SENSITIVITY VERIFICATION FOR ADDITIVE DECISION TREE ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Tree ensemble models, such as Gradient Boosted Decision Trees (GBDTs) and random forests, are widely popular models for a variety of machine learning tasks. The power of these models comes from the ensemble of decision trees, which makes analysis of such models significantly harder than for single trees. As a result, recent work has focused on developing exact and approximate techniques for questions such as robustness verification, fairness and explainability, for such models of tree ensembles.

In this paper, we focus on a specific problem of feature sensitivity for additive decision tree ensembles and build a formal verification framework for it. We start by showing theoretical (NP-)hardness of the problem and explain how it relates to other verification problems. Next, we provide a novel encoding of the problem using pseudo-Boolean constraints. Based on this encoding, we develop a tunable algorithm to perform sensitivity analysis, which can trade off precision for running time. We implement our algorithm and study its performance on a suite of GBDT benchmarks from the literature. Our experiments show the practical utility of our approach and its improved performance compared to existing approaches.

1 INTRODUCTION

Tree ensemble models, such as gradient boosted decision trees (Friedman, 2001) and random forests (Breiman, 2001), are now widely used for machine learning tasks in domains ranging from banking applications (Madaan et al., 2021) to computer vision (Criminisi & Shotton, 2013) to transportation (Podgorelec et al., 2002). The power of these models comes from the ensembling or boosting which is known to empirically improve performance, unlike single decision trees which are explainable and simple to understand, but unwieldy to model complex behavior. XGBoost (Chen & Guestrin, 2016), one such popular tree ensemble learning algorithm, shows remarkable performance with tree ensembles of size 100, where each tree has depth at most 5 or 6. Of course the tradeoff with such modeling power is that it becomes difficult to predict and analyze these models, i.e., to ensure that they are reliable, robust, and behave as expected. This has led to a rich line of work in the last decade on formalizing and verifying different properties of tree ensembles.

The first such property is robustness checking which asks whether there are adversarial input perturbations that could lead to misclassification. This problem has been addressed by several works over the past 5 years including Chen et al. (2019b); Einziger et al. (2019); Devos et al. (2021), using different techniques ranging from optimization/MILP-based (Kantchelian et al., 2016) to SMT-solver based approaches (Ignatiev et al., 2020a). Using SMT-solvers allows one to give guarantees of soundness and completeness, highly desirable when dealing with reliability issues, but is often less scalable than purely optimization based approaches. The literature also distinguishes between local (checking robustness around a given input) and global robustness (checking for all inputs) where the universal quantifier makes the latter problem significantly harder (Chen et al., 2019b; Leino et al., 2021). Many techniques also apply approximations (Devos et al., 2021) or look at subclasses (Andriushchenko & Hein, 2019). Indeed, this is unsurprising since in Kantchelian et al. (2016), it was shown that a very simple related problem is already NP-hard.

Another property of interest is the sensitivity of a subset of inputs or features. A model is sensitive to a set of features if by keeping other features fixed and changing those features, the output of the model changes. In practice, models which are sensitive to a very small subset of features are prone to adversarial attacks. This question has also been seen as a way to formalize fairness (e.g., Dwork et al. (2012)) or causal discrimination (e.g, Galhotra et al. (2017)): sensitive features can be seen as protected, and changing the decision based only on these features can be seen as being unfair. This formulation was also adopted in Calzavara et al. (2023), where an approximation algorithm was provided for the same problem. Further, the generic tool Devos et al. (2021) solves verification questions including sensitivity/fairness, but provides approximate answers using optimization. A formal methods approach towards the problem is also provided in Ignatiev et al. (2020a), where the authors show how the fairness problem can be modeled as a global robustness query and hence relate the above problems. Another related approach in Törnblom & Nadjm-Tehrani (2020) used abstract-interpretation techniques for verifying tree ensembles.

In this paper, we launch a deeper investigation into the sensitivity problem for decision tree ensemble models, both from theoretical and practical perspectives. We start from observing that in most models and benchmarks we are only interested in checking sensitivity of a fixed few or even just one input feature. So the first question we ask is whether the problem remains hard even for a bounded number of features being changed. Second, if we look at tree ensemble training algorithms, like XGBoost, we observe that the decisions done at leaves, though binary, are derived from real number values, obtained from confidence. This leads us to ask if we can use these numbers as parameters to obtain a parametric notion of sensitivity, which can quantify "how sensitive features are" instead of just saying if they are sensitive or not? Third, we observe that existing encodings of the sensitivity problem either use one extreme of Boolean reasoning (e.g, SMT solvers in Ignatiev et al. (2020a)) or the other extreme of purely optimization-based reasoning (e.g, MILP solvers in Kantchelian et al. (2016)). Can we use other forms of powerful reasoning, such as Pseudo-Boolean solvers, that have shown to be effective in other problems (Mexi et al., 2023) for the sensitivity problem? This forms the third question that we try to address in this work.

Surprisingly, we show that even for a single feature, the sensitivity problem remains NP-hard. To show this, we provide a novel reduction from 3CNF-SAT which in fact shows that the problem is NP-hard even when restricted to decision tree ensembles with trees of depth at most 3. Next, we introduce a threshold parameter p to define the class change confidence for the sensitivity problem. In other words, if the output of the classifier is above p threshold, we say that the decision is 1 and if the output the classifier is below $1 - p$, we say that decision is 0. This gives a natural tunable version of the sensitivity problem. Using this, we formulate the sensitivity problem as pseudo-Boolean constraints. Our novelty in the encoding includes a new encoding of trees as pseudo-Boolean constraints, which is rather different from the encoding of trees done in Ignatiev et al. (2020a) for the robustness verification problem. Using this novel encoding we model our property so that we can take advantage of recent advances in pseudo-Boolean solving towards this problem. We then perform experiments to illustrate that our algorithm has an order of magnitude better performance than the state-of-the-art. In sum, our contributions are the following:

1. We formulate a parametric and bounded version sensitivity problem for additive decision tree ensembles and show that it is NP-complete.
2. We develop a novel encoding of this problem using pseudo-Boolean constraints.
3. We implement our algorithm and show that it outperforms state-of-the-art publicly available tools in a suite of benchmarks.

We highlight that our hardness results and encoding ideas work for any tree ensemble which aggregates the trees by summing up the individual tree outputs (these are sometimes called additive tree ensembles, see e.g., Devos et al. (2021)). Such models include GBDTs and random forests which follow an additive predictive model. In particular, our results are independent of the way by which the tree ensemble was trained (e.g, gradient boosting etc).

