.5) \bigwedge (insu \leq 30.5)$	(2,0)	116	r17	$(plas > 155.0) \bigwedge (pres \leq 92.0) \bigwedge (mass > 30.1)$	(0,2)	44
r6	$(plas \leq 104.0)$	(2,0)	116	r18	$(plas > 157.5) \bigwedge (mass > 28.8)$	(0,2)	43
r7	$(plas \leq 112.5)$	(2,0)	115	r19	$(plas > 147.5) \bigwedge (pedi > 0.3)$	(0,2)	41
r8	$(plas \leq 123.5)$	(2,0)	115	r20	$(plas > 139.0) \bigwedge (pedi > 0.2)$	(0,2)	40
r9	$(plas \leq 112.5)$	(2,0)	114	r21	$(plas > 132.0) \bigwedge (mass > 35.5)$	(0,2)	38
r10	$(age \leq 27.5)$	(2,0)	113	r22	$(preg > 7.5)$	(0,2)	36
r11	$(plas \leq 124.5) \bigwedge (mass \leq 27.8)$	(2,0)	112	r23	$(preg > 6.5) \bigwedge (insu > 22.5)$	(0,2)	36
r12	$(mass \leq 30.1)$	(2,0)	111	r24	$(plas > 129.5) \bigwedge (insu > 16.5) \bigwedge (skin \leq 629.5)$	(0,2)	36

positive and negative. We use 668 samples to train a random forest and 100 samples as the testing set.

In this case study, a doctor wants to check if the random forest model $\mathcal{M}_o$ satisfies his expert knowledge. He puts forward a property as follows. When a patient meets the constraint that preg is between 7.51 and 10.65, plas is bigger than 138.52, pres is between 99.02 and 101.23, insu is between 77.94 and 93.51, skin is less than 99.04, mass is between 26.94 and 60.39, and pedi is between 0.63 and 2.23, she should be diagnosed as positive. The doctor argues that the machine learning model is unreliable if it violates this property which is derived from his expertise.

We use our tool TEPV to verify the random forest model $\mathcal{M}_o$ against the property put forward by the doctor. We generate explanation model $\mathcal{M}_e^{opt}$ as Table II shows. And the decision rules in red is one set of the output of Algorithm 1. Based on Algorithm 1 and Algorithm 2, we have narrowed the search space and provide a counterexample of violation. In Table I, compared to the constraint of the tested property $\mathbb{P}$, it is clear that the upper bound of preg, mass, pedi and age shrinks; in the meantime, the lower bound of plas and mass rises.

TABLE III
A COUNTER EXAMPLE OF THE TESTED PROPERTY

preg	plas	pres	insu	skin	mass	pedi	age	predict
8	157.64	99.99	86.62	47.19	28.61	0.66	21	negative

The counterexample that reflects the violation of $\mathcal{M}_o$ is shown in Table III. This sample is generated by our sample generator in the input space defined by the decision rule explanation. Our method narrows down the search space, and the counterexample found in this space reflects the effectiveness of our approach.

B. Experiment

Under normal circumstances, users want to test whether the model violates certain user-specified properties that may be domain-specified. Since we are not domain experts and can not define properties with professional knowledge, we use the following method to randomly generate properties.

Assume the dataset has n features and m classes, we first find the min_i and max_i of feature f_i, $i \in \{1,..,n\}$, which is the minimum and maximum of the f_i in the training samples. For the constraint of the property, we select f_i with a chance of $p1\%$. And for each f_i selected, randomly generate α_i and β_i between min_i and max_i(assume $\alpha_i < \beta_i$). Then we generate the constraint $\alpha_i \leq f_i \leq \beta_i$ with a chance of $p2\%$, otherwise generate the constraint $\alpha_i \leq f_i$ or $f_i \leq \beta_i$ with 50% change each. For the target of the property, we randomly select one class as the target with equal probability.

We test on three datasets: diabetes, JM1 and MiniBooNE. JM1 is a dataset about software defect prediction; it has 21 features and 10k samples. MiniBooNE dataset is used to distinguish electron neutrinos (signal) from muon neutrinos; it has 50 features and 130k samples. For each dataset, we train a random forest model with 100 trees of unlimited depth, which are the usual default settings in real-life applications. We set $p1 \in \{0.5, 0.8\}, p2 \in \{0.5, 0.8\}$ and randomly generate 25 properties of every $(p1, p2)$ combination, totalling 100 properties for each dataset. The number of samples generated by the sample generator is 1000. For JM1 and MiniBooNE, 90% of the samples are used as the training set, and 10% of the samples are used as the testing set. For each property, if no counterexample is found or if the process takes more than one hour, we consider it a failure and the property is skipped, and the program begins to verify the next property.

TABLE IV
PROPERTIES CHECKING ON DIFFERENT DATASETS.

