Formal Verification of Random Forests in Safety-Critical Applications

John Törnblom and Simin Nadjm-Tehrani

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

Abstract. Recent advances in machine learning and artificial intelligence are now being applied in safety-critical autonomous systems where software defects may cause severe harm to humans and the environment. Design organizations in these domains are currently unable to provide convincing arguments that systems using complex software implemented using machine learning algorithms are safe and correct.
In this paper, we present an efficient method to extract equivalence classes from decision trees and random forests, and to formally verify that their input/output mappings comply with requirements. We implement the method in our tool VoRF (Verifier of Random Forests), and evaluate its scalability on two case studies found in the literature. We demonstrate that our method is practical for random forests trained on low-dimensional data with up to 25 decision trees, each with a tree depth of 20. Our work also demonstrates the limitations of the method with high-dimensional data and touches upon the trade-off between large number of trees and time taken for verification.

Keywords: Machine learning · Formal verification · Random forest · Decision tree

1 Introduction

In recent years, artificial intelligence utilizing machine learning algorithms has begun to outperform humans at several tasks, e.g. playing complex board games [21] and diagnosing skin cancer [8]. These advances are now being applied in safety-critical autonomous systems where software defects may cause severe harm to humans and the environment, e.g airborne collision avoidance systems [11].

Several researchers have raised concerns [4, 13, 18] regarding the lack of verification methods for these kinds of systems in which machine learning algorithms are used to train software deployed in the system. Machine learning models with large sets of parameters are hard to interpret. Humans are currently unable to provide convincing arguments that data used to test and train these models is sufficient, and exhaustive testing is generally intractable.

Instead, various formal methods have been suggested and evaluated. Most research is so far focused on the verification of neural networks, but there are

other models that may be more appropriate when verifiability is important, e.g. decision trees [2] and random forests [3]. Their structural simplicity makes them easy to analyze systematically, but large (yet simple) models may still prove hard to verify due to combinatorial explosion.

In this paper, we present a method to efficiently search for violations against interesting properties in random forests. There may be many such properties, some impacting system safety. We implement the method in our tool VoRF (Verifier of Random Forests), and evaluate the tool on two case studies found in the literature. The contributions of this paper are as follows.

- An efficient method to partition the input domain of decision trees into disjoint sets, and explore all path combinations in a random forest in such a way that counteracts combinatorial path explosions.
- A tool named VoRF to support the method.
- Application of the method to two case studies from earlier works.

The rest of this paper is structured as follows. Section 2 presents preliminaries on decision trees, random forests, and a couple of interesting properties. Section 3 discusses related works on formal methods and machine learning, and Section 4 presents our method with our supporting tool VoRF to verify properties of decision trees and random forests. Section 5 presents applications of our method on two case studies; a collision detection problem, and a digit recognition problem. Finally, Section 6 concludes the paper and summarizes the lessons we learned.

2 Preliminaries

Government agencies from several countries have agreed upon guidelines [5, 10] to help design organizations from different industries with assuring quality in software with safety-critical applications. Several methods described in these guidelines rely on human experts to analyze the software. However, manually analyzing large and complex software authored by machine learning algorithms is hard.

Recently, the avionics community published guidelines [6] describing how design organizations may apply formal methods to the verification of safety-critical software. Applying formal methods to complex and safety-critical software is a non-trivial task due to practical limitations in computing power, and challenges in qualifying complex verification tools. These challenges are often caused by a high expressiveness provided by the language in which the software is defined in. In this paper, we address these challenges by selecting machine learning models based on their simplicity rather than their expressiveness. Specifically, we develop a method with supporting tool to analyze decision trees and random forests.

2.1 Decision Trees and Random Forests

A decision tree implements a function $t : X^n \to \mathbb{R}^m$ using a tree structure where each internal node is associated with a decision function, and the leaves

define output values. The n-dimensional input domain X^n includes elements $\boldsymbol{x}$ as tuples where each element x_i captures some feature of the application as an input variable. 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 [9].

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 $\boldsymbol{y} \in \mathbb{R}^m$ associated with the leaf is emitted. Assuming a perfectly balanced binary tree, the number of leaves in a tree is 2^d, where d is the tree depth. Figure 1 depicts a univariate hard decision tree with one decision function ($x \leq 0$) and two outputs (1 and 2).