Other related work In addition to the work mentioned above, there are a few other lines of related work. SMT solvers and solver-based approaches and non-trivial encodings have been widely used for certifying robustness in neural networks and detailed frameworks such as DeepPoly (Singh et al., 2019) and α, β-Crown (Zhang et al., 2018; 2022) and some of these could also be related to sensitivity verification. However, there is comparatively less work on using these approaches for decision tree ensembles, beyond the works mentioned in the introduction. Another rich line of related work is to train robust tree ensembles. Calzavara et al. (2020), for instance, trains trees to make them more evasion aware while Chen et al. (2019a) provides a training method that makes the trees more robust. Our current focus is on verifying sensitivity but it would also be interesting to extend this to training insensitive models of tree ensembles. Finally, there is a relation between notions of explainability such as contrastive explainability and abductive explainability (Ignatiev et al., 2020b) and sensitivity or fairness. However, there is a marked difference as we focus on global sensitivity based on given features, while explainability refers to the change of class with respect to or around an input. However, global notions of contrastive explainability could perhaps be related to our approach.

2 PRELIMINARIES

In the classification setting, given an input space $\mathcal{X} \subseteq \mathbb{R}^d$ defined over d-dimensional space of features $\mathcal{F}$, and an output space $\mathcal{Y} \subseteq \mathbb{R}$, there exists a unknown function $h : \mathcal{X} \to \mathcal{Y}$ that maps each element of $\mathcal{X}$ to its corresponding correct output in $\mathcal{Y}$. A classifier is a function from $\mathcal{X}$ to $\mathcal{Y}$ which approximates h by learning from data. Decision trees and tree ensembles are well-known models of classifiers.

Decision Trees Formally, a decision tree T is either a leaf n with label $n.val \in \mathcal{Y}$ or a internal node n with two children $n.yes$ and $n.no$ decision trees, and a guard $n.g$, where a guard is a linear inequality of the form $f < v$, where f is a feature and v is a constant. Given an input $x \in \mathcal{X}$, the decision tree evaluates it top-down by evaluating the guards along the way to reach a leaf, where it returns the output value of the leaf encountered. Formally, we define the interpretation of a tree as follows. For a guard $g = f < v$, $g(x)$ is true if $x_f < v$, where $x = (x_f)_{f \in \mathcal{F}}$. Then, $T(x) = T.val$, if T is a leaf, else if $T.g(x)$ is true it is $T.yes(x)$ and if $T.g(x)$ is false, it is $T.no(x)$.

Decision Tree Ensembles Rather than relying on a single tree to approximate the underlying function, decision tree ensembles utilize multiple trees, each learning different parts of the problem and then aggregate them to produce a single output. There are many different methods to train decision tree ensembles, and in this paper, we focus on XGBoost (Chen & Guestrin, 2016), a popular gradient-boosting algorithm. Formally, an ensemble model $c = \{T_1, ..., T_m\}$ is a set of decision trees. The ensemble model sums up the results of the member trees. We define the outcome of the ensemble model c as $c(x) = \Sigma_{i=1}^{m} T_i(x)$. Note that the decision tree ensembles that we consider in this paper are always additive (even if we do not explicitly mention it), i.e., they aggregate the trees by summing up the results of individual trees.

Tree Ensemble Classifiers In many real-world applications, the output space $\mathcal{Y}$ is finite, meaning that the set of possible outputs, or commonly known as labels in this scenario, is limited. A tree ensemble classifier c is a decision tree ensemble with an output space $\mathcal{Y} = \{0, 1, \ldots k-1\}$ where k is the total number of distinct classes. We will often conflate the tree ensemble classifier c with its interpretation and write $c : \mathcal{X} \to \mathcal{Y}$. A binary tree ensemble classifier is a decision tree ensemble with an output space of size 2.

While learning the tree ensemble classifers from data (as for instance done by XGBoost algorithm introduced in Chen & Guestrin (2016)) the output $c(x)$ for any input $x \in \mathcal{X}$ is always a real number. To obtain a binary output, e.g, in $\{-1, 1\}$, one way to set it to 1 if $c(x) \geq 0$ and -1 otherwise. However, a more common approach is to obtain probability values by applying the sigmoid function to the output of the ensemble. Let $c(x)$ be the raw output of the ensemble for input x. To map this to a probability space $[0, 1]$, the sigmoid function on $c(x)$, centered at 0 is applied making the new output: $\sigma(c(x)) = \frac{1}{1+e^{-c(x)}}$. This transformation gives us $\sigma(c(x))$, which represents the probability that the input x belongs to the positive class. The closer $\sigma(c(x))$ is to 1, the higher the likelihood that x is a positive example, and vice versa. Finally, if the output of sigmoid function ≥ 0.5, we assign it to the positive class (+1) and if the output < 0.5, we assign it to the negative class (-1).

3 THE SENSITIVITY PROBLEM: MODELING AND HARDNESS

We now define the sensitivity problem for tree ensembles. For an input $x \in \mathcal{X}$ and set of features $F \subseteq \mathcal{F}$, let x_F denote the projection of x onto F. When $F = f$, we use x_f to denote the scalar.

Definition 3.1. Given a tree ensemble classifier $c : \mathcal{X} \longrightarrow \mathcal{Y}$, and a set of features $F \subseteq \mathcal{F}$, c is said to be F-sensitive, if we can find two inputs $x, x' \in \mathcal{X}$ such that $x_{\mathcal{F} \setminus F} = x'_{\mathcal{F} \setminus F}$ and $c(x) \neq c(x')$. [1] The sensitivity problem asks whether a given tree ensemble classifier is F-sensitive to a given set of features F.

One issue with the above definition is that it just requires the two witness inputs to be classified differently, but does not take into account their distance from the decision boundary (of the classifier). As explained earlier, classifier learning algorithms, in fact, provide this information. Consider the example in Figure 1, where we have a two-tree T_1, T_2 ensemble c that is trained to be a recommendation system for television shows suitable for children. It is easy to see that $\alpha_1 = (6, 2, 14)$ and $\alpha_2 = (6, 2, 17)$ are two inputs such that $c(\alpha_1) = 0.1 - 1.2 = -1.1$ while $c(\alpha_2) = 0.7$. However, the pair $\beta_1 = (6, 4, 14)$ and $\beta_2 = (8, 4, 14)$ have $c(\beta_1) = 10.1$ and $c(\beta_2) = -9.9$. In a practical setting, (β_1, β_2) is a much more interesting pair of inputs since it takes an almost sure positive prediction to a quite sure negative prediction, which can potentially be very harmful compared to an unsure prediction getting flipped.

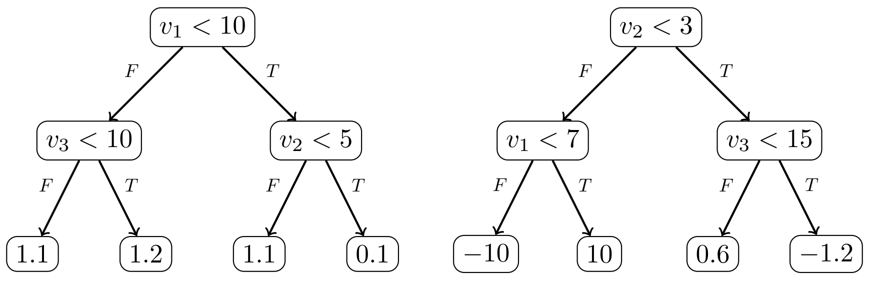


Figure 1: A tree ensemble with two trees T_1, T_2 having real valued (raw) outputs on leaves

This motivates a more nuanced definition that is parametrized by the difference using the sigmoid function σ that we want to see in the inputs that witness the change in classification wrt the sensitive features.

