Watermarking Decision Tree Ensembles

Stefano Calzavara
Università Ca' Foscari Venezia
Venice, Italy
stefano.calzavara@unive.it

Lorenzo Cazzaro
Università Ca' Foscari Venezia
Venice, Italy
lorenzo.cazzaro@unive.it

Donald Gera
Università Ca' Foscari Venezia
Venice, Italy
892604@stud.unive.it

Salvatore Orlando
Università Ca' Foscari Venezia
Venice, Italy
orlando@unive.it

ABSTRACT

Protecting the intellectual property of machine learning models is a hot topic and many watermarking schemes for deep neural networks have been proposed in the literature. Unfortunately, prior work largely neglected the investigation of watermarking techniques for other types of models, including decision tree ensembles, which are a state-of-the-art model for classification tasks on non-perceptual data. In this paper, we present the first watermarking scheme designed for decision tree ensembles, focusing in particular on random forest models. We discuss watermark creation and verification, presenting a thorough security analysis with respect to possible attacks. We finally perform an experimental evaluation of the proposed scheme, showing excellent results in terms of accuracy and security against the most relevant threats.

1 INTRODUCTION

Machine learning models are pervasively used and are often considered intellectual property of the parties who have trained them. This is often a consequence of the incredible number of computational resources required for model training. For example, even a relatively small model like GPT-3 is estimated to cost around 5M dollars for training on the cloud [10]. This motivated a significant amount of research on *model watermarking*, in particular for deep neural networks [2, 11, 18]. A watermark is a piece of identifying information which is embedded into the model to claim copyright, without affecting accuracy too much with respect to the original model. Different watermarking schemes have been proposed with different properties, e.g., zero-bit watermarking [1, 9, 12–14, 16, 19] and multi-bit watermarking [5, 7, 8, 13, 15, 17, 18], based on the amount of information embedded in the watermark.

Although watermarking received a great deal of attention in the field of deep neural networks, it was not carefully investigated for other types of machine learning models for different reasons. First, some models are shallow in the sense that they are not over-parameterized and redundant, lacking room to effectively embed watermarks. Moreover, traditional machine learning models require less computational resources for training than deep neural networks. Yet, the process of collecting high-quality training data, cleaning them and in some cases even manual labeling them should be performed for any type of supervised learning algorithm. This process is generally time-consuming and expensive, thus making copyright protection of highly effective models trained over high-quality datasets an urgent practical need.

Contributions. In this paper, we contribute as follows:

(1) We present the first watermarking scheme designed for *decision tree ensembles*, which are state-of-the-art models for classification tasks on non-perceptual data.
(2) We discuss watermark creation, verification and security against a range of possible attacks.
(3) We perform an experimental evaluation of the proposed scheme on public datasets.

Our analysis shows convincing results in terms of accuracy and security against the most relevant threats. Watermarking entails an accuracy loss of two points at most with respect to the original model, watermark detection is ineffective for two possible attack strategies, watermark suppression is prevented by construction, and watermark forgery is either impossible or can be easily detected in practice.

2 BACKGROUND

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d-dimensional vector space of real-valued *features*. An *instance* $\vec{x} \in \mathcal{X}$ is a d-dimensional feature vector $\langle x_1, x_2, \ldots, x_d \rangle$ representing an object in the vector space $\mathcal{X}$. Each instance is assigned a class label $y \in \mathcal{Y}$ by an unknown function $f : \mathcal{X} \to \mathcal{Y}$. Supervised learning algorithms learn a *classifier* $g : \mathcal{X} \to \mathcal{Y}$ from a *training set* of correctly labeled instances $\mathcal{D}_{train} = \{(\vec{x}_i, f(\vec{x}_i))\}_i$, with the goal of approximating the target function f as accurately as possible. The performance of classifiers is assessed on a *test set* of correctly labeled instances $\mathcal{D}_{test} = \{(\vec{z}_i, f(\vec{z}_i))\}_i$, disjoint from the training set, yet drawn from the same data distribution.

In this paper, we focus on a specific class of supervised learning algorithms training traditional *binary decision trees* for either binary or multiclass classification [4]. Decision trees can be inductively defined as follows: a decision tree t is either a leaf $L(y)$ for some label $y \in \mathcal{Y}$ or an internal node $N(f \leq v, t_l, t_r)$, where $f \in \{1, \ldots, d\}$ identifies a feature, $v \in \mathbb{R}$ is a threshold for the feature, and t_l, t_r are decision trees (left and right child). Decision trees are learned by initially putting all the training set into the root of the tree and by recursively splitting leaves (initially: the root) by identifying the threshold therein leading to the best split of the training data, e.g., the one with the highest information gain, thus transforming the split leaf into a new internal node. At test time, the instance $\vec{x}$ traverses the tree t until it reaches a leaf $L(y)$, which returns the prediction y, denoted by $t(\vec{x}) = y$. Specifically, for each traversed tree node $N(f \leq v, t_l, t_r)$, $\vec{x}$ falls into the left sub-tree t_l if $x_f \leq v$, and into the right sub-tree t_r otherwise. To improve their performance, decision trees are often combined into an *ensemble* $T = \langle t_1, \ldots, t_m \rangle$, which aggregates individual tree predictions, e.g., by performing majority voting. Figure 1 shows an ensemble including $m = 2$ decision trees.