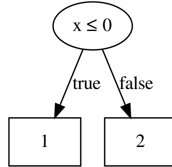

Fig. 1. A decision tree with two possible outputs, depending on the value of single variable x.

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 these effects in decision trees, Breiman [3] propose random forests.

Definition 1 (Random Forest). *A random forest $f : X^n \to \mathbb{R}^m$ is a collection of B decision trees that are combined by averaging the values emitted by each individual tree, i.e.*

$$f(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} t_b(\boldsymbol{x})$$

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

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

Decision trees and random forests may also be used as classifiers. A classifier is a function that categorizes samples from an input domain into one or more classes. In this paper, we only consider one-class classifiers, i.e. functions that map each point from an input domain to exactly one class.

Definition 2 (Classifier). *Let $f(\boldsymbol{x}) = (y_1, \ldots, y_m)$ be a model trained to predict the probability y_i of encountering a class i within disjoint regions in the input domain, where m is the number of classes. Then we would expect that $\forall i \in \{1, \ldots, m\}, 0 \leq y_i \leq 1$, and $\sum_{i=1}^{m} y_i = 1$. A classifier $f_c(\boldsymbol{x})$ may then be defined as*

$$f_c(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}\, y_i.$$

2.2 Safety Properties

In this paper, we consider two properties commonly used in related works; global safety [17], and robustness against noise. Note that compliance with these two properties alone is generally not sufficient to ensure safety. Moreover, the notions used here as an illustration are from AI papers. System safety engineers typically define requirements on software functions that are richer than these properties alone. Hence, global safety may be a misnomer in that context, but we simply repeat it here to be consistent with the literature that we refer to.

Property 1 (Global safety). Let $f : X^n \rightarrow \mathbb{R}^m$ be the function subject to verification. The function is globally safe if and only if

$$\forall \boldsymbol{x} \in X^n, \forall i \in \{1, \ldots, m\}, f(\boldsymbol{x}) = (y_1, \ldots, y_m), \quad \alpha_i \leq y_i \leq \beta_i.$$

for some $\alpha_i, \beta_i \in \mathbb{R}$.

In classification problems, the output tuple $(y_1, \ldots, y_m)$ contains probabilities, and thus $\alpha_i = 0$ and $\beta_i = 1$.

Property 2 (Robustness against noise). Let $f : X^n \rightarrow \mathbb{R}^m$ be the function subject to verification, $\epsilon \in \mathbb{R}_{\geq 0}$ a robustness margin, and $\Delta = \{\delta \in \mathbb{R} : -\epsilon < \delta < \epsilon\}$ noise. We denote by $\boldsymbol{\delta}$ an n-tuple of elements drawn from Δ. The function is robust against noise iff

$$\forall \boldsymbol{x} \in X^n, \ \forall \boldsymbol{\delta} \in \Delta^n, f(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{\delta}).$$

Pulina and Tacchella [17] define a stability property that is similar to our notion of robustness here but use scalar noise.

3 Related Works

Due to the extreme progress made in the application of machine learning in artificial intelligence, awareness regarding its (lack of) security and safety have increased. Researchers from several fields are now addressing these problems in their own way, often in collaboration between fields [20].

There have been extensive research on formal verification of neural networks. Pulina and Tacchella [17] combine SMT solvers with an abstraction-refinement

technique to analyze neural networks with non-linear activation functions. They conclude that formal verification of realistically sized networks is still an open challenge. Scheibler et al. [19] use bounded model checking to verify a non-linear neural network controlling an inverted pendulum. They encode the neural network and differential equations of the system as an SMT formula, and try to verify properties without success. These works [17, 19] suggest that SMT solvers are currently unable to verify realistic non-linear neural networks.

Recent research focuses on piece-wise linear neural networks. Katz et al. [12] combine the simplex method with a SAT solver to verify properties of deep neural networks with piecewise linear activation functions. They successfully verify domain-specific safety properties of a prototype airborne collision avoidance system trained using reinforcement learning. The verified neural network contains a total of 300 nodes organized into 6 layers. Ehlers [7] combines an ILP solver with a modified SAT solver to verify neural networks. His method includes a technique to approximate the overall behavior of the network to reduce the search space for the SAT solver. The method is evaluated on two case studies; a collision detection problem, and a digit recognition problem. We reuse these two case studies in our work, and also provide a global approximation of the overall model (in our cases, random forests).