Definition 3.2. Given a tree ensemble classifier $c : \mathcal{X} \longrightarrow \mathcal{Y}$, sensitive features $F \subseteq \mathcal{F}$ and a parameter $p \geq 0$, c is said to be (p, F)-sensitive, if we can find two inputs $x, x' \in \mathcal{X}$ such that $(x)_{\mathcal{F} \setminus F} = (x')_{\mathcal{F} \setminus F}$ and $\sigma(c(x)) \geq 0.5 + p$, $\sigma(c(x')) \leq 0.5 - p$. The (p, F)-sensitivity problem asks if a given tree ensemble is (p, F)-sensitive wrt a given set of sensitive features F.

It is easy to see that the F-sensitivity problem is a special case of the (p, F)-sensitivity problem. Hence, for our hardness results, we will focus on the F-sensitivity problem. However, for our encoding and algorithm in the next section, as well as experimental results later, we will use (p, F)-sensitivity.

Sensitivity checking for single tree decision models is known to be in polynomial time (Chen et al., 2019b), but as we show next, this is not the case for general decision tree ensembles. We consider three variants of the problem, based on the size of the set of sensitive features F: (i) $|F| = 1$ or the single feature sensitivity problem, (ii) $|F| = k$ for any fixed constant k which we call the k-sized subset feature sensitivity problem and (iii) $|F| = |\mathcal{F}|$, the all feature sensitivity problem. Note that the single feature sensitivity is the same as 1-subset feature sensitivity. Second, the k-sized subset feature sensitivity problem is only defined if the number of features in the problem instance i.e., $|\mathcal{F}| \geq k$. Finally, note that the all feature sensitivity cannot be seen as k-sized subset sensitivity problem, since in the latter k is a part of the input, while in the former it is not.

Our main theoretical contribution is to show that all three variants are NP-hard. We start by showing the NP-hardness of the single feature sensitivity problem and obtain the others as corollaries. Before going into our proof, we recall that the robustness verification problem for decision tree ensembles was shown to be NP-hard using a reduction from the so-called evasion problem for decision tree

[1] We note that this problem has been called causal discrimination in Calzavara et al. (2023) and fairness in Dwork et al. (2012). While these are valid applications of the definition, the problem is itself closer to the analysis of sensitive features, and hence, we have called it such. Further, there are many competing definitions of fairness in the ML literature and while this is certainly one formulation it is not the only one.

ensembles, which was shown to be NP-hard in Kantchelian et al. (2016) (where, evasion for a given tree ensemble model c, asks whether there exists an $x \in \mathcal{X}$ such that $c(x) > 0$). However, while evasiveness can potentially be used to show hardness for the all feature sensitivity problem, it cannot be easily lifted to show single or fixed subset feature hardness. Instead we come up with a novel and direct reduction from 3CNF-SAT problem to show that single feature sensitivity is already NP-hard.

Theorem 1. The single feature sensitivity problem, i.e., checking whether a given tree ensemble classifier is F-sensitive for $|F| = 1$, is NP-hard.

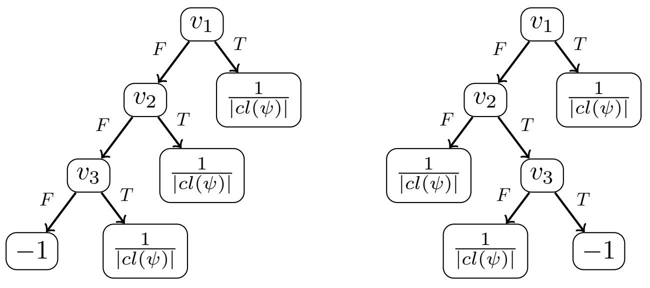
Proof. We will show a reduction from 3CNF-SAT, the classical NP-hard question, which asks given a Boolean formula in conjunctive normal form (CNF) with 3 variables per clause, whether it is satisfiable. Given an instance φ of 3CNF-SAT, let $cl(\varphi)$ be the set of clauses $\{cl_1, cl_2, \ldots, cl_m\}$, with $m = |cl(\varphi)|$ and let $var(\varphi)$ denote the set of variables $\{v_1, v_2, \ldots, v_n\}$, with $n = |var(\varphi)|$. Then from φ we start by creating the formula $\varphi' = \varphi \wedge (v_{n+1} \vee v_{n+1} \vee v_{n+1})$ which is also a 3CNF formula with a new variable v_{n+1} and a new clause $cl_{m+1} = (v_{n+1} \vee v_{n+1} \vee v_{n+1})$. Observe that φ is satisfiable, i.e., there exists an input $x \in \{0,1\}^n$ that satisfies φ iff φ' is satisfiable, i.e., there exists an input $x' \in \{0,1\}^{n+1}$ that satisfies φ'. We will now show a reduction to the (single feature) sensitivity problem. That is, we will construct a decision tree ensemble c with depth 3, such that c is 1-feature sensitive iff φ' is satisfiable.

In formula φ', for every clause cl_i, we create a depth-3 decision tree T_i as depicted in Figure 2, where $m + 1 = |cl(\varphi')|$. That is, for each literal (i.e., v_i or $\neg v_i$) in the clause, we add a "true" branch with output $\frac{1}{|cl(\varphi')|}$, and a "false" branch where we either continue to next literal or return -1 if there are no more literals left in the clause. For each literal, if it occurs positively as v_i (resp. negatively as $\neg v_i$), the true (resp. false) branch outputs $\frac{1}{|cl(\varphi')|}$. We form the decision tree ensemble c using the above decision trees with trees enumerated T_i for $i \in \{1, 2, \ldots, m+1 = |cl(\varphi')|\}$. Note that in this case, the domain of c, i.e., $\mathcal{X} = \{0,1\}^{n+1}$.

Claim. For all $x \in \{0,1\}^{n+1}$ we have $c(x) = 1$ iff $\varphi'(x) = 1$, i.e., x satisfies/models φ'.
Proof (of claim): There are 2 possible scenarios for an input x.

- The input satisfies the 3CNF formula φ', i.e., $\varphi'(x) = 1$. In this case, each of the $m + 1$ clauses are satisfied in the input and thus, for all trees T_i we have $T_i(x) = \frac{1}{m+1}$. Thus, $\Sigma_1^{m+1} T_i(x) = 1 > 0 \implies c(x) = 1$.
- The input does not satisfy the 3CNF formula φ'. Thus, there exists a clause which is not satisfied by the input. Let that clause be cl_j. By the construction of c, for the corresponding tree, $T_j(x) = -1$ and for all $i \neq j$, $T_i(x) \leq \frac{1}{m+1}$. Thus, $\Sigma_1^{m+1} T_i(x) \leq -1 + \frac{m}{m+1} = \frac{-1}{m+1} < 0 \implies c(x) = 0$.