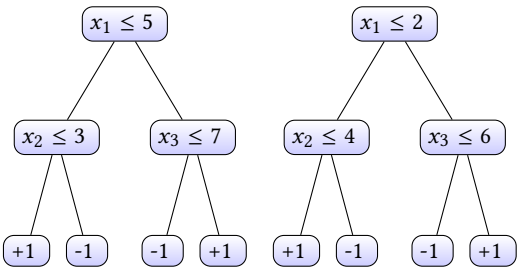


Figure 1: Example of a decision tree ensemble with two trees.

3 ENSEMBLE WATERMARKING

We first motivate our key design choices and we introduce the threat model considered in this work. We then present our watermarking scheme and security analysis.

3.1 Design Choices and Threat Model

We explain the key design choices and the threat model using the terminology of a recent survey [2]. Our watermark is embedded during the training phase by means of a *trigger set*, i.e., a set of instances evoking unusual prediction behavior in the watermarked model. The watermark is *multi-bit*, i.e., it embeds a binary signature of the model owner into the model behavior, and provides *authentication*, i.e., the legitimate model owner may claim copyright in front of a legal entity. Verification is *black-box*, i.e., the legitimate model owner may access the potentially stolen model solely through queries and has no visibility of the model parameters.

We assume that the attacker has illegitimate white-box access to the watermarked model. We also assume that the attacker does not modify the model in any way, e.g., due to some form of integrity protection or because they do not want to risk reducing model accuracy at test time. This is admittedly a strong assumption, but we consider it acceptable for a first investigation on the topic and it is in line with prior work which observed that it is extremely difficult to draw a line between adapting an existing model and creating an entirely different model on its own [8]. We leave to future work a careful analysis of the model modifications that the attacker can perform without breaking the security guarantees of watermarking. Our watermarking scheme is designed to mitigate the following threats:

(1) *Watermark detection*: the attacker should be unable to detect the presence of the watermark. This is important to limit the attacker's knowledge, making it easier to catch them red-handed when they use the model and preventing room for additional attacks against the watermark.
(2) *Watermark suppression*: the attacker should be unable to identify the queries involving the trigger set, otherwise they might change the model predictions over the trigger set to make black-box verification fail, thus rendering the watermark useless in practice.
(3) *Watermark forgery*: the attacker should be unable to construct a valid watermark, otherwise they may unduly claim ownership of the stolen model.

3.2 Watermarking Scheme

Our method is reminiscent of the watermarking scheme proposed for deep neural networks by Guo and Potkonjak [8]. Their scheme generates a binary signature of the model owner and embeds it into the training data in order to generate the trigger set. Instances from the trigger set obtain different labels than the original data points that they were based on, hence the model exhibits abnormal behavior on them. Watermark verification is performed by confirming the abnormal behavior of the model on the trigger set, in their case a significant accuracy drop with respect to a traditional model trained over the same training data.

In our case, we instead use the signature to encode a specific model behavior that the trees in the ensembles are required to show on the trigger set. We focus on random forest models without bootstrap, leaving the generalization to more sophisticated ensemble methods to future work. In these models, each tree is a classifier trained on a subset of the features of the entire training set and the final prediction is computed by aggregating individual tree predictions, e.g., using majority voting. We assume that the output of the ensemble is the sequence of the class predictions performed by each tree. For example, in R the `predict.all` field is exactly used for this purpose and a similar behavior may be easily encoded in sklearn by creating a wrapper of the `RandomForestClassifier` class. For simplicity, we present the algorithm for binary classification, i.e., the set of labels $\mathcal{Y}$ contains just a positive class +1 and a negative class -1. We discuss extension to multi-class classification in Section 3.3

Our watermarking scheme is shown in the WATERMARK function of Algorithm 1 (lines 11-23). It takes as input a training set $\mathcal{D}_{train}$, the number of trees in the ensemble m, the signature of the model owner σ (of length m) and the size of the trigger set $k \ll |\mathcal{D}_{train}|$. We denote by m' the number of bits of σ set to 0, hence $m - m'$ is the number of bits of σ set to 1. Associated with σ, we have a subset of samples of the training set, denoted by $\mathcal{D}_{trigger}$, where each tree of the ensemble must either classify correctly or misclassify based on the setting of the bits of σ.