Mirman et al. [15] use abstract interpretation to verify robustness of neural networks with convolution and fully connected layers. They evaluate their method on four image classification problems (one of which we use in our work), and demonstrate promising performance. In our work, we address similar verification problems, but for random forests. Since decision trees and random forests are generally easier to analyze systematically than neural networks, we expect that formal verification methods scale better when applied to decision trees and random forest compared to neural networks. More importantly, the simplicity of our method allows implementations such as VoRF to be certified for online use in safety-critical applications.

The fact that decision trees may be easier to verify than neural networks is demonstrated by Bastani et al. [1]. They train a neural network to play the game Pong, then extract a decision tree policy from the trained neural network. The extracted tree is significantly easier to verify than the neural network, which they demonstrate by formally verifying properties within seconds using an of-the-shelf SMT solver. Our method provides even greater performance when verifying decision trees. However, our outlook is that decision trees per se may not be sufficient for problems in non-trivial settings and hence we address random forests which provides a counter-measure to overfitting.

4 Analyzing Random Forests

In this section, we define a process for verifying learning-based systems, and define a formal method capable of verifying properties of decision trees and random forests. We also describe VoRF (Verifier of Random Forests), our implementa-

tion of our method, and provide an example on how to define and verify the global safety property of random forest classifiers using VoRF.

4.1 Problem Definition

We formulate the software verification process for learning-based systems using the following problem definitions.

Problem 1 (Constraint Satisfaction). Let $f : X^n \rightarrow \mathbb{R}^m$ be a function that is known to implement some desirable behavior in a system, and a property $\mathbb{P}$ specifying additional constraints on the relationship between $\boldsymbol{x} \in X^n$ and $\boldsymbol{y} \in \mathbb{R}^m$. Verify that $\forall \boldsymbol{x} \in X^n$, the property $\mathbb{P}$ holds.

Since a random forest is a pure function and thus there is no state space to explore, this problem may be addressed by considering all combinations of paths through trees in the forest. Furthermore, by partitioning the input domain into equivalence classes, i.e. sets of points in the input domain that yield the same output, constraint satisfaction may be verified for regions in the input domain, rather than for individual points explicitly.

Problem 2 (Equivalence Class Partitioning). For each path combination p in a random forest $f : X^n \rightarrow \mathbb{R}^m$, determine the complete set of inputs $X_p \subseteq X^n$ that lead to traversing p, and the corresponding output $\boldsymbol{y}_p \in \mathbb{R}^m$. Then verify that $\forall \boldsymbol{x} \in X_p$, the property $\mathbb{P}$ holds.

Our method efficiently generates equivalence classes as pairs of $(X_p, \boldsymbol{y}_p)$, and automatically verifies the satisfaction of a property $\mathbb{P}$. Assuming that the trees in a random forest are of equal size, the number of path combinations in the random forest is $2^{d \cdot B}$. In practice, decisions made by the individual trees are influenced by a subset of features shared amongst several trees within the same forest, and thus several path combinations are infeasible and may be discarded from analysis.

Example 1 (Discarded Path Combination). Consider a random forest with the trees depicted in Figure 2. There are four path combinations. However, x cannot be less than or equal to zero at the same time as being greater than five. Consequently, Tree 1 cannot emit 1 at the same time as Tree 2 emits 3, and thus one path combination may be discarded from analysis.

We postulate that since several path combinations may be discarded from analysis, all equivalence classes in a random forest may be computed and enumerated within a reasonable amount of time for practical applications. To explore this idea, we developed the tool VoRF[1] which automates the computation, enumeration, and verification of equivalence classes.

[1] https://github.com/john-tornblom/vorf

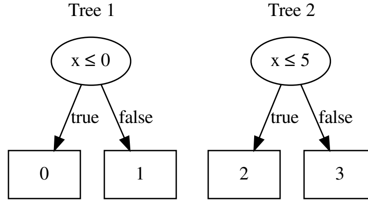

Fig. 2. Two decision trees that when combined into a random forest, contains three feasible path combinations and one discarded path combination.