Now, we use the above claim to prove hardness of sensitivity. More precisely, we will check sensitivity with respect to the single Boolean variable v_{n+1}. Call the set of all features $\mathcal{F}$ and the set for sensitivity checking $F = \{v_{n+1}\}$. To complete the proof, we will show that c is F-sensitive iff φ' is satisfiable.



(a): Clause $v_1 \vee v_2 \vee v_3$ (b): Clause $v_1 \vee \neg v_2 \vee \neg v_3$

Figure 2: Given a formula ψ with $|cl(\psi)|$ clauses, each clause is replaced by a tree above

In one direction, if c is F-sensitive, by definition, there exist $x, x' \in \{0,1\}^{n+1}$, with $x_{\perp F} = x'_{\perp F}$ such that $c(x) = 1$ and $c(x') = 0$. Thus we immediately infer that there exists x such that $c(x) = 1$, which by the above claim means that x satisfies φ' and hence φ' is satisfiable. In the other direction, if c is not F-sensitive. Then for all $x_{\perp F} \in \{0,1\}^n$, for all possible choices of x_F, x'_F, we must have $c(x_{\perp F}, x_F) = c(x_{\perp F}, x'_F)$. But now, if we consider $x_F = 0$, then the decision tree T_{m+1} will evaluate to -1 since $cl_{m+1}[v_{n+1} \mapsto 0] = 0$. As a result, we can conclude that for any $x_{\perp F} \in \{0,1\}^n$, we have $\sum_1^{m+1} T_i(x_{\perp F}, 0) \leq -1 + \frac{m}{m+1} < 0$ and so $c(x_{\perp F}, 0) = 0$. Thus, for any $x_F \in \{0,1\}$, $c(x_{\perp F}, x_F) = 0$, which implies that for all $x \in \{0,1\}^{n+1}$, $c(x) = 0$. Again appealing to the claim above, we can conclude that φ' is not satisfiable.

Thus, we have reduced finding satisfiability of φ' to checking sensitivity for a feature set of size 1 and hence the latter problem is NP-hard. □

We can now infer several interesting corollaries. First, we observe that the above proof can be lifted from single feature sensitivity to k-sized subset feature sensitivity for any fixed k. Given an arbitrary instance of the single feature sensitivity problem, we can construct an instance of a k-sized subset feature sensitivity (for $|F| = k$) by introducing $k - 1$ new dummy variables to the problem. These dummy variables are part of our input to sensitivity checking, but they do not affect the tree's output in any way (formally, one way to do this is to have decision tree stumps on these variables that output 0 irrespective of the variables' values). Thus, checking for sensitivity for k-sized subset of features will be equivalent to checking for sensitivity for just the first feature in the original instance and hence, checking sensitivity with respect to a given fixed size of subset of features is also NP-hard.

Corollary 1. For any fixed constant k, the k-sized subset feature sensitivity problem for decision tree ensembles is NP-hard.

The above argument holds for any k-sized sensitive feature set. But this does not immediately imply that it can be lifted to the case where all features are sensitive, i.e., $|F| = |\mathcal{F}|$. Indeed, one could imagine that the all-feature sensitivity problem could be easier. However, by modifying the proof above, we can show that this problem is also NP-hard. Due to lack of space, we provide this proof in Appendix A.

Corollary 2. The all feature sensitivity problem for decision tree ensembles is NP-hard.

Finally, an important remark is that our NP-hardness proof requires trees in the ensembles that have depth 3. This leaves open the intriguing question of whether the (single/k-sized subset/all) sensitivity problem is NP-hard for tree ensembles where the depth of each tree is at most 2. Unfortunately, our proofs cannot be extended to this case, since they rely on hardness of 3CNF-SAT, which requires 3 variables per clause which we translate to depth 3 trees. However, 2CNF-SAT is poly-time solvable and hence not useful to show hardness. We expect that a different technique/reduction/encoding will be needed to resolve this question and we leave this for future work. We also note that if we fix the number of trees and vary depth, or if we fix the depth and vary the number of trees NP-hardness follows, while if we fix both number of trees and depth, then the problem is easy.

Above, we considered hardness results. On the other hand, for all the problems mentioned above, we can obtain NP upper bounds. To see this, note that given a candidate solution i.e., values of inputs x and x' that only differ on the set of sensitive features F, we can trivially run the decision tree algorithm and check whether it is a valid solution or not, i.e., the decision for x differs from the decision for x'. Thus, both F-sensitivity and (p, F)-sensitivity problems are in NP hence they are NP-complete.

4 Encoding the Sensitivity Problem

In this section, we will consider encoding the problem of sensitivity of binary tree ensemble classifier X into solving of a set of pseudo-Boolean constraints Boros & Hammer (2002), which are arithmetic constraints containing only Boolean variables.

First, we observe that the general p-sensitivity problem in Definition 3.2 can be written as the problem of the search of the p-sensitive pairs of inputs x, x' for the set of features F as follows.

$$\exists x, x', \bigwedge_{f' \in \mathcal{F} \setminus F} x_{f'} = x'_{f'} \wedge \sigma(c(x)) > 0.5 + p \wedge \sigma(c(x')) < 0.5 - p$$

Since the output of the classifiers is via sigmoid function, we compute the inverse of the function to compute the required gap between the inputs of the sigmoid function. Let $\delta = \log(\frac{0.5+p}{0.5-p})$, which is the inverse of sigmoid. Let c consists of trees $T_1, ..., T_m$. The problem translates into

$$\exists x, x', \bigwedge_{f' \in \mathcal{F} \setminus F} x_{f'} = x'_{f'} \wedge \sum_{i=1}^{m} T_i(x) > \delta \wedge \sum_{i=1}^{m} T_i(x') < -\delta$$

Let us start encoding the above constraints using a Pseudo-Boolean encoding.