Algorithm 1 Watermark creation algorithm

1: **function** TRAINWITHTRIGGER($\mathcal{D}_{train}, \mathcal{D}_{trigger}, m, \mathcal{H}$)
2: $\quad\mathcal{H} \leftarrow$ ADJUST($\mathcal{H}$) ▷ Adjust $\mathcal{H}$ to hide the watermark
3: $\quad W \leftarrow \{(\vec{x}, y) \mapsto 1 \mid (\vec{x}, y) \in \mathcal{D}_{train}\}$ ▷ Sample weights
4: $\quad T \leftarrow$ TRAINRANDOMFOREST($\mathcal{D}_{train}, m, \mathcal{H}, W$)
5: $\quad$**while** $\exists t_i \in T : \exists (\vec{x}, y) \in \mathcal{D}_{trigger} : t_i(\vec{x}) \neq y$ **do**
6: $\quad\quad$**for** $(\vec{x}, y) \in \mathcal{D}_{trigger}$ **do**
7: $\quad\quad\quad W[(\vec{x}, y)] \leftarrow W[(\vec{x}, y)] + 1$ ▷ Increase weights
8: $\quad\quad T \leftarrow$ TRAINRANDOMFOREST($\mathcal{D}_{train}, m, \mathcal{H}, W$)
9: $\quad$**return** T
10:
11: **function** WATERMARK($\mathcal{D}_{train}, m, \sigma, k$)
12: $\quad\mathcal{H} \leftarrow$ GRIDSEARCH($\mathcal{D}_{train}, m$) ▷ Find hyper-parameters
13: $\quad\mathcal{D}_{trigger} \leftarrow$ SAMPLE($\mathcal{D}_{train}, k$) ▷ Random sampling
14: $\quad m' \leftarrow |\{1 \leq i \leq m \mid \sigma_i = 0\}|$
15: $\quad T_0 \leftarrow$ TRAINWITHTRIGGER($\mathcal{D}_{train}, \mathcal{D}_{trigger}, m', \mathcal{H}$)
16: $\quad\mathcal{D}'_{trigger} \leftarrow \{(\vec{x}, -y) \mid (\vec{x}, y) \in \mathcal{D}_{trigger}\}$ ▷ Flip labels
17: $\quad\mathcal{D}_{train} \leftarrow (\mathcal{D}_{train} \setminus \mathcal{D}_{trigger}) \cup \mathcal{D}'_{trigger}$
18: $\quad T_1 \leftarrow$ TRAINWITHTRIGGER($\mathcal{D}_{train}, \mathcal{D}'_{trigger}, m - m', \mathcal{H}$)
19: $\quad T \leftarrow \{\}$
20: $\quad$**for** $i \in \{1, \ldots, m\}$ **do**
21: $\quad\quad$**if** $\sigma_i = 0$ **then** $T[i] \leftarrow$ GETNEXTTREE(T_0)
22: $\quad\quad$**else** $T[i] \leftarrow$ GETNEXTTREE(T_1)
23: $\quad$**return** $\langle T, \mathcal{D}_{trigger} \rangle$

Specifically, the i-th tree in the ensemble is forced to classify correctly if and only if the i-th bit of σ is set to 0. This specific output pattern is used for watermark verification and, as we argue, it is difficult to reproduce out of the trigger set, thus mitigating the risk of watermark forgery. Note that, if $m' > m/2$, then the watermarked ensemble must correctly classify all $\mathcal{D}_{trigger}$'s instances when the ensemble uses majority voting.

The algorithm first uses grid search to find the best model hyper-parameters $\mathcal{H}$ for an ensemble of m trees. After sampling a trigger set $\mathcal{D}_{trigger} \subseteq \mathcal{D}_{train}$ of size k, the algorithm trains two ensembles T_0 and T_1 with hyper-parameters $\mathcal{H}$ using the TrainWithTrigger function (lines 1-9). The function uses sample weighting to force a specific model behavior on $\mathcal{D}_{trigger}$. Specifically, all the m' trees of T_0 perform correct predictions for all samples of $\mathcal{D}_{trigger}$, while all the $m-m'$ trees of T_1 misclassify by predicting the opposite label. Before training T_0 and T_1, the function adjusts $\mathcal{H}$ to make the two ensembles look structurally more similar to each other to prevent watermark detection. In particular, we observe that trees in T_1 may have a stronger tendency to overfit than trees in T_0. The reason is that T_1 operates abnormally on the trigger set, i.e., we force prediction errors there, which often pushes trees in T_1 to grow larger than trees in T_0. To mitigate this effect, we train a standard tree ensemble with the hyper-parameters $\mathcal{H}$ and we adjust them as follows. First, we identify the average and standard deviation of the different hyper-parameters (depth and number of leaves) observed in the trained model. We then update $\mathcal{H}$ to the difference between the average and the standard deviation, i.e., we force both depth and number of leaves to be lower than the average. This simple heuristic prevents T_1's trees from growing much more than the ones in T_0, while still overfitting the expected wrong output on the trigger set. Moreover, T_0 does not deviate too much with respect to a standard ensemble trained with the goal of minimizing prediction errors over the training data. The effect is that the trees in T_0 and T_1 look similar to each other, while largely preserving model accuracy.

At the end of the algorithm, the watermarked ensemble T is constructed by picking its i-th tree from T_0 if the i-th bit of σ is 0 and from T_1 otherwise. The algorithm returns a pair $\langle T, \mathcal{D}_{trigger} \rangle$ including the watermarked ensemble and the trigger set. As for watermark verification, assume that Alice has watermarked her model using our algorithm and wants to sue Bob as an illegitimate user of the model. Alice gives to the legal authority Charlie her signature σ, the trigger set $\mathcal{D}_{trigger}$ and a set of test data $\mathcal{D}_{test}$ such that $\mathcal{D}_{trigger} \subseteq \mathcal{D}_{test}$. Charlie feeds $\mathcal{D}_{test}$ to Bob's model and retrieves the predictions associated with $\mathcal{D}_{trigger}$. Charlie then verifies that all the instances in $\mathcal{D}_{trigger}$ are classified correctly by some $t_i \in T$ iff $\sigma_i = 0$. The use of $\mathcal{D}_{test}$ is useful to prevent watermark suppression by disguising $\mathcal{D}_{trigger}$ among other instances fed in the verification phase.

