Journal of Computer Security 0 (2022) 1–0
IOS Press

Certifying Machine Learning Models Against Evasion Attacks by Program Analysis

Stefano Calzavara [a], Pietro Ferrara [a] and Claudio Lucchese [a]
[a] *Università Ca' Foscari Venezia*

Abstract. Machine learning has proved invaluable for a range of different tasks, yet it also proved vulnerable to evasion attacks, i.e., maliciously crafted perturbations of inputs designed to force mispredictions. In this article we propose a novel technique to certify the security of machine learning models against evasion attacks with respect to an expressive threat model, where the attacker can be represented by an arbitrary imperative program. Our approach is based on a transformation of the model under attack into an equivalent imperative program, which is then analyzed using the traditional abstract interpretation framework. This solution is sound, efficient and general enough to be applied to a range of different models, including decision trees, logistic regression and neural networks. Our experiments on publicly available datasets show that our technique yields only a minimal number of false positives and scales up to cases which are intractable for a competitor approach.

Keywords: adversarial machine learning, abstract interpretation, evasion attacks, security certification

1. Introduction

Machine learning (ML) learns predictive models from data and has proved invaluable for a range of different tasks, yet it also proved vulnerable to *evasion attacks*, i.e., maliciously crafted perturbations of inputs designed to force mispredictions [1]. For example, let us assume a credit company uses a ML model to automatically assess whether customers qualify for a loan or not. A dishonest customer who realises that the model privileges unmarried people over married ones could cheat about their marital status to improperly qualify for a loan. Similarly, an attacker could try to fool a ML model used to detect malware by performing semantics-preserving modifications of malicious software with the goal of making it to be incorrectly classified as benign software [2]. In a different setting like image classification, even the corruption of a single pixel may lead to dramatic performance losses by deep neural networks [3], which hinders the effectiveness of face recognition algorithms in security-critical scenarios [4]. Unfortunately, traditional evaluations of ML models are not designed to capture their performance under attack, which motivates the rise of *adversarial* ML in the last few years [5].

In this article we are interested in the *security certification* of ML models deployed in an adversarial setting, i.e., we investigate techniques to quantify their resilience against evasion attacks. Our approach is based on a transformation of the model under attack into an equivalent imperative program, which is then analyzed using the classic abstract interpretation framework [6, 7]. This approach enjoys a unique blend of three desirable properties, which have never been achieved together by the state of the art:

(1) *Soundness*: abstract interpretation computes an over-approximation of the reachable program states. Since our program encodes all the possible evasion attacks, the results of the analysis provide a conservative estimate of the performance of the model in an adversarial setting, i.e., an automated formal proof of its resilience to evasion attacks.
(2) *Efficiency*: thanks to its modular approach, abstract interpretation allows one to apply different abstractions when statically analyzing a program. This way, one can strike an optimal trade-off between precision and efficiency by choosing appropriate abstract domains for the analysis.
(3) *Generality*: abstract interpretation supports arbitrary imperative programs, hence it is amenable for the analysis of different types of ML models under different types of attacks. Verifying ML techniques with respect to highly expressive threat models is nowadays one of the most compelling research directions of adversarial ML [8, 9].

This is a step forward over previous work, which either proposes empirical techniques without formal guarantees [1, 10], suffers from scalability issues [11, 12], or only focuses on specific ML models and artificial attackers expressed as mathematical distances [13, 14]. By analyzing different types of models under different threats and analysis configurations within a unifying framework, our work sheds light on the relative strengths and weaknesses of off-the-shelf program analysis tools for the security certification of ML models.

Contributions. We contribute as follows:

(1) We propose a general technique to certify the security of ML models against evasion attacks attempted by an attacker expressed as an arbitrary imperative program. We instantiate the technique to an expressive threat model based on rewriting rules [11] and we apply it to three different ML models: decision trees, logistic regression and neural networks (Section 3).
(2) We implement our technique into a new tool called ML-Cert. Given a ML model, an attacker and a test set of instances used to estimate prediction errors, ML-Cert outputs an over-approximation of the error rate that the attacker can force on the input model. ML-Cert implements an analysis computing a single over-approximation of the attacker's behavior and reuses it in the analysis of all the test instances, thus boosting efficiency without missing attacks (Section 4).
(3) We assess the effectiveness of ML-Cert against three public datasets. Our results show that ML-Cert is extremely accurate, since it can compute precise over-approximations of the actual error rate under attack, with no loss of precision in most cases. Moreover, ML-Cert is much faster than a competitor approach for decision trees [11] and scales to intractable cases, avoiding its exponential blow-up. Finally, we perform an in-depth evaluation of the effectiveness of ML-Cert for different types of model under different threats and analysis configurations (Section 5).

The present article extends a prior conference publication [15]. The original paper and the companion analysis tool only focused on decision tree models, while the present article shows that the same certification approach can be applied to two other types of ML models as well, i.e., logistic regression and neural networks, which confirms its generality. The present article also improves the prior program analysis by introducing the use of *trace partitioning* [16], which significantly increases the precision of the results while still being efficient enough for practical usage. Importantly, all these extensions have been implemented in the original analysis tool and experimentally validated on three public datasets, leading to a significantly extended experimental evaluation, which now includes different types of ML models, threats and analysis configurations.

2. Background

In this section we introduce the technical background required to appreciate the contributions of the article. In particular, we first discuss the security of supervised learning in general, we then present the specific types of ML models considered in our study, and we finally review the key ideas of the abstract interpretation framework that we use in our security certification approach.

2.1. Security of Supervised Learning

We deal with the security of *supervised learning*, i.e., the task of learning a classifier from a set of labeled data [17]. Formally, let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d-dimensional space of real-valued features and $\mathcal{Y}$ be a finite set of class labels; a *classifier* is a function $f : \mathcal{X} \to \mathcal{Y}$ which assigns a class label to each element of the vector space (*instance*). The correct label assignment for each instance is modeled by an unknown function $g : \mathcal{X} \to \mathcal{Y}$, called *target* function. In the rest of the article, we focus on binary classification, i.e., we assume $\mathcal{Y} = \{-1, +1\}$ for simplicity.

Given a *training set* of correctly labeled data $\mathcal{D}_{train} = \{(\mathbf{x}_1, g(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, g(\mathbf{x}_n))\}$ and a *hypothesis space* $\mathcal{H}$, the goal of supervised learning is finding the classifier $\hat{h} \in \mathcal{H}$ which best approximates the target function g. Specifically, we let:

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{L}(h, \mathcal{D}_{train}),$$

where $\mathcal{L}$ is a *loss* function which estimates the cost of the prediction errors made by h on $\mathcal{D}_{train}$. Once $\hat{h}$ is found, its performance is assessed by computing $\mathcal{L}(\hat{h}, \mathcal{D}_{test})$, where $\mathcal{D}_{test}$ is a *test set* of correctly labeled, held-out data drawn from the same distribution of $\mathcal{D}_{train}$.

When dealing with security certification, one should measure the accuracy of $\hat{h}$ by taking into account all the actions that an attacker could take to fool the classifier into mispredicting, i.e., *evasion attacks* [5, 18]. To provide a more accurate evaluation of the performance of the classifier under attack, the loss $\mathcal{L}$ can thus be replaced by the *loss under attack* $\mathcal{L}^A$ [19]. Formally, the attacker can be modeled as a function $A : \mathcal{X} \to 2^{\mathcal{X}}$ mapping each instance into a set of *perturbed* instances which might fool the classifier. The test set $\mathcal{D}_{test}$ can thus be corrupted into any dataset obtained by replacing each $(\mathbf{x}_i, y_i) \in \mathcal{D}_{test}$ with any $(\mathbf{x}'_i, y_i)$ such that $\mathbf{x}'_i \in A(\mathbf{x}_i)$; we let $A(\mathcal{D}_{test})$ stand for the set of all such datasets. The loss under attack $\mathcal{L}^A$ is thus defined by making the pessimistic assumption that the attacker is always able to craft the most damaging perturbations, as follows:

$$\mathcal{L}^A(\hat{h}, \mathcal{D}_{test}) = \max_{\mathcal{D}' \in A(\mathcal{D}_{test})} \mathcal{L}(\hat{h}, \mathcal{D}'). \tag{1}$$

Unfortunately, computing $\mathcal{L}^A$ by building $A(\mathcal{D}_{test})$ is intractable, given the huge number of perturbations available to the attacker: for example, if the attacker can flip K binary features, then each instance can be perturbed in 2^K different ways, leading to $2^K \cdot |\mathcal{D}_{test}|$ attacks.

2.2. Model Types

In the supervised learning approach, different sets of hypotheses $\mathcal{H}$ give rise to different types of ML models at training time. We discuss here the three types of models considered in the present work.

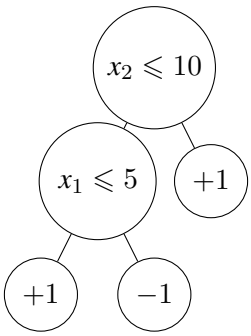

Fig. 1. Example of decision tree.

2.2.1. Decision Trees

We focus on traditional binary decision trees, whose internal nodes perform thresholding over feature values [20]. Such trees can be inductively defined as follows: a decision tree t is either a leaf $\lambda(\hat{y})$ for some label $\hat{y} \in \mathcal{Y}$ or a non-leaf node $\sigma(f, v, t_l, t_r)$, where $f \in [1, d]$ identifies a feature, $v \in \mathbb{R}$ is the threshold for the feature f and t_l, t_r are decision trees. At test time, an instance $\mathbf{x} = (x_1, \ldots, x_d)$ traverses the tree t until it reaches a leaf $\lambda(\hat{y})$, which returns the prediction $\hat{y}$, denoted by $t(\mathbf{x}) = \hat{y}$. Specifically, for each traversed tree node $\sigma(f, v, t_l, t_r)$, $\mathbf{x}$ falls into the left tree t_l if $x_f \leqslant v$, and into the right tree t_r otherwise.

Figure 1 represents an example decision tree, which assigns the instance (6,8) with label -1 to its correct class. In fact: (i) the first node checks whether the second feature, whose value is 8, is less than or equal to 10 and then takes the left sub-tree, and (ii) the second node checks whether the first feature, whose value is 6, is less than or equal to 5 and then takes the right leaf, classifying the instance with label -1.

2.2.2. Logistic Regression

Logistic regression is a statistical model that in its standard formulation uses a logistic function to perform binary predictions [21]. Specifically, given an instance $\mathbf{x}$, the probability of $\mathbf{x}$ belonging to class +1 is estimated as follows:

$$P(g(\mathbf{x}) = +1) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + k)}},$$

where $\mathbf{w}$ is a vector of weights and k is a constant, both identified by supervised learning. If the computed probability value is at least 0.5, the predicted class is +1, otherwise the predicted class is -1.