Encoding inputs The range of feature f is $[-\infty, \infty]$. However, c naturally divides the input in segments, which we define as follows. Let G_f be the set of all guards on feature f in c. Let $C_f = \mathit{toSortedList}(\{v | f < v \in G_f\})$ and $k_f = |C_f|$. The range of f is divided in $k_f + 1$ segments by C_f. We can encode the segments using k_f bits. For $0 \leq j < |C_f|$, let bit b_{1fj} indicate that feature f is less than $C_f[j]$ in input x and let bit b_{2fj} indicate that feature f is less than $C_f[j]$ in input x'. We need to include the following constraints, which encode that the boundaries are in increasing order.

$$b_{qfj} \Rightarrow b_{qf(j+1)} \tag{1}$$

We also need to say for each $f \notin F$, x_1 and x_2 will agree. Therefore, for each $j \in 1..k_f$, we add

$$b_{1fj} = b_{2fj}. \tag{2}$$

Encoding tree We need to encode the structure of the trees in constraints. For each node $n \in T_i$, let t_{qin} indicates that the node visited when evaluating $T_i(x_q)$. Since the roots of all the trees are visited, the following bits are always true.

$$t_{qiT_i} \tag{3}$$

For each internal node $n \in T_i$, let $(f < v) = n.g$ such that $C_f[j] = v$ for some j. We need to say that if n is visited, and $(f < v)$ is true then $n.yes$ is visited otherwise $n.no$ is visited.

$$(t_{qin} \wedge b_{bfj} \Rightarrow t_{qi(n.yes)}) \wedge (t_{qin} \wedge \neg b_{bfj} \Rightarrow t_{qi(n.no)}) \tag{4}$$

For each tree T_i, we may also *optionally* add the following constraints to help the solver to know that, at most one leaf of T_i can be visited by the input.

$$\sum_{n \in T_i.leaves} t_{qin} = 1 \tag{5}$$

Encoding output We now need to encode the condition that the sum of the tree outputs is greater than δ for x_1 and less than $-\delta$ for x_2. The label of a leaf $n \in T_i$ is a real value with infinite precision. We discretize this real value into steps of size $1/\alpha$, where α is the precision factor in our encoding. For the encoding of x_1, we multiply the leaf value $n.val$ by α and take the ceiling of the result. This constant integer is then multiplied by the corresponding t_{1in}. The ceiling operation increases the sum, which is then compared to the bounds derived from the floor of $\alpha\delta$. Similarly, we impose constraints for the output of x_2.

$$(\sum_{i=1}^{m} \sum_{n \in T_i.leaves} t_{1in} \lceil \alpha t.val \rceil) > \lfloor \alpha\delta \rfloor \wedge (\sum_{i=1}^{m} \sum_{n \in T_i.leaves} t_{2in} \lfloor t.val \rfloor) < \lceil -\alpha\delta \rceil \tag{6}$$

The above equations are the set of pseudo-boolean constraints.

Our encoding of the problem differs significantly from the one presented in Ignatiev et al. (2020a), which is a direct SMT encoding of trees and their verification query. The problem we address is similar to the knapsack problem, and the most appropriate encoding for it is through pseudo-Boolean constraints. Encoding the problem as SMT constraints, as done Ignatiev et al. (2020a), may result in a loss of structural information, preventing the solver from fully leveraging the pseudo-Boolean nature of the constraints. We solve these constraints using a pseudo-Boolean solver to check for p-sensitivity. The results from the solver provide the following guarantees.

Theorem 2 (Completeness and α-soundness)**.**

1. If conjunction (1) $\wedge$ (2) $\wedge$ (3) $\wedge$ (4) $\wedge$ (5) $\wedge$ (6) is unsatisfiable then the classifier c is not (p, F)-sensitive.
2. Otherwise, there is a counterexample pair (x, x') such that $c(x) > \frac{1}{1+e^{-\delta+((m+1)/\alpha)}}$ and $c(x') < \frac{1}{1+e^{\delta-((m+1)/\alpha)}}$, where $\delta = \log(\frac{0.5+p}{0.5-p})$ and α is the precision parameter of our analysis.

Proof sketch. Since the floor and ceiling operations are in conservative directions, 1 is true. In case of satisfiable constraints, there will be exactly m bits that are 1 in the sum in 6. The pre-sigmoid output in the reported counterexamples will be closer to zero by amount $(m+1)/\alpha$ in δ, due to m ceiling/floor operations on the left-hand side of the inequalities and one ceiling/floor operation on the right-hand side in 6. Therefore, 2 holds. □

As we increase α, the precision of our counterexamples improves. Thus, we can prove the following corollary, which states that if α is sufficiently large, the counterexamples will exhibit no reduction in gap, in other words, they are precise.

Corollary 3. There is an α such that any counterexample pair (x, x') will satisfy $c(x) > 0.5 + p$ and $c(x') < 0.5 - p$.

Proof Sketch. In the non-approximated constraint $\sum_{i=1}^{m} \sum_{n \in T_i.leaves} t_{1in} t.val > \delta$, there must be a gap between the closest sum to δ and δ. If the gap is bigger than $(m+1)/\alpha$, then approximation has no effect. □

5 Experiments

In this section, we present our tool, SensPB, which implements the above method for p-sensitivity checking. The tool is developed in Python and utilizes Z3 (de Moura & Bjørner, 2008) as its back-end pseudo-Boolean solver. We also tried a dedicated pseudo-Boolean solver Elffers & Nordström (2018), but as its performance was similar we kept to Z3. Our tool accepts as input a binary classifier XGBoost model, a set of features for which sensitivity is being assessed, the parameter p, and a precision parameter α. If the model is not sensitive, the tool outputs “pass”. Otherwise, it returns a pair of inputs that demonstrate p-sensitivity on the specified features.

To assess our method, we begin by running our tool on a set of XGBoost models from Chen et al. (2019b). Additionally, to evaluate the performance of our tool, we train XGBoost models with varying numbers of ensemble trees on 100,000 randomly generated data samples. We did not run experiments for (additive) Random Forests separately since, from the point of view of our encoding, they are equivalent to GBDT models. We ran the experiments on an Ubuntu machine with 20 1.3GHz cores, which has 64GB RAM.

There have been several tools (Devos et al., 2021; Törnblom & Nadjm-Tehrani, 2020; Chen et al., 2019b; Ignatiev et al., 2020a; Calzavara et al., 2023; Kantchelian et al., 2016) that implement different variants of verification for tree ensembles. In our experiment, we compare SensPB with the closest approach in Veritas (Devos et al., 2021) and an SMT based approach presented in Ignatiev et al. (2020a). We used our own implementation of the SMT based approach with Z3 (de Moura & Bjørner, 2008) as the SMT solver. We did not compare with Calzavara et al. (2023) as it only supports random forest trees and we were unable to make it work with XGBoost models. Finally, we did not include a comparison with Kantchelian et al. (2016) since Veritas has already demonstrated superior performance over this tool.