3.3 Properties of the Algorithm

Our watermarking scheme is correct, because the TrainWithTrigger function enforces the correct behavior of T_0 and T_1, i.e., trees in T_0 always perform correct predictions on $\mathcal{D}_{trigger}$, while trees in T_1 always perform wrong predictions on $\mathcal{D}_{trigger}$. In particular, observe that TrainWithTrigger is called twice: once at line 15 over $\mathcal{D}_{trigger}$ and once at line 18 over $\mathcal{D}'_{trigger}$, which is the label-flipped variant of $\mathcal{D}_{trigger}$.

The complexity of the algorithm is dictated by the TrainWithTrigger function, which at its core just trains a random forest using the standard algorithm [3] multiple times. Training a decision tree has complexity $O(d \cdot n^2 \log(n))$, where d is the number of features and n is the size of the training set [4], hence the training of a single forest has complexity $O(m \cdot d \cdot n^2 \log(n))$. The maximum number of iterations allowed to enforce the desired property on $\mathcal{D}_{trigger}$ introduces an additional multiplicative factor to the complexity of the watermarking scheme.

We conclude our analysis with a few additional remarks. First, we presented our algorithm for binary classification tasks, because the existence of a single type of classification error simplifies the presentation. Our watermarking scheme can be easily extended to multi-class classification, because the signature σ just tells whether the tree t_i must classify correctly ($\sigma_i = 0$) or not ($\sigma_i = 1$). When considering more than two classes, the abnormal behavior associated to $\sigma_i = 1$ can be enforced by means of an irreflexive mapping $\phi : \mathcal{Y} \rightarrow \mathcal{Y}$ determining the type of misprediction to enforce for each possible class. Line 16 of our algorithm would be changed by substituting $-y$ with $\phi(y)$. Finally, we observe that our watermarking algorithm introduces two additional hyper-parameters (σ and k) with respect to a classic random forest algorithm. The choice of these parameters comes with a trade-off between accuracy and security. From the point of view of accuracy, picking σ with a small number of bits set to 1 and a small k is better, because this introduces limited changes with respect to a traditional random forest; while the opposite is true for security. As with any hyper-parameter, there is no optimal strategy to set σ and k a priori, but one can investigate multiple options. Our experiments in Section 4 show that picking σ with 50% of the bits set to 1 and k equal to 2% of the original training set size empirically provides good results.

3.4 Security Analysis

We here argue about the security of our watermarking scheme and we present an empirical validation of our claims in Section 4. Our scheme is robust against watermark detection because the trees of T are trained using hyper-parameters tuned by training traditional tree ensembles, i.e., the watermarked ensemble has a similar structure to a standard model. Although hyper-parameters are adjusted as explained before, the attacker does not know the optimal value of the hyper-parameters and cannot infer the adoption of watermarking from the ensemble structure alone. Most importantly, trees in T_0 and T_1 are trained using adjusted hyper-parameters forcing them to look similar in terms of depth and number of leaves, hence the correct signature σ cannot be reconstructed by inspecting the structure of the trees in the ensemble.

Moreover, our scheme is robust against watermark suppression because $\mathcal{D}_{trigger}$ is a subset of $\mathcal{D}_{train}$. This means that $\mathcal{D}_{trigger}$ is sampled from the same distribution of the training data, which are themselves assumed to be representative of the distribution of the test data (otherwise, learning would be ineffective). In other words, data in the trigger set are indistinguishable from standard test data and cannot be easily detected by the attacker during the watermark verification phase, which means that the attacker cannot maliciously adapt the model output on $\mathcal{D}_{trigger}$ to evade verification.

Finally, our scheme is robust against watermark forgery. Assume that the attacker does not know the signature σ and the trigger set $\mathcal{D}_{trigger}$, but generates a fake signature σ' and tries to forge a trigger set $\mathcal{D}'_{trigger}$ where the watermarked model exhibits the output pattern required by σ'. This is equivalent to

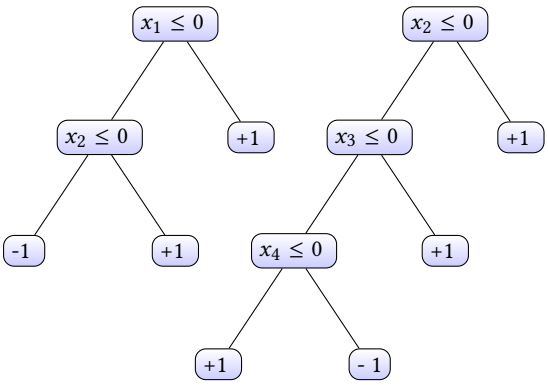


Figure 2: Conversion of the example formula $(x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$ into a tree ensemble.

solving a satisfiability problem for a logical formula encoding the expected model output. To exemplify, consider the tiny ensemble with two trees in Figure 1 and let $\sigma' = 01$ be the fake signature. Forging a positive instance $\langle \vec{x}, +1 \rangle$ matching the fake signature σ' is equivalent to finding a satisfying assignment for the following logical formula:

$$\phi \triangleq ((x_1 \leq 5 \wedge x_2 \leq 3) \vee (x_1 > 5 \wedge x_3 > 7)) \\ \wedge ((x_1 \leq 2 \wedge x_2 > 4) \vee (x_1 > 2 \wedge x_3 \leq 6)).$$