2.2.3. Neural Networks

We focus on artificial neural networks [17] with a single layer of hidden neurons, each using a ReLU activation function. Specifically, given an instance $\mathbf{x}$, the i-th neuron computes an output $o_i(\mathbf{x})$ as follows:

$$o_i(\mathbf{x}) = \max(0, \mathbf{w}_i \cdot \mathbf{x} + k_i),$$

where $\mathbf{w}_i$ is a vector of weights and k_i is a constant. The vector of the outputs of the neurons, denoted as $\mathbf{o}(\mathbf{x})$, is eventually used to estimate the probability of $\mathbf{x}$ belonging to class +1 as follows:

$$P(g(\mathbf{x}) = +1) = \frac{1}{1 + e^{-(\mathbf{v} \cdot \mathbf{o}(\mathbf{x}) + k^*)}},$$

where $\mathbf{v}$ is a vector of weights and k^* is a constant. If the computed probability value is at least 0.5, the predicted class is +1, otherwise the predicted class is -1. Here, all weights and constants are automatically identified by supervised learning.

2.3. Abstract Interpretation

Abstract interpretation is a classic, unifying approach to the static analysis of programs [6, 7]. In this work, we show how abstract interpretation can be applied to certify the security of different types of models trained via supervised learning. We review below the key ingredients of abstract interpretation and we refer to the original papers for more details.

Abstract Domains. In the abstract interpretation framework, the behavior of a program is approximated through *abstract values* from an *abstract domain* with a lattice structure, rather than concrete values. For example, the Sign domain abstracts numbers with their sign, based on the following *abstraction* and *concretization* functions α and γ respectively:

$$\alpha(V) = \begin{cases} \bot & \text{if } V = \emptyset \\ + & \text{if } \forall v \in V : v > 0 \\ 0 & \text{if } \forall v \in V : v = 0 \\ - & \text{if } \forall v \in V : v < 0 \\ \top & \text{otherwise} \end{cases} \qquad \gamma(a) = \begin{cases} \mathbb{R} & \text{if } a = \top \\ \{n \in \mathbb{R} \mid n > 0\} & \text{if } a = + \\ \{0\} & \text{if } a = 0 \\ \{n \in \mathbb{R} \mid n < 0\} & \text{if } a = - \\ \emptyset & \text{if } a = \bot \end{cases}$$

Note that for all sets of concrete values $V \subseteq \mathbb{R}$ we have $V \subseteq \gamma(\alpha(V))$, i.e., the abstraction function provides an over-approximation of the concrete values.

Thanks to its modular approach, abstract interpretation allows one to define multiple abstractions of the same concrete domain. Numerical abstractions have been widely studied and several abstractions have been proposed during the last decades. For instance, the aforementioned Sign domain tracks whether a variable is positive, negative or equal to zero. Instead, Intervals track for each variable its minimum and maximum value. These domains are called non-relational, since they track information about the values but not the relations among variables. Instead, Octagons [22] and Polyhedra [23] track different types of (linear) relations among numerical variables ($\pm x \pm y \leqslant c$ and $a_1 * v_1 + \cdots + a_n * v_n \leqslant c$, respectively), and have been fruitfully applied to different contexts. Apron [24] is a Java library of numerical abstract domains comprising the main domains leveraged in this work.

Abstract Semantics. Operations over concrete values like the sum operation $+$ are over-approximated by abstract counterparts $\oplus$ over the abstract domain, which define the *abstract semantics*. For example, the sum of two positive numbers is certainly positive, while the sum of a positive number and a negative number can be positive, negative or 0; this lack of information is modeled by $\top$. Hence, $\oplus$ is defined such that $+ \oplus + = +$ and $+ \oplus - = \top$. A sound definition of $\oplus$, here omitted, must ensure that for all $V_1, V_2 \subseteq \mathbb{R}$:

$$\{v_1 + v_2 \mid v_1 \in V_1 \wedge v_2 \in V_2\} \subseteq \gamma(\alpha(V_1) \oplus \alpha(V_2)),$$

i.e., abstract operations must over-approximate operations over concrete values.

By simulating the program over the abstract domains, abstract interpretation ensures a fast convergence to an over-approximation of all the reachable program states. In particular, the analysis consists

in computing the fixpoint of the abstract semantics over the abstract domain, making use of a *widening* operator, usually when the upper bound operator does not converge within a given threshold [6, 7]. Consider, for instance, the following for loop that initializes to zero all the cells of an array:

```
1  for(int i=0; i < arr.length; i++)
2    arr[i]=0;
```

In order to compute a sound overapproximation of the values of variable i inside the loop, abstract interpretation computes a fixpoint. In particular, when we analyze the body of the loop for the first time with the Sign domain, we have that i is 0. After the execution of the loop i is incremented by one, becoming +. The fixpoint computation proceeds then by merging the two results (computing the least upper bound of the abstract entry values of i, that is, $0 \sqcup + = \top$) and applying the semantics of the body again. This second iteration immediately converges, since the abstract value of i is $\top$.

Let us now consider the Intervals abstract domain. With such a domain, i will be $[0..0]$ during the first iteration, $[0..1]$ during the second one (since $[0..0] \sqcup [1..1] = [0..1]$), and then $[0..2]$, $[0..3]$, etc. Since the fixpoint computation does not converge, the abstract interpretation framework requires to define a widening operator ∇, that is an upper bound operator ensuring the convergence of the analysis. A common widening operator over intervals abstracts to $+\infty$ and $-\infty$ the right and left bound, respectively, if these bounds continue to expand in subsequent iterations. In the example above, the value of i will be abstracted with $[0.. + \infty]$, enforcing the convergence of the analysis.

Trace partitioning. Trace partitioning [16] is a generic analysis approach based on the abstract interpretation theory that allows for tracking disjunctive information. Assume for instance to analyze the following snippet of code with the Intervals domain:

```
1  if(j>0)
2    i = 1;
3  else i = −1;
```

Assuming that at the beginning of the analysis we know nothing about j (that is, $j = [-\infty.. + \infty]$), after the analysis we will infer that $j = [-\infty.. + \infty], i = [-1.. + 1]$, abstracting away that i cannot be zero, and that if j is positive then i is equal +1, and otherwise i is equal to -1.

Trace partitioning aims at improving the precision of the underneath domain in these cases. In particular, we can instruct this domain to partition the abstract state on the result of the condition at the first line of the program. In this way, the analysis will track two states: one for $j > 0$, and one for $j \leqslant 0$. Therefore, at the end of the analysis we will obtain the states $j = [1.. + \infty], i = [+1.. + 1]$ and $j = [-\infty..0], i = [-1.. - 1]$, thus recovering full precision.

3. Security Certification of ML Models

Here we detail the key points of our security certification technique. We first introduce our threat model, then we explain how we translate attackers and three types of ML models to imperative programs. We finally detail how we prove the robustness of these models under attack using static analysis and we discuss possible extensions of our approach.

3.1. Threat Model

Our approach is general enough to be applied to attackers represented as arbitrary imperative programs. However, to exemplify it, we discuss how it can be applied to an expressive threat model based on *rewriting rules* [11]. This relatively new threat model goes beyond traditional distance-based models, which are plausible for perceptual tasks like image recognition, but are inappropriate for non-perceptual tasks (e.g., loan assignment) where mathematical distances do not capture useful semantic properties of the domain of interest. Notice that our experimental evaluation also includes a standard distance-based attacker model for the sake of completeness, thus confirming our generality claims (see Section 5.5.2).

We model the attacker A as a pair (R, K), where R is a set of *rewriting rules*, defining how instances can be corrupted, and $K \in \mathbb{R}^+$ is a *budget*, limiting the amount of alteration the attacker can apply to each instance. Each rule $r \in R$ has form:

$$[a, b] \xrightarrow{f}_k [\delta_l, \delta_u],$$

where: (i) $[a, b]$ and $[\delta_l, \delta_u]$ are intervals on $\mathbb{R} \cup \{-\infty, +\infty\}$, with the former defining the *precondition* for the application of the rule and the latter defining the *magnitude* of the perturbation enabled by the rule; (ii) $f \in [1, d]$ is the index of the feature to perturb; and (iii) $k \in \mathbb{R}^+$ is the *cost* of the rule. The semantics of the rewriting rule can be explained as follows: if an instance $\mathbf{x} = (x_1, \ldots, x_d)$ satisfies the condition $x_f \in [a, b]$, then the attacker can corrupt it by adding any $v \in [\delta_l, \delta_u]$ to x_f and spending k from the available budget. The attacker can corrupt each instance by using as many rewriting rules as desired in any order, possibly multiple times, up to budget exhaustion. Note that, since the attacker can pick *any* perturbation $v \in [\delta_l, \delta_u]$, which is a continuous interval, hence the model readily applies to continuous data and the number of possible attacks is unbounded.

According to this attacker model, we can define $A(\mathbf{x})$, the set of the attacks against the instance $\mathbf{x}$, as follows.

Definition 1 (Attacks)**.** Given an instance $\mathbf{x}$ and an attacker $A = (R, K)$, we let $A(\mathbf{x})$ be the set of the *attacks* that can be obtained from $\mathbf{x}$, i.e., the set of the instances $\mathbf{x}'$ such that there exist a sequence of rewriting rules $r_1, \ldots, r_n \in R$ and a sequence of instances $\mathbf{x}_0, \ldots, \mathbf{x}_n$ where:

(1) $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_n = \mathbf{x}'$;
(2) for all $i \in [1, n]$, the instance $\mathbf{x}_{i-1}$ can be corrupted into the instance $\mathbf{x}_i$ by using the rewriting rule r_i, as described above;
(3) the sum of the costs of the rewriting rules $r_1, \ldots, r_n$ is not greater than K.

Notice that $\mathbf{x} \in A(\mathbf{x})$ for any A by picking an empty sequence of rewriting rules, i.e., the attacker can always leave the original instance unchanged.

To exemplify the threat model at work, consider the attacker $A = (\{r_1, r_2\}, 10)$, where:

- $r_1 = [0, 10] \xrightarrow{1}_5 [-1, 0]$ allows the attacker to corrupt the first feature by adding any value in the interval $[-1, 0]$, provided that the feature value is in $[0, 10]$ and the available budget is at least 5;
- $r_2 = [5, 10] \xrightarrow{2}_4 [0, 1]$ allows the attacker to corrupt the second feature by adding any value in the interval $[0, 1]$, provided that the feature value is in $[5, 10]$ and the available budget is at least 4.