We ran SensPB on the benchmarks, and the results are presented in Table 1. For each benchmark ensemble, in one experiment we pick a feature f and run SensPB on the ensemble to check whether the classifier is p-sensitive to f. We repeat this experiment over all possible f, and the maximum, minimum and average time taken by us for termination is reported. We set a timeout of 1 hour for each experiment. In our experiments, we have set gap $p = 0.15$ and precision $\alpha = 10 \times |\#\text{Trees}|$. For the SMT solver-based approach, our experimental setup is the same as SensPB and we report the average time taken. More experiments can be found in Appendix C

Veritas doesn’t solve the sensitivity problem directly. We instead ask Veritas to maximise the difference between the outputs produced by two inputs which differ only in a feature. We define the

Benchmark Name	Details			SENSPB time taken (s)			SMT
	#Trees	Depth	#Feat	Min	Max	Average	
Breast cancer robust	4	5	11	2.72	2.81	2.76	2.76
Breast cancer unrobust	4	6	11	2.75	2.82	2.8	2.8
Diabetes robust	20	5	9	3	3.2	3.02	3.443
Diabetes unrobust	20	5	9	3.4	23	5.8	57.2
Cod-rna unrobust	80	4	8	7.2	12.9	8.3	106
Binary MNIST robust	50	6	784	14.6	15.3	14.9	TO
Higgs unrobust	100	8	28	130	TO	1188	TO
IJCNN robust	60	8	23	16	TO	330.7	TO
Synthetic 1	100	6	10	5.36	5.85	5.57	TO
Synthetic 2	125	6	10	6	6.35	6.2	TO
Synthetic 3	150	6	10	7.09	8.19	7.36	TO
Synthetic 4	175	6	10	6.25	6.57	6.37	TO
Synthetic 5	200	6	10	4.40	124.51	16.48	TO

Table 1: Times taken for verifying or countering sensitivity of all singular feature sets. The Min, Max and Averages in SENSPB times are taken by running the tool with different features of the benchmark tree ensembles as the sensitive feature. More information on these experiments is available in Appendix B

Benchmark Name	Time Taken (in seconds)		Accuracy comparison			
	SENSPB	**VERITAS**	**1x**	**2x**	**5x**	**3600**
Breast cancer robust	2.7	2.5	81.82%	81.82%	81.82%	81.82%
Breast cancer unrobust	2.7	2.53	45.45 %	45.45%	45.45%	45.45%
Diabetes robust	3.0	3.2	77.78%	77.78%	77.78%	77.78%
Diabetes unrobust	5.9	198.1	55.56%	66.67%	77.78%	88.89%
Cod-rna unrobust	8.3	346.2	0.00%	0%	25.00%	37.50%
Binary-mnist robust	14.9	TO	TO	TO	TO	TO
Higgs unrobust	1188	TO	TO	TO	TO	TO
IJCNN unrobust	330	OOM	TO	TO	0%	OOM

Table 2: 1) Runtime Comparison by letting VERITAS run on benchmarks with a timeout of 3600s. 2) Accuracy analysis of VERITAS with the time outs set to different values, depending on the time it took SENSPB. The percentages recorded represent that fraction of features where VERITAS gave better than or equivalent results as compared to SENSPB. For Time Analysis, TO implies that VERITAS timed out without reaching an optimal difference, while in the Accuracy measurements, TO means that VERITAS timed out without producing a single valid solution

bounds found by VERITAS being ”better than” the ones found by SENSPB if the bound found by VERITAS is greater than 2×gap. Note that this is a very relaxed definition since the bounds found by VERITAS might have a larger spread but might still be lying in the same output class. Since we can stop VERITAS anytime and observe the best solution found till then, we considered two kinds of experiments for fine-grained comparisons between the performance of SENSPB and VERITAS.

Firstly, we asked if we fix a timeout of 3600s, how does the performance of VERITAS compare with SENSPB, i.e., how long does it take VERITAS to reach an optimal solution? The results are present in the Time Analysis part of Table 2. For the Binary MNIST robust and Higgs unrobust benchmark ensembles, VERITAS always times out without producing a single solution. For IJCNN robust, VERITAS runs out of memory after roughly 900s.

Secondly, we asked if we ran VERITAS for the time relative to the time taken by SENSPB, what is be the relative performance? Let the time taken by SENSPB be x. We run VERITAS for sensitivity analysis, one feature at a time, with the following timeouts: x, $2x$, $5x$ and 3600. We look at the bound produced by VERITAS at the end of the timeout and compare this bound to the bound found by SENSPB. In the accuracy comparison section of Table 2, we report the percentage of features in which VERITAS performed better than or equivalent to SENSPB. As expected, on increasing the time given to VERITAS, it starts performing better on more and more features, e.g., Diabetes unrobust and Cod-rna unrobust. However, even on running VERITAS for 3600s it leaves out a lot of features in which SensPB performs better. As noted earlier, for three of our largest benchmarks, Veritas does not produce a single solution in the time given.

Our experiments clearly demonstrate that our pseudo-Boolean encoding significantly outperforms both the standard encoding and output configuration-based approaches by an order of magnitude. Given that the problem is NP-hard, SMT-based approaches are likely to surpass output configuration-based methods unless those methods are highly optimized. An SMT solver typically uses CDCL along with Simplex to solve the problem but may not fully exploit the specialized nature of our problem, particularly the limited role of arithmetic during bit summation at the output. As a result, pseudo-Boolean solvers specifically designed for such problems are expected to deliver the best performance. Our encoding allows us to not only use pseudo-boolean solvers but also provide new sets of benchmarks for the solvers. The availability of the benchmarks would likely improve the performances of the solvers.

6 Conclusion

In this paper, we investigated the sensitivity problem in two variants, the exact and a parametrized version. We presented new hardness proofs as well an efficient encoding into pseudo-Boolean constraints. Our implementation allowed us to exploit the recent advances in pseudo-Boolean solvers to solve the p-sensitivity problem. We successfully addressed XGBoost models of practical sizes, including scores of features, hundreds of trees, and a depth of 6-8. We believe that our work and especially the pseudo-Boolean encoding opens a new direction for scalable solutions for sensitivity and general verification problems for tree ensembles. For instance, an immediate extension would be to consider other tree ensemble models, such as random forests. Another natural future direction is also towards the question of multiclass labels, e.g., as done in Devos et al. (2024) for robustness verification.

References

Maksym Andriushchenko and Matthias Hein. Provably robust boosted decision stumps and trees against adversarial attacks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12997–13008, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/4206e38996fae4028a26d43b24f68d32-Abstract.html`.

Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1):155–225, 2002. ISSN 0166-218X. doi: https://doi.org/10.1016/S0166-218X(01)00341-9. URL `https://www.sciencedirect.com/science/article/pii/S0166218X01003419`.

Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL `https://doi.org/10.1023/A:1010933404324`.

S. Calzavara, L. Cazzaro, C. Lucchese, and F. Marcuzzi. Explainable global fairness verification of tree-based classifiers. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 1–17, Los Alamitos, CA, USA, feb 2023. IEEE Computer Society. doi: 10.1109/SaTML54575.2023.00011. URL `https://doi.ieeecomputersociety.org/10.1109/SaTML54575.2023.00011`.

Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: training evasion-aware decision trees. *Data Min. Knowl. Discov.*, 34(5):1390–1420, September 2020. ISSN 1384-5810. doi: 10.1007/s10618-020-00694-9. URL `https://doi.org/10.1007/s10618-020-00694-9`.

Hongge Chen, Huan Zhang, Duane S. Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1122–1131. PMLR, 2019a. URL `http://proceedings.mlr.press/v97/chen19m.html`.

Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane S. Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12317–12328, 2019b. URL `https://proceedings.neurips.cc/paper/2019/hash/cd9508fdaa5c1390e9cc329001cf1459-Abstract.html`.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145/2939672.2939785`.

Antonio Criminisi and Jamie Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. volume 4963, pp. 337–340, 04 2008. ISBN 978-3-540-78799-0. doi: 10.1007/978-3-540-78800-3_24.

Laurens Devos, Wannes Meert, and Jesse Davis. Versatile verification of tree ensembles. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2654–2664. PMLR, 2021. URL `http://proceedings.mlr.press/v139/devos21a.html`.

Laurens Devos, Lorenzo Cascioli, and Jesse Davis. Robustness verification of multi-class tree ensembles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21019–21028, Mar. 2024. doi: 10.1609/aaai.v38i19.30093. URL `https://ojs.aaai.org/index.php/AAAI/article/view/30093`.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In Shafi Goldwasser (ed.), *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pp. 214–226. ACM, 2012. doi: 10.1145/2090236.2090255. URL `https://doi.org/10.1145/2090236.2090255`.

Gil Einziger, Maayan Goldstein, Yaniv Sa'ar, and Itai Segall. Verifying robustness of gradient boosted models. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 2446–2453. AAAI Press, 2019. doi: 10.1609/AAAI.V33I01.33012446. URL `https://doi.org/10.1609/aaai.v33i01.33012446`.

Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang (ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 1291–1299. ijcai.org, 2018. doi: 10.24963/IJCAI.2018/180. URL `https://doi.org/10.24963/ijcai.2018/180`.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL `https://doi.org/10.1214/aos/1013203451`.

Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. *CoRR*, abs/1709.03221, 2017. URL `http://arxiv.org/abs/1709.03221`.

Alexey Ignatiev, Martin C. Cooper, Mohamed Siala, Emmanuel Hebrard, and João Marques-Silva. Towards formal fairness in machine learning. In Helmut Simonis (ed.), *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pp. 846–867. Springer, 2020a. doi: 10.1007/978-3-030-58475-7_49. URL `https://doi.org/10.1007/978-3-030-58475-7_49`.

Alexey Ignatiev, Nina Narodytska, Nicholas Asher, and João Marques-Silva. From contrastive to abductive explanations and back again. In Matteo Baldoni and Stefania Bandini (eds.), *AIxIA 2020 - Advances in Artificial Intelligence - XIXth International Conference of the Italian Association for Artificial Intelligence, Virtual Event, November 25-27, 2020, Revised Selected Papers*, volume 12414 of *Lecture Notes in Computer Science*, pp. 335–355. Springer, 2020b. doi: 10.1007/978-3-030-77091-4_21. URL `https://doi.org/10.1007/978-3-030-77091-4_21`.

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

Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6212–6222. PMLR, 2021. URL `http://proceedings.mlr.press/v139/leino21a.html`.

Mehul Madaan, Aniket Kumar, Chirag Keshri, Rachna Jain, and Preeti Nagrath. Loan default prediction using decision trees and random forest: A comparative study. *IOP Conference Series: Materials Science and Engineering*, 1022:012042, 01 2021. doi: 10.1088/1757-899X/1022/1/012042.

Gioni Mexi, Timo Berthold, Ambros M. Gleixner, and Jakob Nordström. Improving conflict analysis in MIP solvers by pseudo-boolean reasoning. In Roland H. C. Yap (ed.), *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pp. 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICS.CP.2023.27. URL `https://doi.org/10.4230/LIPIcs.CP.2023.27`.

Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. Decision trees: an overview and their use in medicine. *Journal of medical systems*, 26:445–463, 2002.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019. doi: 10.1145/3290354. URL `https://doi.org/10.1145/3290354`.

John Törnblom and Simin Nadjm-Tehrani. Formal verification of input-output mappings of tree ensembles. *Science of Computer Programming*, 194:102450, 2020. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2020.102450. URL `https://www.sciencedirect.com/science/article/pii/S0167642320300605`.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 4944–4953, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html`.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. General cutting planes for bound-propagation-based neural network verification. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/0b06c8673ebb453e5e468f7743d8f54e-Abstract-Conference.html`.

A Additional Theoretical Results

A.1 Proof for Corollary 2

The proof of Theorem 1 can be directly lifted with some minor changes to prove Corollary 2. Instead of checking sensitivity for $F = \{v_{n+1}\}$, we check sensitivity for $F = \mathcal{F}$ in the same setting. The first direction holds with the same argument as before. The reverse direction also holds with the same argument with the following changes.

- $x_{\perp F}$ is empty
- $x_F \in \{0,1\}^{n+1}$ instead of $\{0,1\}$

However, for the sake of completeness, we give the full proof below:

Proof. As before, we show reduction from 3CNF-SAT. Given an instance φ of 3CNF-SAT, let $cl(\varphi)$ be the set of clauses $\{cl_1, cl_2, \dots, cl_m\}$, with $m = |cl(\varphi)|$ and let $var(\varphi)$ denote the set of variables $\{v_1, v_2, \dots, v_n\}$, with $n = |var(\varphi)|$. Then from φ we start by creating the formula $\varphi' = \varphi \wedge (v_{n+1} \vee v_{n+1} \vee v_{n+1})$ which is also a 3CNF formula with a new variable v_{n+1} and a new clause $cl_{m+1} = (v_{n+1} \vee v_{n+1} \vee v_{n+1})$. Observe that φ is satisfiable, i.e., there exists an input $x \in \{0,1\}^n$ that satisfies φ iff φ' is satisfiable, i.e., there exists an input $x' \in \{0,1\}^{n+1}$ that satisfies φ'. We will now show a reduction to the (single feature) sensitivity problem. That is, we will construct a decision tree ensemble c with depth 3, such that c is 1-feature sensitive iff φ' is satisfiable.

In formula φ', for every clause cl_i, we create a depth-3 decision tree T_i as depicted in Figure 2, where $m+1 = |cl(\varphi')|$. That is, for each literal (i.e., v_i or $\neg v_i$) in the clause, we add a "true" branch with output $\frac{1}{|cl(\varphi')|}$, and a "false" branch where we either continue to next literal or return -1 if there are no more literals left in the clause. For each literal, if it occurs positively as v_i (resp. negatively as $\neg v_i$), the true (resp. false) branch outputs $\frac{1}{|cl(\varphi')|}$. We form the decision tree ensemble c using the above decision trees with trees enumerated T_i for $i \in \{1, 2, \dots, m+1 = |cl(\varphi')|\}$. Note that in this case, the domain of c, i.e., $\mathcal{X} = \{0,1\}^{n+1}$.

Claim. For all $x \in \{0,1\}^{n+1}$ we have $c(x) = 1$ iff $\varphi'(x) = 1$, i.e., x satisfies/models φ'.
Proof (of claim): There are 2 possible scenarios for an input x.