A similar reasoning may be applied to forge a negative instance $\langle \vec{x}, -1 \rangle$. In this toy example, it is easy to see that $\vec{x} = \langle x_1, x_2, x_3 \rangle = \langle 4, 3, 5 \rangle$ is a possible satisfying assignment for ϕ. However, as the size of the ensemble grows larger, such formulas become increasingly more difficult to solve and might not even admit any satisfying assignment. Note that formulas like ϕ do not define a system of linear inequalities, because they involve the disjunction operator and require solving an instance of the *Boolean satisfiability problem* (SAT), which is NP-hard in general. Indeed, we can provide a formal NP-hardness proof for the watermark forgery problem.

Definition 1. The *watermark forgery problem* is defined as follows: given a tree ensemble T, a label $y \in \{-1, +1\}$ and a signature σ, find an instance $\vec{x}$ such that $\forall t_i \in T : t_i(\vec{x}) = y \Leftrightarrow \sigma_i = 0$.

Theorem 1. *The watermark forgery problem is NP-hard.*

Proof. We show a reduction from 3SAT to watermark forgery, i.e., we show that if there exists a polynomial time algorithm to solve the watermark forgery problem, then there exists a polynomial time algorithm to solve 3SAT, which is known to be NP-complete. This proves that there is no polynomial time algorithm to solve the watermark forgery problem. First of all, we recap the 3SAT problem. A boolean variable x is a variable that can only take value true or false, while a literal l is a boolean variable or its negation. A 3CNF formula ϕ is a formula of the form $\psi_1 \wedge \ldots \wedge \psi_k$ with $k \geq 1$, where each ψ_i is a disjunction of three or less literals. More formally, 3CNF formulas ϕ are generated by the following context-free grammar: $l ::= x \mid \neg x \quad \psi ::= l \mid l \vee l \mid l \vee l \vee l \quad \phi ::= \psi \mid \phi \wedge \phi$.

An example of a 3CNF formula is $(x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$.

The 3SAT problem requires, given a 3CNF formula ϕ, to find the values of the boolean variables that make the formula true (or return a message that no such values exist). The reduction operates by first constructing an ensemble T including a decision tree t_i of depth three or less for each sub-formula ψ_i in ϕ, using prediction paths to encode the truth value of the literals therein. In particular, each internal node of the tree branches over the value of a variable x_j occurring in ψ_i with threshold 0, using the left child to represent the value false and the right child to represent the value true. We set just one of the children to have label +1, based on whether setting x_j to false or to true is a sufficient condition for the satisfiability of the sub-formula ψ_i. The conversion from 3CNF formulas to ensembles is intuitive and exemplified in Figure 2 for the example formula given above.

Generalization to arbitrary 3CNF formulas is conceptually simple, but technical to define. In particular, we define a conversion function $[\![\cdot]\!]$ by induction on the structure of the formulas as follows:

$$[\![l]\!] = \begin{cases} N(x \leq 0, L(-1), L(+1)) & \text{if } l = x \\ N(x \leq 0, L(+1), L(-1)) & \text{if } l = \neg x \end{cases}$$

$$[\![\psi]\!] = \begin{cases} [\![l]\!] & \text{if } \psi = l \\ N(x \leq 0, [\![\psi']\!], L(+1)) & \text{if } \psi = x \vee \psi' \\ N(x \leq 0, L(+1), [\![\psi']\!]) & \text{if } \psi = \neg x \vee \psi' \end{cases}$$

$$[\![\phi]\!] = \begin{cases} [\![\psi]\!] & \text{if } \phi = \psi \\ \langle [\![\phi_1]\!], [\![\phi_2]\!] \rangle & \text{if } \phi = \phi_1 \wedge \phi_2. \end{cases}$$

By construction, we have that ϕ is satisfiable if and only if the watermark forgery problem has a solution for the ensemble $[\![\phi]\!]$ using label $y = +1$ and signature $\sigma = \langle 0, \ldots 0 \rangle$. Indeed, the leaves of a tree t_i with label +1 identify prediction paths encoding sufficient conditions for the satisfiability of the sub-formula ψ_i, hence finding a positive instance $\vec{x}$ such that $t_i(\vec{x}) = +1$ is equivalent to finding a satisfying assignment for ψ_i. The bits of σ are all set to 0 because ϕ is satisfiable if and only if all the sub-formulas ψ_i are satisfiable, being ϕ a conjunction. If a solution $\vec{x}$ is found for the watermark forgery problem, we can translate into a value assignment for 3SAT by having each variable x_j set to true if and only if the j-th component of the solution is positive. □

4 EXPERIMENTAL EVALUATION

We implemented the proposed watermarking scheme on top of the `sklearn` library and we make our code publicly available to support reproducibility[1]. We here evaluate the accuracy of watermarked models and the security of our watermarking scheme on public datasets (MNIST2-6[2], breast-cancer[3] and ijcnn1[4]) normalized in the interval $[0, 1]$. Table 1 reports the most relevant dataset statistics, showing that the considered datasets are diverse in terms of number of instances, number of features and class distribution. Note that MNIST2-6 includes digits representing numbers 2 and 6 from the MNIST dataset, while ijcnn1 has been reduced to 20,000 instances using stratified random sampling to speed up the experimental evaluation. MNIST is a standard benchmark that has been used for evaluating watermarking techniques in many papers [5, 12, 13]. Breast-cancer and ijccn1 are instead two numerical tabular datasets that are representative of typical tasks where tree ensembles are fruitfully adopted.