Such an attacker A would force the decision tree in Figure 1 to change its original prediction -1 on the instance $(6, 8)$. In particular, we can show that $(5, 8)$ is a possible attack against $(6, 8)$, since A can apply

r_1 once by spending 5 from the budget, and $(5, 8)$ is classified as $+1$ by the decision tree in Figure 1. Note that this just a simple example of a possible attack, because the attacker can apply rules r_1, r_2 arbitrarily many times up to budget exhaustion, provided that their preconditions are still satisfied after corruption. For example, another possible attack against $(6, 8)$ is $(5.5, 8.5)$, where both features are corrupted by 0.5: this is possible because the cost of the attack is 9 and the attacker's budget is 10. Further increasing the attacker's budget would enable more combinations of rules: for example, an attacker with budget 20 could choose to apply rule r_1 four times, to apply rule r_2 five times, or to apply both rules twice each.

3.2. Proving Security by Program Analysis

We now discuss the details of the proposed analysis methodology. We first explain how to convert an attacker into an imperative program, we then discuss how different types of ML models can be similarly converted and we finally profit by the conversion by leveraging the abstract interpretation framework.

3.2.1. Attacker Conversion

We observe that the attacker $A = (R, K)$ can be represented by means of a *non-deterministic* program which behaves as follows:

(1) Select a random rewriting rule $r \in R$.
(2) Let $[a, b] \xrightarrow{f}_k [\delta_l, \delta_u]$ be the selected rewriting rule r and let $\mathbf{x} = (x_1, \ldots, x_d)$ be the instance to perturb. If $x_f \in [a, b]$ and the available budget is at least k, then select a random $\delta \in [\delta_l, \delta_u]$, replace x_f with $x_f + \delta$ and subtract k from the available budget.
(3) Non-deterministically go to step 1 or terminate the process. This stop condition allows the attacker to spare part of the budget, which is needed to enforce termination when the entire budget cannot be spent or does not need to be spent.

Figure 2 summarizes our construction, where the attacker is modeled as an imperative program, using traditional functions for random number generation. Note that the same idea could be applied to model distance-based attackers from the literature as well. For example, attackers based on the infinity-norm L_∞ can corrupt features by adding a maximum perturbation $k \in \mathbb{R}^+$ to them [14]. These attackers can be encoded through a program adding a randomly sampled $\delta \in [-k, k]$ to each corrupted feature.

3.2.2. Model Conversion

We now show how classifiers are translated into imperative programs. In particular, we discuss how we translate the three model types (decision trees, logistic regression, and neural networks) supported by our approach.

Decision Trees. Translating the decision tree into an imperative program is straightforward (see Figure 3). In particular, each internal node is translated into an if-then-else statement (e.g., the root of the tree in Figure 1 is translated into the if statement starting at line 2), while leaves are translated into a statement that returns the value of the label in the leaf (e.g., the leaf +1 that is in the right child of the root of the tree in Figure 1 is translated into the return statement at line 9).

Logistic Regression. As explained in Section 2.2.2, logistic regression models check whether the probability returned by the logistic function is at least 0.5. However, our encoding just checks whether the value of the exponent of e, that is, $(\mathbf{w} \cdot \mathbf{x} + k)$, is greater than or equal to 0. It is easy to see that this is equivalent to checking whether the probability is greater than or equal to 0.5. However, existing numerical domains (and Polyhedra in particular) focus mostly on linear constraints among variables, and

```
float [] attack (float [] x) {
    float K = 10;
    boolean done = false;
    while (!done) {
        int rule = random_int(1,3);
        switch (rule) {
            case 1:
                if (x[1] >= 0 && x[1] <= 10 && K >= 5) {
                    float delta = random_float(−1,0);
                    x[1] = x[1] + delta;
                    K = K − 5;
                }
                break;
            case 2:
                if (x[2] >= 5 && x[2] <= 10 && K >= 4) {
                    float delta = random_float(0,1);
                    x[2] = x[2] + delta;
                    K = K − 4;
                }
                break;
            case 3:
                // non−deterministic termination
                done = true;
        }
    }
    return x;
}
```

Fig. 2. Encoding the attacker into an imperative program.

```
int predict (float [] x) {
    if (x[2] <= 10) {
        if (x[1] <= 5)
            return +1;
        else
            return −1;
    }
    else
        return +1;
}
```

Fig. 3. Translation of the decision tree in Figure 1 into an imperative program.

therefore their direct application to the logistic function would lead to very imprecise results by the static analyzer.

Neural Networks. The encoding of neural networks employs, for each neuron in the hidden layer, a fresh local variable to store the linear combination of features computed to feed the ReLU activation function, as formalized in Section 2.2.3. The values resulting from applying the ReLU activations are then aggregated again through another linear combination and, similarly to the encoding of logistic regression, if the computed linear combination is non-negative then the predicted class is $+1$, otherwise the predicted class is -1.

```
int predict_att (float [] x) {
    float [] z = attack(x);
    return predict(z);
}
```

Fig. 4. Translation of a ML model under attack into an imperative program.

3.2.3. Models Under Attack

For a classifier h, an attacker A and a test set $\mathcal{D}_{test}$, we can compute a sound over-approximation of $\mathcal{L}^A(h, \mathcal{D}_{test})$ as follows. We first translate the classifier h together with the attacker A into an imperative program Q representing the classifier under attack, based on the conversions discussed in the previous section. This program is shown in Figure 4.

For each instance $(\mathbf{x}_i, y_i) \in \mathcal{D}_{test}$, we build an abstract state $\alpha(\{\mathbf{x}_i\})$ representing $\mathbf{x}_i$ in the chosen abstract domain and we analyze Q with such entry state. Then, the output of the analysis might be either of the following:

(1) only program points returning the correct class label y_i are reachable. This means that, for all possible attacks against $\mathbf{x}_i$, the classifier h always returns the correct class;
(2) program points returning the wrong label are reachable as well. If h correctly classifies the instance in the unattacked setting, this might happen either because there is indeed an attack leading to a misprediction or for a loss of precision due to the over-approximation performed by the static analysis.

Since our approach relies on sound static analysis engines, it is not possible to miss attacks, i.e., every instance which can be mispredicted upon attack must fall in the second case of our analysis.

Let $Q^{\#}(\mathbf{x}_i) = Y_i$ stand for the set of labels Y_i returned by the analysis of Q on the instance $\mathbf{x}_i$. By using this information, we can construct an *abstraction* of the behaviour of h under attack on $\mathcal{D}_{test}$ defined as follows:

$$\forall (\mathbf{x}_i, y_i) \in \mathcal{D}_{test} : h^{\#}(\mathbf{x}_i) = \begin{cases} y_i & \text{if } Q^{\#}(\mathbf{x}_i) = \{y_i\} \\ -y_i & \text{otherwise} \end{cases}$$

By construction, we have $\mathcal{L}^A(h, \mathcal{D}_{test}) \leqslant \mathcal{L}(h^{\#}, \mathcal{D}_{test})$ for any loss function which depends just on the number of mispredictions, like the *error rate*, i.e., the fraction of wrong predictions among all the performed predictions. This means that $h^{\#}$ enables an efficient approach to over-approximate the loss under attack $\mathcal{L}^A$ by computing just a traditional loss $\mathcal{L}$, which does not require the computation of the set of attacks.

3.3. Extensions

We discuss here possible extensions of our approach to different popular settings. We leave the implementation of these extensions to future work, since most of them are essentially an engineering effort.

3.3.1. Regression

The regression task requires one to learn a regressor rather than a classifier from the training data. The key difference between a regressor and a classifier is that the former does not assign a class from a

finite set $\mathcal{Y}$, but rather infers a numerical quantity from an unbounded set, e.g., estimates the salary of an employee based on their features. Regression can be modeled by revising the abstraction $h^\#$ such that it returns an abstract value over-approximating all the values of the predictions found in the leaves which are reachable upon attack. Formally, this means requiring $h^\#(\mathbf{x}_i) = \sqcup_{y_i \in P^\#(\mathbf{x}_i)} \alpha(\{y_i\})$, where $\sqcup$ stands for the least upper bound operator on the abstract domain underlying the analysis.

3.3.2. Multi-Class Classification

We explained our analysis approach for binary classification tasks, however our technique is general enough to be applied to multi-class classification as well. If the classifier already outputs a class from a set of labels $\mathcal{Y}$ such that $|\mathcal{Y}| > 2$, we can readily apply the approach in Section 3.2.3. Specifically, given an instance $(\mathbf{x}_i, y_i) \in \mathcal{D}_{test}$, we can change the abstraction $h^\#$ such that $h^\#(\mathbf{x}_i)$ returns any $y_i' \neq y_i$ whenever $Q^\#(\mathbf{x}) \neq \{y_i\}$. This is the most natural approach for decision trees, since they can be trained to predict classes from an arbitrary finite sets of labels. For other models, instead, multi-class classification can be reduced to multiple binary classifications using standard techniques, known as *one-vs.-rest* and *one-vs.-one* [25]. Thus, the restriction to binary classification does not yield any loss of generality.

3.3.3. Ensembles

Ensemble methods train multiple classifiers and combine them to improve prediction accuracy. Traditional ensemble approaches for decision trees include random forest [26] and gradient boosting [27]. Irrespective of how an ensemble is trained, its final predictions are performed by aggregating the predictions of the individual classifiers, e.g., using majority voting or averaging. This means that it is possible to extend our analysis technique to ensembles by translating each classifier therein and by aggregating their predictions in the generated imperative program.

3.3.4. Generalization to Other ML Models

Summing up, our approach consists of (i) encoding the attacker and the ML model as programs, and (ii) applying existing abstract domains to approximate all possible executions of these programs to detect evasion attacks. As already discussed, during the last decades abstract domains have been mostly focused on tracking numerical linear constraints over variables. In such a context, several robust implementations of these domains have been developed, and this technological ecosystem allows us to study how these analyses perform in practice on some popular models considered in the present work.

While our methodology is general, we acknowledge that its effectiveness depends on the behavior of the ML model and the abstract domain used to analyze it. Generally speaking, such an approach works well for ML models that are amenable to be represented in terms of assignments and conditions of linear expressions. For instance, tracking linear constraints is precise for any architecture of neural networks, since each neuron is a linear combination of features or of other neurons encoded in the program as local variables. Instead, some activation functions, such as the hyperbolic tangent, expose non-linear behaviors, and existing numerical abstract domains might achieve little precision in these cases. For this reason, we focused our approach on the ReLU activation function (that is amenable to reasoning on linear constraints), leaving other functions to future work.

Extending our analysis technique to arbitrary ML models requires to formalize, prove the correctness and develop novel abstract domains. In most cases, this would be by itself a distinct scientific contribution requiring additional research.