4.2 Tool Overview

VoRF consists of two distinct components, VoRF Core and VoRF Property Checker. VoRF Core takes as input a random forest $f : X^n \to \mathbb{R}^m$, a hyper-rectangle defining the input domain X^n (which may include $\pm\infty$), and emits all equivalence classes in f. These equivalence classes are then processed by VoRF Property Checker that checks if all input/output mappings captured by each equivalence class are valid according to a property $\mathbb{P}$, as illustrated by Figure 3.

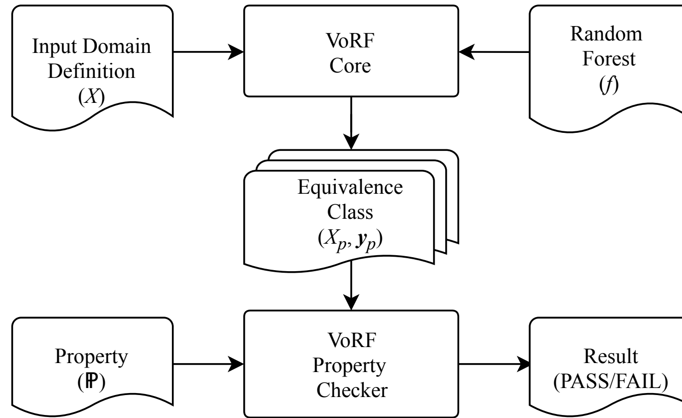

Fig. 3. Overview of VoRF.

4.3 Computing Equivalence Classes

There are three distinct tasks being carried out by VoRF Core while computing equivalence classes of a random forest:

- partitioning the input domain of decision trees into disjoint sets
- exploring all feasible path combinations in the random forest
- deriving output tuples from leaves.

Path exploration is performed by simply walking the trees depth-first. When a leaf is hit, the output $\boldsymbol{y}_p$ for the traversed path combination p is incremented with the value associated with the leaf, and path exploration continues with the next tree. The set of inputs X_p is captured by a set of constraints derived from decision functions associated with internal nodes encountered while traversing p. When the final leaf in a path combination is hit, $\boldsymbol{y}_p$ is divided by the number of trees B (recall the definition of a random forest in Section 1 which includes the same division). Finally, the VoRF Property Checker checks if the mappings from X_p to $\boldsymbol{y}_p$ comply with the property $\mathbb{P}$. If the property holds, the next available path combination is traversed, otherwise verification terminates with a "FAIL" and the most recent (X_p, $\boldsymbol{y}_p$) mapping as a counterexample.

4.4 Approximating Output Bounds

The output of a random forest may be bounded by analyzing each leaf in the collection of trees exactly once. Assuming that all trees are of equal size, the number of leaves in a random forest is $B \cdot 2^d$, where B is the number of trees and d the tree depth, thus making the analysis scale linearly with respect to the number of trees.

Let $f : X^n \to \mathbb{R}^m$ be a random forest, Y_t the union of all output tuples from all trees in the forest, and $L = |Y_t|$. A conservative approximation for an upper bound $\boldsymbol{y}_{\max} \geq f(\boldsymbol{x})$ may then be defined as

$$\boldsymbol{y}_{\max} = (\max\{y_{1,1}, \ldots, y_{1,L}\}, \ldots, \max\{y_{m,1}, \ldots, y_{m,L}\}),$$

where $y_{i,j}$ denotes the i-th element in the j-th tuple in Y_t. Analogously, a conservative approximation of a lower bound $\boldsymbol{y}_{\min} \leq f(\boldsymbol{x})$ may be defined. These bounds may then be used by a property checker to approximate f in e.g. the global safety property from Section 2.2. Note that these output bounds are conservative and approximate. If property checking does not return "PASS" with the approximation (see details below), the property $\mathbb{P}$ may still hold, and further analysis of the forest is required, e.g. by computing all possible equivalence classes (which are precise).

4.5 Implementation

This section presents implementation details of VoRF Core and VoRF Property Checker, and aspects that impact accuracy in floating point computations.