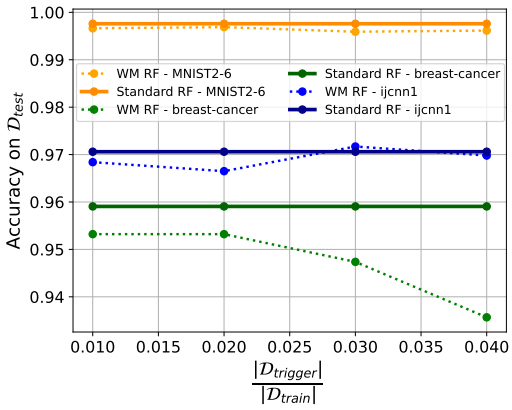
4.1 Accuracy Evaluation

Since watermarked models force a specific prediction pattern over the trigger set, their predictive power on the test data might

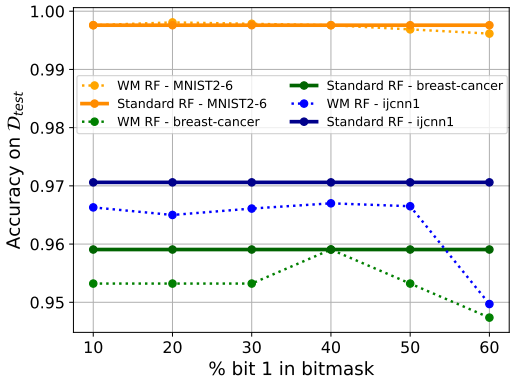
[1] https://github.com/LorenzoCazzaro/watermarking-decision-tree-ensembles
[2] https://www.openml.org/search?type=data&sort=runs&id=554&status=active
[3] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#breast-cancer
[4] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#ijcnn1

Table 1: Dataset statistics.

Dataset	Instances	Features	Distribution
MNIST2-6	13,866	784	51%/49%
breast-cancer	569	30	37%/63%
ijcnn1	20,000	22	90%/10%



(a)



(b)

Figure 3: Accuracy of watermarked models on the test set when varying the percentage of training instances included in $\mathcal{D}_{trigger}$ (top figure) and the percentage of bits set to 1 in the signature σ (bottom figure).

be penalized. In our first set of experiments, we evaluate the accuracy loss introduced by our watermarking scheme. Figure 3a plots how accuracy downgrades for increasing sizes of the trigger set, given a fixed randomly generated signature including 50% of the bits set to 1. The figure shows that the accuracy loss is limited in general and even negligible when the size of the trigger set does not exceed 2%.

Of course, the number of bits set to 1 in the signature might also impact the accuracy of the watermarked model, because such bits denote forced prediction errors. Figure 3b shows how accuracy changes when we increase the number of bits set to 1 in the signature, given a fixed trigger set (including 2% of the training data). Again, the accuracy loss is small in practice, with the largest drop in accuracy amounting to around two points.

4.2 Security Evaluation

We focus on watermark detection and watermark forgery, because protection against watermark suppression is immediately achieved by construction. We assume that σ includes 50% of the bits set to 1 and $\mathcal{D}_{trigger}$ includes 2% of the training set.

4.2.1 Watermark Detection. We compare the depth and the number of leaves of the trees corresponding to bits set to 0 and to 1 in the signature σ to understand whether there are relevant differences leaking information about σ. This is a significant threat, because trees associated with a bit set to 1 are forced to make prediction errors in the trigger set, hence they might grow larger than the other trees when trying to achieve overfitting. We simulate two watermark detection strategies by means of the following experiment: given a hyper-parameter like depth or number of leaves, the attacker computes its mean and standard deviation over the ensemble. Intuitively, "small" trees are more likely to be associated with bit 0 and "large" trees are more likely to be associated with bit 1. To formalize this intuition, in our first strategy the attacker associates bit 0 with all trees falling below the difference of the mean and standard deviation, and bit 1 to all trees falling above the sum of mean and standard deviation; all the other trees around the mean correspond to *uncertain* cases, where the attacker might try random guessing. Note that this strategy may produce a large number of uncertain cases, thus making random guessing of them infeasible for the attacker. However, the technique is interesting because we can check whether it can correctly identify at least the rest of the trees. The second strategy does not produce uncertain trees, as it uses the mean as a sharp threshold to determine whether a tree is associated with bit 0 or 1. Table 2 reports the results, showing that both the attack strategies are ineffective. The first strategy (in red) yields a huge number of uncertain cases, but surprisingly it also produces wrong predictions for the rest of the trees. The second strategy (in blue) has no uncertainty, but produces many prediction errors and is unable to reconstruct the signature. Finally, we can observe that standard deviation values are relatively small compared to the values of the associated means. Therefore, the trees trained by our techniques are all similar to each other, thus making it very difficult for an attacker to identify σ.