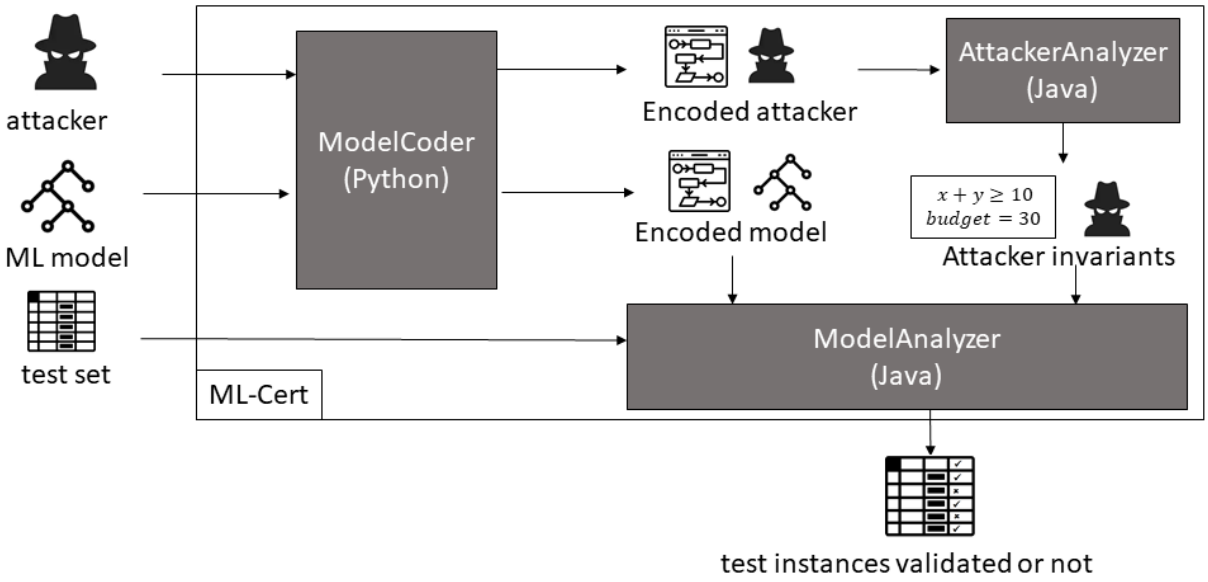

Fig. 5. The architecture of ML-Cert.

4. Implementation: ML-Cert

Figure 5 depicts the architecture of ML-Cert, our tool for the security certification of ML models.[1] Its inputs are: (*i*) the attacker, encoded in a JSON file supporting the threat model of Section 3.1, (*ii*) an ML model to analyse, serialized through the `joblib` library, and (*iii*) a test set in CSV format. ML-Cert reports for each test instance whether it is correctly classified for each possible attack or it might be incorrectly classified. The analysis is performed along three different modules, called ModelCoder, AttackerAnalyzer and ModelAnalyzer respectively, which we detail in the following.

4.1. ModelCoder

The first step of ML-Cert is to encode the attacker and the ML model as Java programs through the module ModelCoder, as described in Sections 3.2.1 and 3.2.2. ModelCoder is a Python script that, given an attacker model and a decision tree, produces two distinct Java files encoding the attacker (see method `attack` in Figure 2) and the machine learning algorithm (see method `predict` in Figure 3).

There are only two small technical differences over the previous presentation. First, given that all instances of the same dataset share the same set of features, instances are not encoded as arrays, but rather modeled using a distinct local variable for each feature, which simplifies the static analysis; specifically, we let variable x_i represent the initial value of the *i*-th feature and variable x_i' represent its value after the attack. In addition, each time a rewriting rule *r* is applied, we increment a counter r_counter, initially set to 0, which allows one to capture useful analysis invariants. Clearly, these changes do not affect the semantics of the generated program, so we did not include them in Figure 2 for simplicity.

[1] We release ML-Cert as opensource software and make it available at https://github.com/pietroferrara/staticanalyzer.

4.2. AttackerAnalyzer

The encoded attacker is then passed to the AttackerAnalyzer module, a static analyzer based on abstract interpretation. The analyzer interfaces with Apron, a standard library implementing many popular abstract domains. The analyzer then computes a fixpoint over the Java program representing the attacker, using the Polka implementation[2] of the Polyhedra domain [23].

Polka tracks linear equalities and inequalities over an arbitrary number of variables. These invariants allow AttackerAnalyzer to infer the upper and lower bounds of each attacked feature, based on how many times a feature can be attacked using the available budget. In addition, we combine this domain with *trace partitioning* [16], an abstract interpretation-based domain that allows one to track several abstract states per program point based on specific criteria. For instance, one might collect a distinct state for the first *n* iterations of a loop, or based on the value of a numerical variable (e.g., whether it is positive or negative), or on whether a specific program point is traversed. In our setting, we partition at each program point of the attacker that modifies the value of a feature. In this way, we distinguish the cases where a feature can be attacked (e.g., because the concrete values of a test case satisfy the rule precondition) or not.

To exemplify, pick the attacker in Figure 2. ML-Cert partitions the states at lines 10 and 17, that are the two program points where the attacker modifies the values of the features. In this way, we obtain four distinct abstract states that distinguishes when (i) both the first and second features cannot be attacked, because their initial values were outside the bounds of the attacker (that is, between 0 and 10 for the first feature, and between 5 and 10 for the second feature), (ii) only the first feature can be attacked, (iii) only the second feature can be attacked, and (iv) both features can be attacked. Let us consider the fourth case, that is, when both features can be attacked. Here, AttackerAnalyzer infers on the encoding of the attacker that, after the attack has been performed: (i) the value of the first feature may have been decreased by at most r1_counter (formally, $\mathsf{x}_1' \in [\mathsf{x}_1 - 1 * \mathsf{r1\_counter}, \mathsf{x}_1]$), (ii) the second feature may have been increased by at most r2_counter ($\mathsf{x}_2' \in [\mathsf{x}_2, \mathsf{x}_2 + 1 * \mathsf{r2\_counter}]$), (iii) both the counters are non-negative ($\mathsf{r1\_counter} \geqslant 0 \wedge \mathsf{r2\_counter} \geqslant 0$), and (iv) the budget spent in the application of the two rewriting rules is less than or equal to the initial budget ($5 * \mathsf{r1\_counter} + 4 * \mathsf{r2\_counter} \leqslant 10$). Note that the last invariant is inferred only if the calculation of a fixpoint over the abstract semantics did not require to apply the Polyhedra widening operator to convergence. Otherwise, the analysis would drop such information to ensure termination.

4.3. ModelAnalyzer

The attacker invariants are then passed to the ModelAnalyzer module together with the test set. Like AttackerAnalyzer, ModelAnalyzer performs a static analysis using the Polka implementation of the Polyhedra abstract domain. For each test instance $\mathbf{x}$, ModelAnalyzer (i) adds the initial values of the features of $\mathbf{x}$ to the attacker invariants, (ii) computes the fixpoint over the program encoding the decision tree t under attack, and (iii) uses it to return the output $t^{\#}(\mathbf{x})$. Thanks to the initial values of the features, the analysis can then prune away all the cases produced by trace partitioning that do not apply to the current test instance, which makes the analysis efficient.

To clarify, consider again the example in Figure 1, where the test instance $(6, 8)$ is correctly classified as -1 by the given decision tree, but can be misclassified upon attack. First of all, ModelAnalyzer adds the invariants $\mathsf{x}_1 = 6$ and $\mathsf{x}_2 = 8$ to the inferred attacker invariants, pruning away the first three

[2] http://apron.cri.ensmp.fr/library/0.9.10/mlapronidl/Polka.html

cases inferred by trace partitioning on the attacker encoding, and keeping only the fourth one, i.e., both features can be attacked. This leads to an initial Polyhedra state tracking that $x_1' \in [6 - \mathsf{r1\_counter}, 6]$ and $x_2' \in [8, 8 + \mathsf{r2\_counter}]$ with $5 * \mathsf{r1\_counter} + 4 * \mathsf{r2\_counter} \leqslant 10$. Then the static analysis of the encoded tree starts with the evaluation of the condition $x_2' \leqslant 10$, inferring that such condition is always evaluated to true: indeed, x_2 could become greater than 10 only if r2_counter was strictly greater than 2, but then $5 * \mathsf{r1\_counter} + 4 * \mathsf{r2\_counter} \leqslant 10$ could not hold since $\mathsf{r1\_counter} \geqslant 0$. ModelAnalyzer then analyzes the condition $x_1' \leqslant 5$. In this case, it cannot definitely conclude that the condition is always evaluated to false, since x_1 can become less than or equal to 5 if $\mathsf{r1\_counter} \geqslant 1$, which is allowed by the invariant $5 * \mathsf{r1\_counter} + 4 * \mathsf{r2\_counter} \leqslant 10$. ModelAnalyzer then concludes that the test instance might be wrongly classified, since a branch that classifies the instance as +1 could be reached.

Let us now consider the test instance (5, 11). Such instance is classified as +1 by the decision tree in Figure 1. The attacker cannot fool the classifier to mispredict such instance, since in our model the second feature cannot be attacked, as it is outside the bounds of the preconditions of the attacker's rewriting rules and the decision tree will always classify it as +1.

If we analyze the attacker and the instance by applying trace partitioning, ModelAnalyzer is in the position to conclude that such test instance cannot be wrongly classified, since the value of x_2 will always be 11. In fact, the bounds of the attacker are encoded as an if-then-else statement. If the value of the feature is inside such bounds, then the attacker modifies this value. Therefore, if we partition the abstract state w.r.t. the condition of this if-then-else statement (as we did in the example in Section 2.3), we infer a state where the feature is modified if it is inside the attacker bounds, and another state where it cannot be attacked if it is outside such bounds. Thank to this information, when we apply this abstraction to the instance (5, 11) we infer that the value of the second feature cannot be modified, and therefore the instance cannot be wrongly classified.

If instead we analyze the attacker without trace partitioning, we are not in the position to conclude that the second feature cannot be attacked, as we symbolically track only the bounds of the attacked features, but not the fact that they can only be attacked if their initial value was inside some given bounds. Therefore, for such instance trace partitioning is required in order to conclude that the instance cannot be wrongly classified.

5. Experimental Evaluation

We now report on an experimental evaluation of our approach. We first introduce our methodology and then describe the key experimental results in terms of precision and efficiency on public datasets.

5.1. Methodology

We evaluate our proposal on three public datasets: Census, House and Wine, which are described in Section 5.2. Our methodology includes multiple steps. We start with a preliminary *threat modeling* phase, where we define the attacker's capabilities by means of a set of rewriting rules R and a set of possible budgets $\{K_1, \ldots, K_n\}$, as explained in Section 3.1. Our attackers are primarily designed to perform an experimental evaluation of ML-Cert, yet they are representative of plausible attack scenarios which do not fit traditional distance-based models and are readily supported by the expressiveness of our threat model.