VoRF Core. For efficiency, core features in VoRF are implemented as a library in C, and utilize a pipeline architecture as illustrated by Figure 4 to compute and enumerate equivalence classes. The first processing element in the pipeline

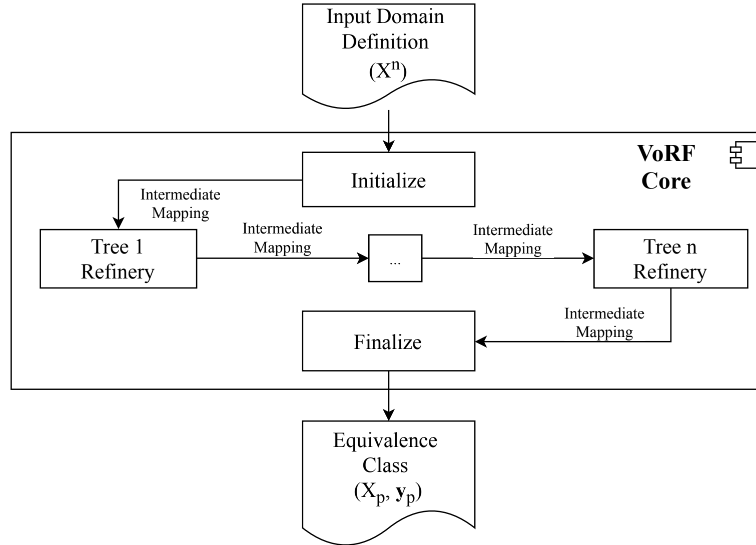

Fig. 4. Control flow of equivalence class partitioning in VoRF Core.

constructs an intermediate mapping from the entire input domain to an output tuple of zeros. The final processing element divides output tuples with the number of trees in the forest. In between, there is one refinery element for each tree that splits intermediate mappings into disjoint regions according to decision functions in the tree, and increments the output with values carried by the leaves.

To decouple VoRF from any particular random forest training library, a random forest is loaded into memory by reading a JSON-formatted file from disk. VoRF includes a tool[2] to convert random forests trained by the library scikit-learn [16] to this file format.

VoRF Property Checker. VoRF includes two pre-defined property checkers which are parameterized and executed from a command line interface; the global safety property checker, and the robustness property checker.

The global safety property checker first uses the output bounds approximation to check for property violations, and resorts to equivalence class analysis only when a violation is detected when using the approximation.

The robustness property checker checks that all points X_r within a hypercube with sides ϵ, centered around a test point $\boldsymbol{x}_t$, map to the same output. Note that selecting which test points to include in the verification may be problematic. In principle, all points in the input domain should be checked for robustness, but with random forest *classifiers*, there is always a hyperplane separating two classes from each other, and always points which violate the robustness property

[2] https://github.com/john-tornblom/vorf/blob/v0.1.0/support/train-sklearn.py

(adjacent to each side of the hyperplane). Hence, the property is only applicable to points at distances greater than ϵ from the classification boundary.

VoRF also includes Python bindings for easy prototyping of domain-specific property checkers. Example 2 depicts an implementation of the global safety property that uses these Python bindings to do sanity checking for a classifier's output.

Example 2 (Global Safety of a Classifier). Ensure that the probability of all classes in every prediction is within $[0, 1]$.

```
import sys
import vorf

def global_safety(mapping, alpha=0, beta=1):
    minval = min([mapping.outputs[dim].lower
                  for dim in range(mapping.nb_outputs)])

    maxval = max([mapping.outputs[dim].upper
                  for dim in range(mapping.nb_outputs)])

    return (minval >= alpha) and (maxval <= beta)

f = vorf.Forest(sys.argv[1]) # load model from disk
assert f.forall(global_safety)
```

Computational Accuracy. Implementations of random forests normally approximate real values as floating point numbers, and thus may suffer from inaccurate computations. In general, VoRF and the software subject to verification must use the same precision on floating point numbers and averaging function as in Definition 1 to get a compatible property satisfaction. In this version of VoRF, we use the same representation so that the calculation errors are the same as in the machine learning library scikit-learn [16]. Specifically, we approximate real values as 32-bit floating point numbers, and implement the averaging function literally as presented in Definition 1, i.e. by first computing the sum of all individual trees, then dividing by the number of trees. Other machine learning libraries may use 64-bit floating point numbers, and may implement the averaging function differently, e.g.

$$f(\boldsymbol{x}) = \sum_{b=1}^{B} \frac{t_b(\boldsymbol{x})}{B}.$$

This would be easily changeable in VoRF.