4.2.2 Watermark Forgery. To show security against watermark forgery, we simulate a scenario where the attacker generates a fake signature σ' and tries to forge a trigger set $\mathcal{D}'_{trigger}$ where the watermarked model exhibits the output required by σ'. We showed that this requires solving an NP-hard problem, however recent advances in automated verification enable dealing with large inputs even for computationally intensive problems, hence we complement our theoretical analysis with empirical evidence. We implement our forgery attempts by generating 10 random signatures and solving a satisfiability problem for a logical formula encoding the expected model output using Z3, a state-of-the-art SMT solver [6]. For each fake signature, we iterate over all the instances in the test set and we look for a satisfying assignment for our logical formula, while requiring that the L_∞-distance between the solution and the original test instance is bounded by some $0 < \varepsilon < 1$. The distance constraint is useful to ensure that the forged trigger set $\mathcal{D}'_{trigger}$ is reminiscent of real test instances. We are in fact assuming that, as usual, the test set has the same distribution of the training set.

Our experiment shows different results on the different datasets. In the case of breast-cancer, the forged trigger set reaches at most

Table 2: Number of trees correctly/wrongly associated with their bits, using two watermark detection strategies. For each dataset, *mean* and *standard deviation* of "Depth" and "#leaves" are reported in round brackets.

Dataset	Hyper-Parameters	#correct	#wrong	#uncertain
MNIST2-6	Depth (19.82 - 2.69)	31 / 57	11 / 33	48 / 0
	#leaves (229.99 - 0.10)	1 / 46	0 / 44	89 / 0
breast-cancer	Depth (7.03 - 0.81)	34 / 46	9 / 24	27 / 0
	#leaves (18.90 - 0.45)	4 / 39	0 / 31	66 / 0
ijcnn1	Depth (18.00 - 0.00)	0 / 40	0 / 40	80 / 0
	#leaves (498.88 - 5.86)	0 / 37	3 / 43	77 / 0

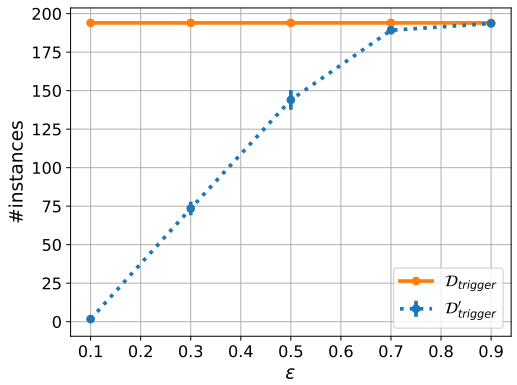


Figure 4: Size of the forged trigger set $\mathcal{D}'_{trigger}$ when varying the amount of distortion ε on the MNIST2-6 dataset.

Figure 5: Instances generated by Z3 for $\varepsilon \in \{0.3, 0.5, 0.7\}$.

14% of the size of the original trigger set, even when setting a high $\varepsilon = 0.9$. This is explained by the fact that Z3 does not find satisfying assignments for most of the logical formulas, hence the legitimate model owner is the only one who is able to present a trigger set of significant size. In the case of ijcnn1, instead, the forged trigger set is just 1% of the size of the original trigger set on average for $\varepsilon = 0.1$. Forging a trigger set of the same size as the original trigger set for $\varepsilon > 0.1$ does not scale, already requiring more than four hours for a single bitmask for $\varepsilon = 0.3$. The reason is that the ensemble for ijcnn1 contains more than twice the leaves of the ensembles for the other two datasets, making the satisfiability problem more difficult. The results are more interesting for the MNIST2-6 dataset and we visualize them in Figure 4. The figure shows that, when ε increases, it becomes easier to forge trigger sets of comparable size to the original trigger set. However, the amount of distortion required by the forgery makes it easy to detect such malicious attempts, because the size of the forged trigger set become comparable to the original only when $\varepsilon \geq 0.7$. Figure 5 shows three forged images of $\mathcal{D}'_{trigger}$ for increasing values of $\varepsilon \in \{0.3, 0.5, 0.7\}$. As we can see, the image with the highest amount of distortion is rather blurry and quite far from the original image. Indeed, a standard decision tree ensemble achieves 0.99 accuracy on the original trigger set, while its accuracy drops to 0.62 on the forged trigger set.

4.3 Performance Evaluation

In our experiments, we enforced a maximum of 500 attempts to achieve the desired behavior on the trigger set during training. Given the efficiency of random forest training, our watermarking scheme remains relatively efficient despite the need for multiple training attempts through sample weighting. On the smallest dataset, breast-cancer, training a forest of 70 trees (with a maximum depth of 8) takes less than one second, both with and without watermark embedding. However, as the dataset size increases, so do the trigger set size and the computational cost of the watermarking algorithm. For instance, on the MNIST2-6 dataset, training a forest of 90 trees with a maximum depth of 24 takes 20 seconds. Similarly, training a forest of 80 trees with a maximum depth of 18 on the ijcnn1 dataset takes 27 seconds. Although incorporating a watermark adds an overhead, compared to the two seconds required to train a traditional forest without watermarking on MNIST2-6 and ijcnn1, our training algorithm demonstrates acceptable performance for practical use, as watermarked models of reasonable size can be trained in a matter of seconds.