Datasets are divided into $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ by using a 90-10 splitting with stratified sampling (though we use an 80-20 splitting for the smaller Wine dataset). We first train a classifier on $\mathcal{D}_{train}$ using the popular

Table 1
Details of Hyper-Parameter Tuning

Model type	Hyper-parameters
Decision tree	Number of leaves: $\{2^1, 2^2, \ldots, 2^{10}\}$
Logistic regression	Regularization: $\{10^{-3}, 10^{-2}, \ldots, 10^3\}$
Neural network	Number of hidden neurons: $\{4, 8, 12\}$

`scikit-learn` library, performing hyper-parameter tuning through cross-validation on $\mathcal{D}_{train}$. We use the *error rate*, i.e., the fraction of wrong predictions among all the performed predictions, as the loss function $\mathcal{L}$ used to estimate the performance of the models. Minimizing the error rate is thus equivalent to maximizing the *accuracy* of the models, a standard measure amounting to the fraction of correct predictions out of all predictions, i.e., accuracy can be computed by subtracting the error rate $\mathcal{L}$ from 1. Table 1 shows the tested hyper-parameters for the different trained models. Only the best-performing models for each dataset according to cross-validation are included in our experimental evaluation.

We then evaluate the model resilience to attacks against each attacker $A = (R, K_i)$ on $\mathcal{D}_{test}$. Ideally, we would like to compute the actual value of $\mathcal{L}^A$ (Equation 1) for each model, so as to assess the quality of its over-approximation computed by ML-Cert. Unfortunately, computing $\mathcal{L}^A$ by enumerating all the possible attacks is intractable, which is the key motivation of our work. To work around this problem, we reuse the certification technique for decision trees given in [11], which is the only available solution for attackers expressed using rewriting rules. In particular, the algorithm exploits the possible thresholds of the decision tree to compute $\mathbb{A}(\mathbf{x}_i)$, the set of *representative* attacks against the decision tree, for each instance $\mathbf{x}_i$ in $\mathcal{D}_{test}$. This is a comparatively small subset of the attacks $A(\mathbf{x}_i)$, which suffices to detect the successful evasions attacks without losing soundness or precision. We observe and we experimentally confirm that computing even the representative attacks does not scale in general, which motivates the need for approximated analyses like ours. For simplicity, we reuse the same set of representative attacks also in the evaluation of logistic regression and neural networks. Note that this might just *penalize* the evaluation of ML-Cert, since $\mathbb{A}(\mathbf{x}_i)$ may only represent a subset of $A(\mathbf{x}_i)$ for logistic regression and neural networks, yielding an estimate of $\mathcal{L}^A$ which is *lower* than its actual value (and we would like the over-approximation computed by ML-Cert to be as close as possible to $\mathcal{L}^A$).

In the end, we evaluate the performance of ML-Cert by classifying each $(\mathbf{x}_i, y_i) \in \mathcal{D}_{test}$ as follows:

- *True Positive* (*TP*): ML-Cert states that the instance $\mathbf{x}_i$ can be misclassified upon attack and this conclusion is correct.
- *False Positive* (*FP*): ML-Cert states that the instance $\mathbf{x}_i$ can be misclassified upon attack, but this conclusion is wrong.
- *True Negative* (*TN*): ML-Cert states that the instance $\mathbf{x}_i$ cannot be misclassified upon attack and this conclusion is correct.
- *False Negative* (*FN*): ML-Cert states that the instance $\mathbf{x}_i$ cannot be misclassified upon attack, but this conclusion is wrong.

Since our analysis is sound, we cannot have *FN*. We then assess the quality of ML-Cert by computing its *False Positive Rate FPR* and *False Discovery Rate FDR*:

$$FPR = \frac{FP}{FP + TN}, \qquad FDR = \frac{FP}{FP + TP}.$$

Table 2
Properties of datasets used in the experiments

Dataset	#Instances	#Features	Maj. class
Census	29169	51	0.75
House	21613	19	0.51
Wine	6497	12	0.63

Finally, we evaluate the efficiency of ML-Cert, in particular focusing on a comparison against the time required for the computation of the set of the representative attacks discussed above. This yields a fair comparison in the case of decision trees, since the set of representative attacks covers all the possible evasion attacks. For logistic regression and neural networks we just focus on the running times of ML-Cert and assess whether they are appropriate for practical use, since we lack a sound analysis baseline for attackers expressed using rewriting rules.

5.2. Datasets and Attackers

We perform our experiments on three publicly available datasets, whose key statistics are shown in Table 2. The preconditions of the rewriting rules and the magnitude of the perturbations have been set after a preliminary data exploration step, based on the observed data distribution in the dataset. A real-world application of our analysis technique would require input from domain experts to define the relevant threats, which is beyond the scope of our evaluation. We consider different values of the attacker's budget: 10, 20, 30, 40, 50, 60. For the House dataset, we stop the evaluation at 40, since the computation of the set of the representative attacks (our ground truth) becomes intractable after that.

5.2.1. Census

The Census[3] dataset includes demographic information about American citizens. The prediction task is estimating whether the income of a citizen is above 50,000$ per year. For this dataset, we define four rewriting rules:

- cost 5: if the capital gain is in [0,100000], it can be raised by 200;
- cost 5: if the capital loss is in [0,100000], it can be lowered by 200;
- cost 10: if the number of work hours is in [0,40], it can be raised by 1;
- cost 10: if the age is in [0,40], it can be raised by 1.

5.2.2. House

The House[4] dataset contains house sale prices for the King County area. The prediction task is inferring whether a house costs at least as the median house price. For this dataset, we define four rewriting rules:

- cost 5: if the square footage of the living space of the house is in [0,3000], it can be increased by 50;
- cost 5: if the square footage of the land space is in [0,2000], it can be increased by 50;
- cost 5: if the average square footage of the living space of the 15 closest houses is in [0,2000], it can be increased by 50;
- cost 5: if the construction year is in [1900,1970], it can be increased by 10.

[3] http://archive.ics.uci.edu/ml/machine-learning-databases/adult
[4] https://www.kaggle.com/harlfoxem/housesalesprediction

Table 3
Accuracy results for decision trees (a star marks approximated results)

Dataset	Budget	$\mathcal{L}$	$\mathcal{L}^A$	ML-Cert	FPR	FDR
Census	10	0.14	0.16	0.16	0.00	0.00
	20	0.14	0.17	0.17	0.00	0.00
	30	0.14	0.17	0.17	0.00	0.00
	40	0.14	0.17	0.17	0.00*	0.00*
	50	0.14	0.18	0.18	0.00	0.00
	60	0.14	0.18	0.18	0.00*	0.00*
House	10	0.10	0.11	0.11	0.00	0.00
	20	0.10	0.13	0.13	0.00*	0.00*
	30	0.10	0.14	0.14	0.00	0.00
	40	0.10	0.16	0.16	0.00*	0.00*
Wine	10	0.23	0.27	0.27	0.00*	0.00*
	20	0.23	0.30	0.30	0.00*	0.00*
	30	0.23	0.34	0.34	0.00*	0.00*
	40	0.23	0.36	0.36	0.00*	0.00*
	50	0.23	0.37	0.37	0.00*	0.00*
	60	0.23	0.38	0.38	0.00*	0.00*

5.2.3. Wine

The Wine[5] dataset represents different types of wines. The prediction task is detecting whether a wine has quality score at least 6 on a scale 0–10. For this dataset, we define four rewriting rules:

- cost 2: if the residual sugar is in [2,4], it can be lowered by 0.01;
- cost 5: if the alcohol level is in [0,11], it can be increased by 0.01;
- cost 5: if the volatile acidity is in [0,1], it can be lowered by 0.01;
- cost 5: if the free sulfur dioxide is in [20,40], it can be lowered by 0.1.

5.3. Quality of the Analysis

We start by presenting the results for decision trees in Table 3. For these models we are able to compute the actual value of $\mathcal{L}^A$ by means of the representative attack approach, hence we have a ground truth and we can perform a fully reliable assessment. The experimental evaluation yields virtually no false positives: we only identified 9 false positives across all datasets and budgets, i.e., the average number of false positives across all the experiments is less than 1. This implies that both *FPR* and *FDR* are always very close to 0 (and actually 0 in several cases).

We now focus on the analysis of logistic regression and neural networks. For these models, the set of representative attacks computed over decision trees might not cover all the possible attacks, which means that the value of $\mathcal{L}^A$ might be under-estimated. Though this might penalize ML-Cert by over-estimating the number of false positives, we show that this is not a problem in practice, because ML-Cert achieves excellent results despite this penalization. We start by presenting the results for the logistic regression models in Table 4. Two observations are noteworthy on this experiment. First, the quality of the analysis performed by ML-Cert is even better than for the decision tree models, because just one false positive was reported across all datasets and budgets: this explains why both *FPR* and *FDR* are always very close to 0. Moreover, we note that the accuracy of logistic regression is worse than that of the decision tree

[5] https://www.openml.org/data/get_csv/49817/wine_quality.arff

Table 4
Accuracy results for logistic regression (a star marks approximated results)

Dataset	Budget	$\mathcal{L}$	$\mathcal{L}^A$	ML-Cert	FPR	FDR
Census	10	0.15	0.17	0.17	0.00	0.00
	20	0.15	0.19	0.19	0.00	0.00
	30	0.15	0.20	0.20	0.00*	0.00*
	40	0.15	0.21	0.21	0.00	0.00
	50	0.15	0.22	0.22	0.00	0.00
	60	0.15	0.23	0.23	0.00	0.00
House	10	0.15	0.15	0.15	0.00	0.00
	20	0.15	0.16	0.16	0.00	0.00
	30	0.15	0.17	0.17	0.00	0.00
	40	0.15	0.17	0.17	0.00	0.00
Wine	10	0.27	0.28	0.28	0.00	0.00
	20	0.27	0.29	0.29	0.00	0.00
	30	0.27	0.30	0.30	0.00	0.00
	40	0.27	0.31	0.31	0.00	0.00
	50	0.27	0.32	0.32	0.00	0.00
	60	0.27	0.33	0.33	0.00	0.00

models on all datasets, yet logistic regression fares better under attack on the Wine dataset. We believe the reason for this is the simplicity of logistic regression, which yields more regular decision boundaries which are harder to evade.

Finally, Table 5 shows the results for the neural network models. Here we observe again near-perfect analysis precision on the House and Wine datasets: only one false positive was reported overall. The picture is a bit different on the Census dataset, which turned out to be more challenging to analyze; however, *FPR* is at most 0.02 and *FDR* is at most 0.05. In particular, we remark that a *FPR* of 0.10 is considered a state-of-the-art reference for static analysis techniques [28]. Though *FDR* is slightly higher than *FPR*, this is not a major problem in our application setting: contrary to what happens in traditional program analysis, where users are forced to investigate all false alarms to identify possible bugs, here we are rather interested in the aggregated analysis results, i.e., the final over-approximation of $\mathcal{L}^A$. The table shows that the difference between $\mathcal{L}^A$ and its over-approximation computed by ML-Cert is at most 0.01, which is negligible in practice.