5 Case Studies

In this section, we present an evaluation of VoRF on two case studies found in the literature where neural networks have been analyzed for compliance with

interesting properties. Each case study defines a training set and a test set, and we used scikit-learn [16] to train random forests of different sizes. All training parameters except the number of trees and maximum tree depth were kept constant and at their default values. We evaluated accuracy on each trained model against its test set, i.e. the percentage of samples from the test set where there are no misclassifications. We then implemented verification cases for the global safety and robustness against noise properties (from Section 2.2) using VoRF. The time spent on verification was recorded for each trained model as presented below. All experiments were conducted on an Intel Core i5 2500K with 16GB RAM, running Ubuntu 18.04.

5.1 Vehicle Collision Detection

In this case study, we verified properties of random forests trained to detect collisions between two moving vehicles traveling along curved trajectories at different speeds. Each verified random forest accepts six input variables, emits two output variables, and contains 10-25 trees with depths 10-20.

Dataset. We used a simulation tool from Ehlers [7] to generate 30,000 training samples and 3,000 test samples. Unlike neural networks which Ehlers used in his case study, the size of a random forest is limited by the amount of data available during training, hence we generated ten times more training data than Ehlers to ensure that sufficient data is available for the size and number of trees assessed in our case study. Each sample contains the relative distance between the two vehicles, the speed and starting direction of the second vehicle, and the rotation speed of both vehicles.

Robustness. We verified the robustness against noise for all trained models by defining input regions surrounding each sample in the test set with the robustness margin $\epsilon = 0.05$. Table 1 lists random forests included in the experiment with their maximum tree depth d, number of trees B, accuracy of the classifications (Accuracy), elapsed time T during verification, and the percentage of samples from the test set where there were no misclassifications within the robustness region (Robustness).

Increasing the maximum depth of trees increased accuracy on the test set, but reduced the robustness against noise. This suggests that the models were overfitted with noiseless examples during training, and thus adding noisy examples to the training set may improve robustness. Verifying the largest random forest with $B = 25$ trees and depth $d = 20$ took approximately 1.5h. The significant drop in elapsed time between $\{d = 10, B = 25\}$ and $\{d = 20, B = 10\}$ may seem counter-intuitive at first. However, recall that the theoretical upper limit of the number of path combinations in a random forest is $2^{d \cdot B}$, and that $2^{20 \cdot 10} \ll 2^{10 \cdot 25}$.

Scalability. Next, we assessed the scalability of VoRF Core when the number of trees grows by verifying the trivial property $\mathbb{P} = \mathit{true}$ which accepts all

Table 1. Accuracy and robustness of random forests in the vehicle collision detection case study.

d	B	Accuracy (%)	T (s)	Robustness (%)
10	10	90.5	1	41.0
10	15	90.3	11	45.0
10	20	90.4	84	48.9
10	25	90.0	449	**50.3**
20	10	94.0	3	28.0
20	15	94.1	77	27.5
20	20	94.2	930	29.5
20	25	**94.5**	5499	29.6

input/output mappings. We implemented this trivial property in a verification case that also counts the number of equivalence classes emitted by VoRF Core. We then executed the verification case for all models with a tree depth of $d = 10$. The recoded number of equivalence classes C for different number of trees B is depicted in Figure 5 on a logarithmic scale. The number of equivalence classes

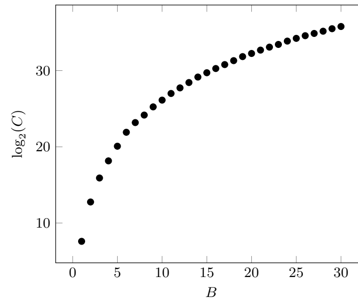

Fig. 5. Number of equivalent classes C on a logarithmic scale from the vehicle collision detection case study for different number of trees B with a depth $d = 10$.

increased exponentially as more trees were added, but the magnitude of the growth decreased for each added tree. The number of equivalence classes for large number of trees are significantly smaller than the upper limit of $2^{d \cdot B}$ (which occurs when there are no shared features amongst trees, and thus each path combination yields a distinct equivalence class).