Dataset	$\mathcal{M}_e^{opt}$ Fidelity Parameters (θ, ϕ, ψ, k)	Acc.	Vio.	Time	Std.
Diabetes	93% $(0.57, 0.1, 0.67, 12.0)$	80%	75%	388	1424
JM1	92% $(0.66, 0.14, 0.7, 12.0)$	92%	60%	354	600
MiniBooNE	88% $(0.8, 0.04, 0.93, 20.0)$	93%	58%	524	752

The results of the experiment are shown in Table IV. $\mathcal{M}_e^{opt} Fidelity$ is the proportion that $\mathcal{M}_e^{opt}$ and $\mathcal{M}_o$ predict the same results in the test set. $Acc.$ is the prediction accuracy of $\mathcal{M}_o$ on the test set. $Vio.$ is the percentage of the properties for which we can find a counterexample within timeout, $Time$ is the average time of finding counterexamples (in seconds), and $Std.$ is the standard deviation of the time. The results show that our tool, TFPV, can find some violation of tree ensemble in different datasets. When the model violates properties, the average time required is less than 20 minutes. Since the properties are generated randomly, a large standard deviation is acceptable. But it is clear that our approach is scalable and works well on datasets of different sizes and large tree ensemble models.

V. Related Work

Since machine learning algorithms are used more and more frequently in daily life, people have some doubts about whether they meet certain requirements or properties. Therefore, more and more researchers are now working on the property verification of machine learning models. In the following, we will introduce the property verification of machine learning models in two parts: verification of deep learning models and verification of tree ensemble models.

A. Verification of Deep Learning Models

In 2010, Pulina et al. [2] proposes abstract interpretation to verify a DNN and introduces a linear approximation algorithm to estimate the interval of ReLU and Sigmoid output.

Fichetti et al. [13] propose a 0-1 MILP encoding to model a DNN for property verification and reasons through a MILP solver and implements a bound tightening mechanism to reduce the search space.

A method for verification of feed-forward neural networks with piecewise linear activation function was presented in [14]. They treat the neural network model as a block-box and use the SMT solver to verify the approximation of the block-box. Compared with Ehlers, Huang et al. [1] describe a white-box approach to verify the feed-forward neural networks and introduce a feed-forward analysis that partially based on discretization to test robustness and find adversarial examples.

B. Verification of Tree Ensemble Models

Tree ensemble models are non-continuous step functions, which is different from neural networks in deep learning. Therefore, the techniques mentioned above cannot be used to verification of tree ensemble models. Recently several researchers have pursued approaches to the verification of tree ensemble models.

Tornblem et al. [3] proposed a robustness verification tool of tree ensembles called VoTE. Their method is an abstraction-refinement procedure that iteratively refines a partition of the input space where each block of the partition is a hyperrectangle. A tool named Silva introduced by Ranzato and Zanella [15] pushes forward the 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. Unlike tool VoTE, the soundness and completeness properties of Tornblem's verification algorithm are not formally proved. The algorithm of Silva is based on the principles of abstract interpretation, which is endowed with a formal soundness and completeness proof.

VI. Conclusion and Future Work

This paper presents a method for verifying whether a tree ensemble model violates a user-specific property. We give one case study and show that our approach works. Moreover, in the experiment, we test various properties on datasets of different scale, which reflects the effectiveness of our method. In addition, the number of trees in the tree ensemble we tested is 100, and the depth is unlimited. By contrast, related methods struggle to verify 25 trees of depth 20 [3]. This demonstrates that our approach is scalable. Our approach relies heavily on sample generation, and in some certain cases, the performance of our tool, TEPV, may degrade dramatically. As future work, we plan to improve the sample generation algorithm so that our method can produce samples that meet the requirements more quickly and improve the speed of finding counterexamples.

References

[1] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International conference on computer aided verification*. Springer, 2017, pp. 3–29.

[2] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[3] J. Törnblom and S. Nadjm-Tehrani, "Formal verification of input-output mappings of tree ensembles," *Science of Computer Programming*, vol. 194, p. 102450, 2020.

[4] G. Zhang, Z. Hou, Y. Huang, J. Shi, H. Bride, J. S. Dong, and Y. Gao, "Extracting optimal explanations for ensemble trees via logical reasoning," *arXiv preprint arXiv:2103.02191*, 2021.

[5] Z. Cui, W. Chen, Y. He, and Y. Chen, "Optimal action extraction for random forests and boosted trees," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 179–188.

[6] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[7] A. Papenmeier, G. Englebienne, and C. Seifert, "How model accuracy and explanation fidelity influence user trust," *arXiv preprint arXiv:1907.12652*, 2019.

[8] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[11] OpenML, "openml.org," https://www.openml.org, Accessed 2019.

[12] P. D. Turney, "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm," *Journal of artificial intelligence research*, vol. 2, pp. 369–409, 1994.

[13] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, 2018.

[14] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 269–286.

[15] F. Ranzato and M. Zanella, "Abstract interpretation of decision tree ensemble classifiers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5478–5486.