5.4. Efficiency of the Analysis

To show the efficiency of our approach, we compare in Figure 6 the running times of ML-Cert on the decision tree models against the time taken to compute the full set of the representative attacks. It is possible to observe that the two curves exhibit completely different trends. The time taken to construct the representative attacks shows an *exponential* trend: the approach is efficient and feasible when the attacker's budget is low, but blows up to intractability very quickly. For example, each increase in the attacker's budget multiplies the execution time of a 2x-3x factor in the case of the Census dataset and 2.4 hours of computation are needed for budget 60. Conversely, the execution time of ML-Cert is not largely affected by the attacker's budget and only 46 minutes of computation are needed for budget 60, with virtually no false positive. In the case of the House dataset, computing the set of the representative attacks is basically infeasible: even for small budgets, the running time is remarkably high, due to the fact that the trained decision tree uses many different thresholds, which makes the number of representative

Table 5
Accuracy results for neural networks (a star marks approximated results)

Dataset	Budget	$\mathcal{L}$	$\mathcal{L}^A$	ML-Cert	*FPR*	*FDR*
Census	10	0.15	0.17	0.17	0.00*	0.02*
	20	0.15	0.18	0.18	0.00*	0.01*
	30	0.15	0.19	0.19	0.00*	0.01*
	40	0.15	0.21	0.21	0.00*	0.01*
	50	0.15	0.22	0.23	0.01*	0.02*
	60	0.15	0.24	0.25	0.02*	0.05*
House	10	0.10	0.10	0.10	0.00	0.00
	20	0.10	0.11	0.11	0.00	0.00
	30	0.10	0.11	0.11	0.00	0.00
	40	0.10	0.12	0.12	0.00	0.00
Wine	10	0.25	0.26	0.26	0.00	0.00
	20	0.25	0.27	0.27	0.00	0.00
	30	0.25	0.28	0.28	0.00	0.00
	40	0.25	0.30	0.30	0.00	0.00
	50	0.25	0.31	0.31	0.00	0.00
	60	0.25	0.32	0.32	0.00*	0.00*

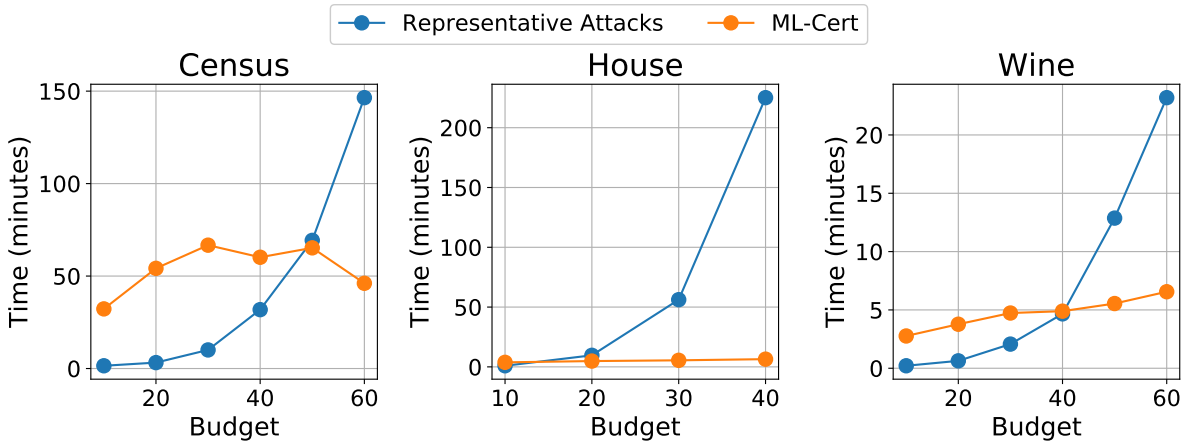

Fig. 6. Running time of ML-Cert against the enumeration of representative attacks for the decision tree model.

attacks blow up. Computing the representative attacks for budget 40 takes almost 4 hours, while ML-Cert can perform the same analysis in around 6 minutes. Also the Wine dataset shows similar figures, though the running times there are lower due to its smaller size. This confirms that brute-force approaches based on the exhaustive enumeration of the representative attacks do not scale, yet luckily they can be replaced by more efficient abstraction techniques with near-perfect precision.

To complete our analysis, we show in Figure 7 the analysis times of ML-Cert on all models, for all datasets and budgets. The figure confirms that it is possible to efficiently certify the security of ML models using program analysis, in particular in the case of decision trees and logistic regression models: in the worst case, verification takes around 67 minutes for those models. Neural networks are significantly harder to analyze than decision trees and logistic regression, yet their verification on the House and Wine dataset can be performed in less than 35 minutes in the worst case, which is perfectly

Fig. 7. Running time of ML-Cert for all datasets, budgets and models.

Table 6
Average analysis time in seconds for a single instance of the dataset

Dataset	Budget	Decision Tree	Logistic Regression	Neural Network
Census	10	0.66	0.11	3.90
	20	1.11	0.25	5.00
	30	1.37	0.46	4.37
	40	1.24	0.62	4.80
	50	1.34	0.51	6.05
	60	0.95	0.40	8.63
House	10	0.10	0.04	0.71
	20	0.13	0.06	0.76
	30	0.15	0.06	0.80
	40	0.17	0.07	0.93
Wine	10	0.13	0.03	0.29
	20	0.17	0.06	0.35
	30	0.22	0.08	0.47
	40	0.23	0.09	0.62
	50	0.26	0.11	0.66
	60	0.30	0.13	0.75

appropriate for practical use. The Census dataset is more computationally expensive to deal with, since verification takes around 7 hours for budget 60. Nevertheless, the analysis only needs to be performed once, leading to an acceptable price to pay for a certified security proof.

Table 6 reports the average analysis time per instance in the different settings. The table shows that the average analysis time per instance is consistently less than one second for all models and datasets, with the notable exception of the neural network model trained over Census, where the analysis still runs in the order of a few seconds per instance.

5.5. Additional Experiments

The previous experiments showed that ML-Cert can efficiently compute a precise over-approximation of the loss under attack $\mathcal{L}^A$ in a wide range of settings. However, they do not provide insights on several

Table 7
Effects of trace partitioning on average analysis time (in seconds) and number of attacks reported by ML-Cert. Best results in boldface.

Dataset	Budget	Trace Partitioning	Decision Tree		Logistic Regression		Neural Network	
			Time	Attacks	Time	Attacks	Time	Attacks
Census	10	yes	0.24	**462**	0.09	**516**	**2.41**	**510**
		no	**0.21**	**462**	**0.05**	**516**	2.53	**510**
	60	yes	0.73	**636**	0.29	**684**	**8.27**	**644**
		no	**0.29**	**636**	**0.06**	852	8.55	722
House	10	yes	**0.13**	**246**	0.08	**335**	**0.50**	**202**
		no	0.17	263	**0.05**	457	1.29	254
	40	yes	**0.26**	**364**	0.13	**374**	**0.85**	**257**
		no	0.39	419	**0.11**	846	9.50	736
Wine	10	yes	0.11	**361**	0.04	**363**	**0.36**	**321**
		no	**0.10**	370	**0.02**	364	0.55	324
	60	yes	0.26	**500**	0.15	**429**	**0.57**	**399**
		no	**0.20**	523	**0.03**	443	0.71	414

important aspects that we investigate in the present section. We focus on the impact of trace partitioning, the generality with respect to different attacker models and the scalability of the analysis.

5.5.1. Impact of Trace Partitioning

Trace partitioning is expected to have a positive impact on the quality of the analysis, however it might negatively affect its efficiency. To quantify the actual impact of trace partitioning, we compare the number of instances marked as susceptible to evasion attacks, as well as the analysis times when trace partitioning is activated or not. Since trace partitioning does not affect soundness, an increase in the number of instances susceptible to evasion attacks can only be attributed to false positives, hence reporting a lower number of possible attacks is preferable. The results of our experiments are shown in Table 7. The considered models include a decision tree with 256 leaves, a logistic regression model and a neural network with 8 neurons; for each model we perform two experiments, using the minimum and maximum attacker's budget respectively.

The results show that trace partitioning is very effective at reducing the number of false positives of the analysis, in some cases dramatically: for example, activating trace partitioning for the decision tree analysis on the House dataset removes 55 false positives (for budget 40) and the improvement is even higher for logistic regression and neural network models, where hundreds of false positives are pruned away. In most cases, the benefits of trace partitioning on the quality of the analysis are more apparent for higher budgets, likely because the single approximation computed for the attacker without trace partitioning becomes more imprecise due to the higher number of possible attacks. As to efficiency, we observe an interesting phenomenon, i.e., trace partitioning also reduces the analysis time in several cases, thus leading to a win-win situation where the analysis is both more precise and more efficient - this happened for all the neural network models. We are not the first to observe such a fact: also the original authors of trace partitioning experimentally observed that disabling trace partitioning may lead to a much higher number of fixpoint iterations, leading to larger analysis times [29]. In our setting, we observe that the average analysis time per instance for the neural network model trained on the House dataset drops from 9.50 seconds to 0.85 seconds when trace partitioning is activated (for budget 40). Remarkably, even when trace partitioning has a negative impact on efficiency, we observe that it is mild: we never observed cases where trace partitioning turned an efficient analysis into an intractable one.

5.5.2. Distance-based Attackers

Our experimental analysis focused on an expressive threat model based on rewriting rules [11], however prior work mostly focused on distance-based attackers [13, 14]. Since our approach is based on an encoding as a traditional imperative program, we can readily analyze distance-based attackers, e.g., based on the infinity-norm L_∞, which expresses a maximum perturbation for each attacked feature. We now measure the quality and the efficiency of our analysis for such attackers. We use the same datasets and attacked features as before, considering as maximum perturbation the same perturbation assumed for a single application of the rewriting rules, e.g., if the attacker can add 0.5 to feature 1 according to a rewriting rule, we assume feature 1 can be perturbed by ± 0.5.

A first observation we make is that the L_∞-attacker discussed above is intuitively simpler to analyze than attackers expressed by means of rewriting rules, because they can be approximated by means of interval-based abstractions, rather than requiring the use of complex domains like Polyhedra. In particular, the Box implementation of the Interval domain provided by Apron sounds appropriate to support a precise yet efficient analysis. Table 8 reports the analysis times per instance for all datasets under the L_∞-attacker model, using both the Box and Polka domains. The table also includes the number of instances marked as susceptible to evasion attacks: since both domains are sound, an increase in the number of such instances can only be attributed to false positives, hence reporting a lower number of possible attacks is preferable. The considered models include a decision tree with 256 leaves, a logistic regression model and a neural network with 8 neurons.