Global Safety. Finally, we verified the global safety property (here ensuring that all predicted probabilities are in the range $[0, 1]$). All trained models passed the verification case within fractions of a second. This is expected since the output bound approximation algorithm implemented in the global safety property checker scales linearly with respect to the number of leaves in a forest, and thus there is no combinatorial explosion when the number of trees grows.

5.2 Digit Recognition

In this case study, we verified properties of random forests trained to recognize images of hand-written digits.

Dataset. The MNIST dataset [14] 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 was 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 (a similar split was used in [14]).

Robustness. We verified the robustness against noise for all trained models by defining 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. Each input region contains 2^{784} noisy images, which would be be too many for VoRF to handle within a reasonable amount of time. Consequently, we reduced the complexity of the problem significantly by only considering robustness against noise within a sliding window of 5x5 pixels. For a given sample from the test set, noise was added within the 5x5 window, yielding $2^{5 \cdot 5}$ noisy images. This operation was then repeated on the original image, but with the window placed at an offset of 1px relative to its previous position. Applying this operation on an entire image yields $2^{5 \cdot 5} \cdot (28-5)^2 \approx 2^{34}$ distinct noisy images per sample from the test set, and about 10^{14} noisy images when applied to the entire test set.

Table 2 lists random forests included in the experiment with their maximum tree depth d, number of trees B, accuracy on the test set (Accuracy), elapsed time T during verification, and the percentage of samples from the test set where there were no misclassifications within the robustness region (Robustness). Increasing the complexity of a random forest slightly increased its accuracy, and

Table 2. Accuracy and robustness of random forests in the digit recognition case study.

d	B	Accuracy (%)	T (s)	Robustness (%)
10	10	93.0	245	65.8
10	15	93.6	824	68.8
10	20	93.8	2010	75.2
10	25	94.2	10787	74.8
20	10	94.9	482	70.4
20	15	95.8	1626	77.6
20	20	96.0	4101	82.3
20	25	**96.4**	17411	**83.7**

significantly increased its robustness against noise. Verifying the largest random forest with $B = 25$ trees and depth $d = 20$ took approximately 5h.

Figure 6 depicts one of many examples from the MNIST dataset that were misclassified by the random forest with $B = 25$ and $d = 10$. Since the added noise is invisible to the naked eye, the noise (a single pixel) is highlighted in red.

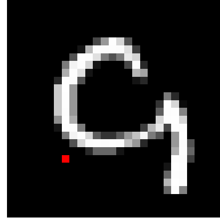

Fig. 6. A missclassified noisy sample from the MNIST dataset.

Scalability. Next, we assessed the scalability of VoRF Core when the number of trees grows by verifying the trivial property $\mathbb{P} = true$. This was done in a similar way as described in the vehicle collision detection use case presented in Section 5.1. We then executed the verification case for all models with a tree depth of $d = 10$. Enumerating all possible equivalence classes was intractable for random forests with more than $B = 4$ trees. We aborted the experiment after running the verification case with a random forest of $B = 5$ for 72h. Figure 7 depicts the four data points we managed to acquire. The number of equivalence

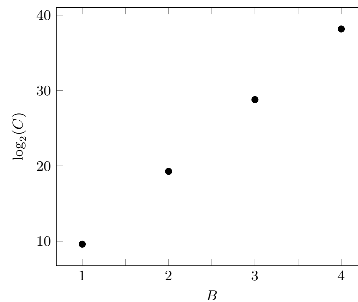

Fig. 7. Number of equivalent classes C on a logarithmic scale from the digit recognition case study for different number of trees B with a depth $d = 10$.

classes increased exponentially as more trees were added, without demonstrating any signs of stagnation. The ability to discard infeasible path combinations in a random forest is an essential ingredient to our method. When random forests are trained on high-dimensional data, the number of features shared between trees is

relatively low, so it is not surprising that our method experiences combinatorial path explosion. This shows that in non-trivial applications, transforming domain knowledge into reasonable constraints in the form of a property $\mathbb{P}$ is a useful means of addressing combinatorial problems in verification.