- The input satisfies the 3CNF formula φ', i.e., $\varphi'(x) = 1$. In this case, each of the $m+1$ clauses is satisfied in the input, and thus, for all trees T_i, we have $T_i(x) = \frac{1}{m+1}$. Thus, $\Sigma_1^{m+1} T_i(x) = 1 > 0 \implies c(x) = 1$.
- The input doesn't satisfy the 3CNF formula φ'. Thus, there exists a clause which is not satisfied by the input. Let that clause be cl_j. By the construction of c, for the corresponding tree $T_j(x) = -1$ and $T_i(x) \leq \frac{1}{m+1}$ for all $i \neq j$. Thus, $\Sigma_1^{m+1} T_i(x) \leq -1 + \frac{m}{m+1} = \frac{-1}{m+1} < 0 \implies c(x) = 0$.

Now, we use the above claim to prove hardness of sensitivity. More precisely, we will check sensitivity with respect to the singleton Boolean variable v_{n+1}. Call the set of all features $\mathcal{F}$ and the set for sensitivity checking $F = \mathcal{F}$. To complete the proof, we will show that c is F-sensitive iff φ' is satisfiable.

In one direction, if c is F-sensitive, by definition, there exist $x, x' \in \{0,1\}^{n+1}$, such that $c(x) = 1$ and $c(x') = 0$. Thus, we immediately infer that there exists x such that $c(x) = 1$, which by the above claim means that x satisfies φ' and hence φ' is satisfiable. In the other direction, if c is not F-sensitive. Then for all possible choices of x_F, x'_F, we must have $c(x_F) = c(x'_F)$. But now, if we consider $x_{v_{n+1}} = 0$, then the decision tree T_{m+1} will evaluate to -1 since $cl_{m+1}[v_{n+1} \mapsto 0] = 0$. As a result, we can conclude that for any $x_{F\setminus\{v_{n+1}\}} \in \{0,1\}^n$, we have $\sum_1^{m+1} T_i(x_{F\setminus\{v_{n+1}\}}, 0) \leq -1 + \frac{m}{m+1} < 0$ and so $c(x_{F\setminus\{v_{n+1}\}}, 0) = 0$. Thus, for any $x_F \in \{0,1\}^{n+1}$, $c(x_F) = 0$, which implies that for all $x \in \{0,1\}^{n+1}$, $c(x) = 0$. Again, appealing to the claim above, we can conclude that φ' is not satisfiable.

Thus, we have reduced finding satisfiability of φ' to checking sensitivity for the whole input feature set and hence the latter problem is NP-hard. $\square$

An interesting question that arises from the above proof is the requirement of the new clause cl_{m+1}. What we require is an input which doesn't satisfy φ'. If there is no such input, then even when the decision tree ensemble is insensitive to the set F, the 3CNF formula φ' can be satisfiable. Thus, to ensure such an input exists, we add the clause cl_{m+1}.

A.2 Hardness of the Differentiating Input Problem

From the same construction of trees in Theorem 1, we can show a novel yet interesting problem that we call the differentiating input problem is also NP-hard.

Definition A.1. Given 2 tree ensemble classifiers $c : \mathcal{X} \longrightarrow \mathcal{Y}$ and $c' : \mathcal{X} \longrightarrow \mathcal{Y}$, we say $x \in \mathcal{X}$ is a differentiating input for them if $c(x) \neq c'(x)$. Given 2 tree ensemble classifiers $c, c' : \mathcal{X} \longrightarrow \mathcal{Y}$, the differentiating input problem asks if there exists a differentiating input, i.e., $\exists x \in \mathcal{X}, c(x) \neq c'(x)$?

Corollary 4. The differentiating input problem for decision tree ensembles is NP-Hard.

Proof. We will show a reduction from 3CNF-SAT to this problem. Given an arbitrary 3CNF-SAT problem, we can create a decision tree classifier that solves this problem. Given an instance φ of 3CNF-SAT, let $cl(\varphi)$ be the set of clauses $\{cl_1, cl_2, \ldots, cl_m\}$, with $m = |cl(\varphi)|$. Then, for each clause, $cl_i = l_1 \vee l_2 \vee l_3$ in the 3CNF-SAT problem, create a decision tree, T_i such that $T_i(x) = \frac{1}{m}$ if x satisfies the clause and $T_i(x) = -1$ otherwise. Two examples for the same are shown in Figure 2(a) and 2(b). The ensemble $c = \{T_1, T_2, ..., T_m\}$ outputs a positive class (+1) if and only if the 3CNF-SAT formula was satisfiable as shown in the proof of Theorem 1.

Finally, we create another tree ensemble c', which always returns a negative class(-1). Thus, asking the question of whether there is a differentiating input for these tree ensembles is equivalent to asking whether the 3CNF-SAT formula was satisfiable, thus completing the hardness proof.

B More information on the experiments

The following table gives the fraction of all the features to which the benchmark trees are singularly sensitive to.

Benchmark Name	Number of Features Ran On	Percentage of Sensitive Features
Breast cancer robust	11	18.2
Breast cancer unrobust	11	36.4
Diabetes robust	9	44.4
Diabetes unrobust	9	80
Cod-rna unrobust	8	100
Binary mnist robust	10	0
Higgs unrobust	10	100
IJCNN robust	23	100

Table 3: Percentage of Sensitive Features

C Additional Experiments

Working on the IJCNN robust benchmark, we try to understand the effects of changing p and α (one at a time) on the running time of our algorithm. The IJCNN benchmark has 60 trees, with a maximum depth of 80 and 23 features and is a good representative of the kind of ensembles we aim to verify. While changing p, we keep the value of α fixed to the value we used in the main experiments (i.e. $10 \times |\#\text{Trees}|$). Likewise, while varying α, we keep p fixed to 0.15. We also give a table detailing the fraction of features where SensPB performs better than Veritas.

p	Time (s)
0.1	15.8
0.15	16
0.2	17.0
0.4	1100.0
0.45	923.4

Table 4: Effect of changing gap

α	Time (s)
100	16.2675621509552
200	15.795006513595581
500	15.795297384262085
700	16.174979209899902
1000	15.67112421989441
1500	22.39055609703064
2000	19.01390266418457
5000	19.632317543029785
100000	16.586485147476196
1000000	16.71101140975952

Table 5: Effect of changing precision

Benchmark Name	VERITASbetter	%SENSPBbetter
Breast cancer robust	81.82%	18.18%
Breast cancer unrobust	45.45 %	54.55%
Diabetes robust	77.78%	22.2%
Diabetes unrobust	55.56%	44.44%
Cod-rna unrobust	0.00%	100%
Binary-mnist robust	TO	100%
Higgs unrobust	TO	100%
IJCNN unrobust	TO	100%

Table 6: Comparison between the features where VERITASfinds a better bound than SENSPB. We first run SENSPBand then run VERITASfor the same amount of time. We look at the best results produced by VERITASin this time limit and use that value for the comparison.