As we can see, the analysis times are dramatically lower for the Box domain, since the average analysis time per instance is always way less than 0.1 seconds. This means that the model performance on a test set with 1000 instances can be assessed in less than 2 minutes. In terms of the quality of the analysis, we observe that Box behaves comparably to Polka in most cases. This happens for all decision trees and logistic regression models, as well as for the neural network trained on Wine. There are two cases though where the quality of the analysis considerably downgrades when using Box, i.e., the neural networks trained on Census and House. For example, the number of reported attacks for the neural network trained on House increases from 277 to 392 when using the Box domain. Two observations are in order here: first, the approximation of the loss under attack $\mathcal{L}^A$ just increases from 0.13 to 0.18, which is still a useful conservative approximation of the performance of the model under attack. Second, one is not forced to rely on a single abstract domain for the analysis. Indeed, an appealing analysis strategy could be based on a two-step approach, where instances are first analyzed using Box, and only those instances which are marked as potentially susceptible to evasion attacks are additionally analyzed using Polka. This would lead to a significant speedup in the analysis times, e.g., in the case of the neural network model trained on House we would be required to run the expensive Polka analysis only on 392 instances, rather than on the 2161 instances in the full test set. This would reduce the analysis time of the test set tfrom roughly one hour to approximately 12 minutes.

5.5.3. Scalability of the Analysis

The last experiment that we carry out is designed to better understand the *scalability* of our analysis technique, i.e., we measure how the average analysis time per instance changes when increasing the model size. In particular, we analyze decision trees with different number of leaves ($2^7, 2^8, 2^9, 2^{10}$) and neural networks with different number of hidden neurons ($4, 8, 16, 32$). For each dataset, we carry out different analyses, setting a timeout of 30 seconds per instance. In particular we consider both attackers based on rewriting rules, with minimum and maximum budget respectively, and distance-based attackers, which we analyze using both Polka and Box. The results are shown in the plot of Figure 8.

Table 8
Effects of abstract domain on average analysis time (in seconds) and number of attacks reported by ML-Cert. Best results in boldface.

Dataset	Abstract Domain	Decision Tree		Logistic Regression		Neural Network	
		Time	Attacks	Time	Attacks	Time	Attacks
Census	Polka	0.27	**492**	0.07	**543**	3.56	**552**
	Box	**0.02**	**492**	**0.01**	**543**	**0.02**	642
House	Polka	0.23	**266**	0.06	**461**	1.74	**277**
	Box	**0.03**	267	**0.01**	**461**	**0.02**	392
Wine	Polka	0.13	**372**	0.03	**372**	0.61	**326**
	Box	**0.02**	**372**	**0.01**	**372**	**0.01**	329

Our experiment shows that the analysis of decision tree models is very scalable, since increasing the number of leaves has only a limited impact on performance. Even for the Census dataset, which is the most challenging in terms of analysis times, we observe that the analysis time increases sub-linearly with respect to the number of leaves: instances are analyzed in 0.21 seconds on average for a decision tree with 256 leaves and the analysis time increases to 0.31 seconds for a decision tree with 1024 leaves, for the minimum budget. A similar figure applies to the maximum budget, where the average analysis time increases from 0.64 to 1.01 seconds. Neural networks, instead, are harder to analyze and pose bigger threats to scalability when considering attackers based on rewriting rules with a large budget. In particular, when considering the maximum budget, we observe that neural networks with 32 neurons are challenging to analyze for all datasets, and even more so for Census. However, the analysis complexity seems more related to the attacker rather than to the ML model itself, because traditional L_∞-attackers can be efficiently analyzed using the Box domain even for neural network models. For such cases, instances can always be analyzed in less than 0.1 seconds on average: the hardest setting is the Census dataset, where instances can be analyzed in 0.07 seconds on average in the case of a neural network with 32 neurons, meaning that the model performance on a test set with thousands of instances can be assessed in a matter of minutes. Also the trend with respect to the number of neurons is reassuring.

6. Related Work

Adversarial machine learning is a hot topic nowadays and several papers studied the problem of the security certification of ML models. As explained in the introduction, no existing proposal is at the same time sound, efficient and general enough to cover different types of ML models and attackers. ML-Cert shows that soundness, efficiency and high generality can be achieved together by leveraging state-of-the-art program analysis techniques, at least for relatively simple ML models and datasets. Here we review relevant related work on the verification of decision trees, logistic regression and neural networks.

6.1. Decision Trees

Ranzato and Zanella proposed a technique to analyze the security of decision trees and decision tree ensembles against evasion attacks using abstract interpretation, which is close to our proposal [14]. However, their approach assumes attackers who admit a simple mathematical characterization as a set of perturbations, e.g., based on distances. In particular, their soundness theorem relies on the hypothesis that, for each test instance $\mathbf{x}$, one has $A(\mathbf{x}) \subseteq \gamma(\alpha(\{\mathbf{x}\}))$, i.e., the abstraction of $\mathbf{x}$ must cover all the

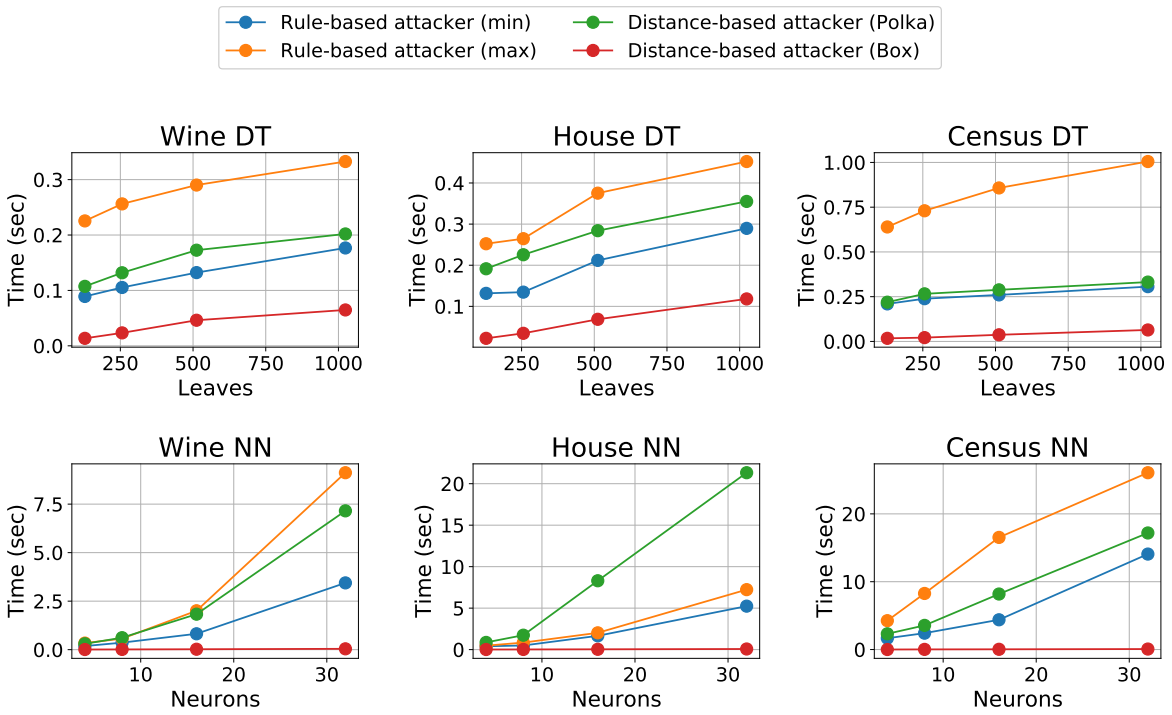

Fig. 8. Scalability analysis of ML-Cert

possible attacks. Checking this condition for distance-based attackers is straightforward, yet this is computationally infeasible in general. For example, in the case of the rewriting rules we considered, $A(\mathbf{x})$ is unknown *a priori*, but it is induced by the application of the rules. Indeed, their tool *silva* only supports attackers based on the infinity-norm L_∞, which generally falls short of representing realistic threats, while our approach is sufficiently general to be applied to attackers modeled as arbitrary imperative programs. On the other hand, contrary to ML-Cert, their tool can handle ensembles of decision trees, which are important for real-world practical applications, and implements a *complete* analysis technique, i.e., no false positive can be produced. This is certainly a desirable property, although it is enabled by the restriction to L_∞-attackers, who can be precisely abstracted by means of hyper-rectangles using intervals. As a matter of fact, the vast majority of static analyses lack completeness.

Other approaches to the verification of decision trees are not based on abstract interpretation. Einzinger et al. use SMT solving to verify the robustness of gradient-boosted models [12]. Their approach requires to explicitly encode the set of attacks $A(\mathbf{x})$ in closed form, which is only easily doable for artificial distance-based attackers. Moreover, SMT solving suffers from scalability issues, which required the authors to develop custom optimizations to make their approach practical. It is unclear whether this line of work can be adapted and scale to more expressive attackers or not, also because their tool is not publicly available. Other notable work includes the robustness verification algorithm by Chen et al. [13], which only works for attackers based on the infinity-norm L_∞, and the abstraction-refinement approach by Törnblom and Nadjm-Tehrani [10], which is not proved sound, yet was applied to decision tree ensembles as well. The experimental evaluation by Ranzato and Zanella showed that this proposal

is less effective than theirs, that we discussed above, with respect to both the quality and the efficiency of the analysis.

Finally, it is worth mentioning adversarial learning algorithms which train decision trees more resilient to evasion attacks by construction [11, 30–32]. This line of work is orthogonal to the security verification of decision trees, i.e., our approach can be applied to estimate the robustness guarantees of trees trained using such algorithms.

6.2. Logistic Regression

We are not aware of papers which studied the security verification of logistic regression models. However, logistic regression is a (generalized) linear model, which is a class of models which received attention by the adversarial ML community. In particular, Lowd and Meek first coined the term "adversarial learning" in the context of linear classifiers [33]. The security of linear classifiers has been specifically studied in an experimental evaluation by Demontis et al. [34] and in a theoretical analysis by Fawzi, Fawzi and Frossard [35]. However, none of these papers proposed certification techniques, in contrast to our work.

6.3. Neural Networks

Many papers proposed techniques to verify the security of deep neural networks: representative examples include a number of papers recently published at reputable conferences [36–40]. These papers showed that neural networks can be soundly and efficiently analyzed, also using program analysis techniques like ours, however they only focused on simple distance-based attackers as those arising in image recognition, which we confirmed to be very easy to analyze even with off-the-shelf program analysis libraries - instances can be analyzed on average on less than 0.1 seconds in all cases, using intervals. Remarkably, ML-Cert shows that existing program analysis techniques can be readily applied to a wide range of ML models and attacks, while keeping the traditional soundness and efficiency properties of abstract interpretation. We acknowledge that the neural network architecture considered in the present work is simple and did not require aggressive optimizations for scalability, however our key goal was showing the generality of our approach, which can be straightforwardly applied to arbitrary network architectures and different attackers. We expect more work and tailored optimizations, e.g., novel abstract domains, are needed to make the analysis scale to more complex network architectures and different activation functions.