Global Safety. Finally, we verified the global safety property (again ensuring that that all predicted probabilities are in the range $[0, 1]$). All trained models passed the verification case within seconds. This is expected since the output bound approximation algorithm implemented in the global safety property checker scales linearly with respect to the number of leaves in a forest, and thus there is no combinatorial explosion when the number of trees grows.

6 Conclusions and Future Work

In this paper, we proposed a method to formally verify properties of random forests. Our method exploits the fact that several trees make decisions based on a shared subset of the input variables, and thus several path combinations in a random forest are infeasible. We implemented the method in a tool called VoRF, and demonstrated its scalability on two case studies.

In the first case study, a collision detection problem with six input variables, we demonstrated that problems with a low-dimensional input space can be verified using our method within a reasonable amount of time. In the second case study, a digit recognition problem with 784 input variables, we demonstrated that our method copes with high-dimensional input space when verifying robustness against noise. But it does so only if the systematically introduced noise does not attempt to exhaustively cover all possibilities. Since the number of shared input variables between trees is low, we observed a combinatorial explosion of paths in the forest. However, we successfully verified the global safety property in both case studies within seconds by using a fast approximation algorithm that scales linearly with respect to the number of trees in a random forest.

For future work, we plan to extend our method to include concepts from abstract interpretation to address the combinatorial path explosion observed when verifying the robustness property on high-dimensional data. Other directions of work include studying different search strategies, applying to use cases where control is involved (and not only sensing), and creating new properties that are meaningful in the context of the problem at hand, e.g. decisive classifications.

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. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. In: Advances in Neural Information Processing Systems (NIPS) (2018)
2. Breiman, L.: Classification and regression trees. Wadsworth International Group (1984)
3. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
4. Burton, S., Gauerhof, L., Heinzemann, C.: Making the case for safety of machine learning in highly automated driving. In: International Conference on Computer Safety, Reliability, and Security (SAFECOMP). pp. 5–16. Springer (2017). https://doi.org/10.1007/978-3-319-66284-8_1
5. DO-178C: Software Considerations in Airborne Systems and Equipment Certification. RTCA, Inc. (2012)
6. DO-333: Formal Methods Supplement to DO-178C and DO-278A. RTCA, Inc. (2012)
7. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 269–286. Springer (2017). https://doi.org/10.1007/978-3-319-68167-2_19
8. Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S.: Dermatologist-level classification of skin cancer with deep neural networks. Nature **542**(7639), 115 (2017). https://doi.org/10.1038/nature21056
9. Irsoy, O., Yildiz, O.T., Alpaydin, E.: Soft decision trees. In: International Conference on Pattern Recognition (ICPR) (2012)
10. ISO 26262: Road vehicles - Functional safety. International Organization for Standardization (2011)
11. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th. pp. 1–10. IEEE (2016). https://doi.org/10.1109/DASC.2016.7778091
12. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification (CAV). pp. 97–117. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_5
13. Kurd, Z., Kelly, T., Austin, J.: Developing artificial neural networks for safety critical systems. Neural Computing and Applications **16**(1), 11–19 (2007). https://doi.org/10.1007/s00521-006-0039-9
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
15. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning (ICML) (2018)
16. 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**(Oct), 2825–2830 (2011)
17. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Communications **25**(2), 117–135 (2012). https://doi.org/10.3233/AIC-2012-0525

18. Russell, S., Dewey, D., Tegmark, M.: Research priorities for robust and beneficial artificial intelligence. AI Magazine **36**(4), 105–114 (2015). https://doi.org/10.1609/aimag.v36i4.2577
19. Scheibler, K., Winterer, L., Wimmer, R., Becker, B.: Towards verification of artificial neural networks. In: Automatic Verification And Analysis of Complex Systems (MBMV). pp. 30–40 (2015)
20. Seshia, S.A., Zhu, X.J., Krause, A., Jha, S.: Machine learning and formal methods (dagstuhl seminar 17351). In: Dagstuhl Reports. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018). https://doi.org/10.4230/DagRep.7.8.55
21. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961