5 CONCLUSION

We proposed the first watermarking scheme designed for decision tree ensembles and we proved the security of our construction. Our experimental evaluation shows convincing results, because watermarked models largely preserve their accuracy and are robust against relevant attacks. As future work, we plan to extend our security analysis to more powerful attackers, e.g., who are able to modify the watermarked model and forge trigger sets using more sophisticated strategies. We would also like to generalize our watermarking scheme to more advanced decision tree ensembles, such as those trained using gradient boosting.

ACKNOWLEDGMENTS

We thank the reviewers for their constructive feedback, which has greatly contributed to the improvement of this paper. This research was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Moreover, it acknowledges support from the European Union - Next-GenerationEU - PNRR – M.4 C.2, I.1.1 - PRIN 2022 WHAM!, 2022ZZX57L, H53D23003750006.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cissé, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1615–1631. https://www.usenix.org/conference/usenixsecurity18/presentation/adi

[2] Franziska Boenisch. 2021. A Systematic Review on Model Watermarking for Neural Networks. *Frontiers Big Data* 4 (2021), 729663. https://doi.org/10.3389/FDATA.2021.729663

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

[4] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.

[5] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. 2019. BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks. *CoRR* abs/1904.00344 (2019). arXiv:1904.00344 http://arxiv.org/abs/1904.00344

[6] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science)*, C. R. Ramakrishnan and Jakob Rehof (Eds.), Vol. 4963. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[7] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. 2019. Rethinking Deep Neural Network Ownership Verification: Embedding Passports to Defeat Ambiguity Attacks. 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*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 4716–4725. https://proceedings.neurips.cc/paper/2019/hash/75455e062929d32a333868084286bb68-Abstract.html

[8] Jia Guo and Miodrag Potkonjak. 2018. Watermarking deep neural networks for embedded systems. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2018, San Diego, CA, USA, November 05-08, 2018*, Iris Bahar (Ed.). ACM, 133. https://doi.org/10.1145/3240765.3240862

[9] Minoru Kuribayashi, Takuro Tanaka, and Nobuo Funabiki. 2020. DeepWatermark: Embedding Watermark into DNN Model. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA 2020, Auckland, New Zealand, December 7-10, 2020.* IEEE, 1340–1346. https://ieeexplore.ieee.org/document/9306331

[10] Chuan Li. 2020. Demystifying GPT-3. https://lambdalabs.com/blog/demystifying-gpt-3 [Accessed: (23 May 2024)].

[11] Yue Li, Hongxia Wang, and Mauro Barni. 2021. A survey of Deep Neural Network watermarking techniques. *Neurocomputing* 461 (2021), 171–193. https://doi.org/10.1016/J.NEUCOM.2021.07.051

[12] Ryota Namba and Jun Sakuma. 2019. Robust Watermarking of Neural Network with Exponential Weighting. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang (Eds.). ACM, 228–240. https://doi.org/10.1145/3321705.3329808

[13] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 485–497. https://doi.org/10.1145/3297858.3304051

[14] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. 2021. DAWN: Dynamic Adversarial Watermarking of Neural Networks. In *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, Heng Tao Shen, Yueting Zhuang, John R. Smith, Yang Yang, Pablo César, Florian Metze, and Balakrishnan Prabhakaran (Eds.). ACM, 4417–4425. https://doi.org/10.1145/3474085.3475591

[15] Ruixiang Tang, Mengnan Du, and Xia Hu. 2023. Deep Serial Number: Computational Watermark for DNN Intellectual Property Protection. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part VI (Lecture Notes in Computer Science)*, Gianmarco De Francisci Morales, Claudia Perlich, Natali Ruchansky, Nicolas Kourtellis, Elena Baralis, and Francesco Bonchi (Eds.), Vol. 14174. Springer, 157–173. https://doi.org/10.1007/978-3-031-43427-3_10

[16] Ruixiang Tang, Hongye Jin, Mengnan Du, Curtis Wigington, Rajiv Jain, and Xia Hu. 2023. Exposing Model Theft: A Robust and Transferable Watermark for Thwarting Model Extraction Attacks. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, Ingo Frommholz, Frank Hopfgartner, Mark Lee, Michael Oakes, Mounia Lalmas, Min Zhang, and Rodrygo L. T. Santos (Eds.). ACM, 4315–4319. https://doi.org/10.1145/3583780.3615211

[17] Enzo Tartaglione, Marco Grangetto, Davide Cavagnino, and Marco Botta. 2020. Delving in the loss landscape to embed robust watermarks into neural networks. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021.* IEEE, 1243–1250. https://doi.org/10.1109/ICPR48806.2021.9413062

[18] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, ICMR 2017, Bucharest, Romania, June 6-9, 2017*, Bogdan Ionescu, Nicu Sebe, Jiashi Feng, Martha A. Larson, Rainer Lienhart, and Cees Snoek (Eds.). ACM, 269–277. https://doi.org/10.1145/3078971.3078974

[19] Jie Zhang, Dongdong Chen, Jing Liao, Han Fang, Weiming Zhang, Wenbo Zhou, Hao Cui, and Nenghai Yu. 2020. Model Watermarking for Image Processing Networks. 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, 12805–12812. https://doi.org/10.1609/AAAI.V34I07.6976