7. Conclusion

We proposed a technique to certify the security of ML models against evasion attacks by leveraging the abstract interpretation framework and we implemented it in a new tool called ML-Cert. ML-Cert is the first solution which is at the same time sound, efficient and general enough to deal with different types of ML models and sophisticated attackers represented as arbitrary imperative programs. Our experimental evaluation on public datasets showed that our technique is also very precise, yielding a negligible number of false positives on most cases that can be fully analyzed using a competitor approach [11], which quickly blows to intractability.

We foresee several avenues for future work. First, we plan to extend our approach to the analysis of regression tasks and tree ensembles: though this sounds straightforward from an engineering perspective,

we want to analyze the precision and the efficiency of the proposed solution in such settings. Moreover, we plan to investigate techniques to automatically infer the minimal attacker's budget required to induce a given error rate on the test set, so as to efficiently provide security analysts with this useful information. Finally, we plan to further extend ML-Cert with support for different abstract domains, so as to provide security analysts with the ability to fine-tune the trade-off between analysis precision and analysis efficiency on their datasets.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I.J. Goodfellow and R. Fergus, Intriguing properties of neural networks, in: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2014. http://arxiv.org/abs/1312.6199.

[2] L. Demetrio, S.E. Coull, B. Biggio, G. Lagorio, A. Armando and F. Roli, Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection, *ACM Trans. Priv. Secur.* **24**(4) (2021), 27:1–27:31. doi:10.1145/3473039.

[3] J. Su, D.V. Vargas and K. Sakurai, One Pixel Attack for Fooling Deep Neural Networks, *IEEE Trans. Evol. Comput.* **23**(5) (2019), 828–841. doi:10.1109/TEVC.2019.2890858.

[4] M. Sharif, S. Bhagavatula, L. Bauer and M.K. Reiter, Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition, in: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers and S. Halevi, eds, ACM, 2016, pp. 1528–1540. doi:10.1145/2976749.2978392.

[5] B. Biggio and F. Roli, Wild patterns: Ten years after the rise of adversarial machine learning, *Pattern Recognit.* **84** (2018), 317–331. doi:10.1016/j.patcog.2018.07.023.

[6] P. Cousot and R. Cousot, Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, in: *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, R.M. Graham, M.A. Harrison and R. Sethi, eds, ACM, 1977, pp. 238–252. doi:10.1145/512950.512973.

[7] P. Cousot and R. Cousot, Systematic Design of Program Analysis Frameworks, in: *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, A.V. Aho, S.N. Zilles and B.K. Rosen, eds, ACM Press, 1979, pp. 269–282. doi:10.1145/567752.567778.

[8] T. Dreossi, S. Jha and S.A. Seshia, Semantic Adversarial Deep Learning, in: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, H. Chockler and G. Weissenbacher, eds, Lecture Notes in Computer Science, Vol. 10981, Springer, 2018, pp. 3–26. doi:10.1007/978-3-319-96145-3_1.

[9] I.J. Goodfellow, P.D. McDaniel and N. Papernot, Making machine learning robust against adversarial inputs, *Commun. ACM* **61**(7) (2018), 56–66. doi:10.1145/3134599.

[10] J. Törnblom and S. Nadjm-Tehrani, An Abstraction-Refinement Approach to Formal Verification of Tree Ensembles, in: *Computer Safety, Reliability, and Security - SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings*, A.B. Romanovsky, E. Troubitsyna, I. Gashi, E. Schoitsch and F. Bitsch, eds, Lecture Notes in Computer Science, Vol. 11699, Springer, 2019, pp. 301–313. doi:10.1007/978-3-030-26250-1_24.

[11] S. Calzavara, C. Lucchese and G. Tolomei, Adversarial Training of Gradient-Boosted Decision Trees, in: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, W. Zhu, D. Tao, X. Cheng, P. Cui, E.A. Rundensteiner, D. Carmel, Q. He and J.X. Yu, eds, ACM, 2019, pp. 2429–2432. doi:10.1145/3357384.3358149.

[12] G. Einziger, M. Goldstein, Y. Sa'ar and I. 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*, AAAI Press, 2019, pp. 2446–2453. doi:10.1609/aaai.v33i01.33012446.

[13] H. Chen, H. Zhang, S. Si, Y. Li, D.S. Boning and C. Hsieh, Robustness Verification of Tree-based Models, in: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E.B. Fox and R. Garnett, eds, 2019, pp. 12317–12328. https://proceedings.neurips.cc/paper/2019/hash/cd9508fdaa5c1390e9cc329001cf1459-Abstract.html.

[14] F. Ranzato and M. Zanella, Abstract Interpretation of Decision Tree Ensemble Classifiers, in: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 5478–5486. https://ojs.aaai.org/index.php/AAAI/article/view/5998.

[15] S. Calzavara, P. Ferrara and C. Lucchese, Certifying Decision Trees Against Evasion Attacks by Program Analysis, in: *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II*, L. Chen, N. Li, K. Liang and S.A. Schneider, eds, Lecture Notes in Computer Science, Vol. 12309, Springer, 2020, pp. 421–438. doi:10.1007/978-3-030-59013-0_21.

[16] X. Rival and L. Mauborgne, The trace partitioning abstract domain, *ACM Trans. Program. Lang. Syst.* **29**(5) (2007), 26. doi:10.1145/1275497.1275501.

[17] P. Tan, M.S. Steinbach and V. Kumar, *Introduction to Data Mining*, Addison-Wesley, 2005. ISBN 0-321-32136-7. http://www-users.cs.umn.edu/%7Ekumar/dmbook/.

[18] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto and F. Roli, Evasion Attacks against Machine Learning at Test Time, in: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, H. Blockeel, K. Kersting, S. Nijssen and F. Zelezný, eds, Lecture Notes in Computer Science, Vol. 8190, Springer, 2013, pp. 387–402. doi:10.1007/978-3-642-40994-3_25.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, Towards Deep Learning Models Resistant to Adversarial Attacks, in: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018. https://openreview.net/forum?id=rJzIBfZAb.

[20] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, 1984. ISBN 0-534-98053-8.

[21] D. Jurafsky and J.H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*, Prentice Hall series in artificial intelligence, Prentice Hall, Pearson Education International, 2009. ISBN 9780135041963. https://www.worldcat.org/oclc/315913020.

[22] A. Miné, The octagon abstract domain, *High. Order Symb. Comput.* **19**(1) (2006), 31–100. doi:10.1007/s10990-006-8609-1.

[23] P. Cousot and N. Halbwachs, Automatic Discovery of Linear Restraints Among Variables of a Program, in: *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, A.V. Aho, S.N. Zilles and T.G. Szymanski, eds, ACM Press, 1978, pp. 84–96. doi:10.1145/512760.512770.

[24] B. Jeannet and A. Miné, Apron: A Library of Numerical Abstract Domains for Static Analysis, in: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, A. Bouajjani and O. Maler, eds, Lecture Notes in Computer Science, Vol. 5643, Springer, 2009, pp. 661–667. doi:10.1007/978-3-642-02658-4_52.

[25] C.M. Bishop, *Pattern recognition and machine learning, 5th Edition*, Information science and statistics, Springer, 2007. ISBN 9780387310732. https://www.worldcat.org/oclc/71008143.

[26] L. Breiman, Random Forests, *Mach. Learn.* **45**(1) (2001), 5–32. doi:10.1023/A:1010933404324.

[27] J.H. Friedman, Greedy function approximation: a gradient boosting machine, *Annals of statistics* (2001), 1189–1232.

[28] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon and C. Jaspan, Lessons from building static analysis tools at Google, *Commun. ACM* **61**(4) (2018), 58–66. doi:10.1145/3188720.

[29] L. Mauborgne and X. Rival, Trace Partitioning in Abstract Interpretation Based Static Analyzers, in: *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, S. Sagiv, ed., Lecture Notes in Computer Science, Vol. 3444, Springer, 2005, pp. 5–20. doi:10.1007/978-3-540-31987-0_2.

[30] A. Kantchelian, J.D. Tygar and A.D. Joseph, Evasion and Hardening of Tree Ensemble Classifiers, in: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, M. Balcan and K.Q. Weinberger, eds, JMLR Workshop and Conference Proceedings, Vol. 48, JMLR.org, 2016, pp. 2387–2396. http://proceedings.mlr.press/v48/kantchelian16.html.

[31] H. Chen, H. Zhang, D.S. Boning and C. Hsieh, Robust Decision Trees Against Adversarial Examples, in: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, eds, Proceedings of Machine Learning Research, Vol. 97, PMLR, 2019, pp. 1122–1131. http://proceedings.mlr.press/v97/chen19m.html.

[32] S. Calzavara, C. Lucchese, G. Tolomei, S.A. Abebe and S. Orlando, Treant: training evasion-aware decision trees, *Data Min. Knowl. Discov.* **34**(5) (2020), 1390–1420. doi:10.1007/s10618-020-00694-9.

[33] D. Lowd and C. Meek, Adversarial learning, in: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, R. Grossman, R.J. Bayardo and K.P. Bennett, eds, ACM, 2005, pp. 641–647. doi:10.1145/1081870.1081950.

[34] A. Demontis, P. Russu, B. Biggio, G. Fumera and F. Roli, On Security and Sparsity of Linear Classifiers for Adversarial Settings, in: *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*, A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano and R.C. Wilson, eds, Lecture Notes in Computer Science, Vol. 10029, 2016, pp. 322–332. doi:10.1007/978-3-319-49055-7_29.

[35] A. Fawzi, O. Fawzi and P. Frossard, Analysis of classifiers' robustness to adversarial perturbations, *Mach. Learn.* **107**(3) (2018), 481–508. doi:10.1007/s10994-017-5663-3.

[36] S. Wang, K. Pei, J. Whitehouse, J. Yang and S. Jana, Formal Security Analysis of Neural Networks using Symbolic Intervals, in: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A.P. Felt, eds, USENIX Association, 2018, pp. 1599–1614. https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi.

[37] S. Wang, K. Pei, J. Whitehouse, J. Yang and S. Jana, Efficient Formal Safety Analysis of Neural Networks, in: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H.M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, eds, 2018, pp. 6369–6379. https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html.

[38] X. Huang, M. Kwiatkowska, S. Wang and M. Wu, Safety Verification of Deep Neural Networks, in: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, R. Majumdar and V. Kuncak, eds, Lecture Notes in Computer Science, Vol. 10426, Springer, 2017, pp. 3–29. doi:10.1007/978-3-319-63387-9_1.

[39] G. Katz, C.W. Barrett, D.L. Dill, K. Julian and M.J. Kochenderfer, Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, *CoRR* **abs/1702.01135** (2017). http://arxiv.org/abs/1702.01135.

[40] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri and M.T. Vechev, AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation, in: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, IEEE Computer Society, 2018, pp. 3–18. doi:10.1109/SP.2018.00058.