Trained Random Forests Completely Reveal your Dataset

Julien Ferry [1] **Ricardo Fukasawa** [2] **Timothée Pascal** [3] **Thibaut Vidal** [1]

Abstract

We introduce an optimization-based reconstruction attack capable of completely or near-completely reconstructing a dataset utilized for training a random forest. Notably, our approach relies solely on information readily available in commonly used libraries such as `scikit-learn`. To achieve this, we formulate the reconstruction problem as a combinatorial problem under a maximum likelihood objective. We demonstrate that this problem is $\mathcal{NP}$-hard, though solvable at scale using constraint programming – an approach rooted in constraint propagation and solution-domain reduction. Through an extensive computational investigation, we demonstrate that random forests trained without bootstrap aggregation but with feature randomization are susceptible to a complete reconstruction. This holds true even with a small number of trees. Even with bootstrap aggregation, the majority of the data can also be reconstructed. These findings underscore a critical vulnerability inherent in widely adopted ensemble methods, warranting attention and mitigation. Although the potential for such reconstruction attacks has been discussed in privacy research, our study provides clear empirical evidence of their practicability.

1. Introduction

Machine learning (ML) techniques are increasingly used on sensitive data, such as medical records for kidney exchange (Aziz et al., 2021), criminal records (Angwin et al., 2016) or credit history. As this raises significant ethical and societal challenges, the use of such private data is directly regulated by several legal texts, such as the recent European Union General Data Protection Regulation* or the forthcoming AI Act**. Privacy has attracted significant attention during the last decades (Liu et al., 2021a) in order to protect sensitive or personal information about individual users while still being able to extract useful patterns from data. Moreover, privacy risks may further be exacerbated by the consideration of other ethical desiderata, e.g., when releasing a trained ML model for the sake of transparency.

In this work, we specifically study such privacy concerns in the white-box setting in which a trained random forest (RF) is publicly released. More precisely, we attempt to reconstruct the entire dataset used to train the RF by only using information available by default in widespread libraries such as `scikit-learn` (Pedregosa et al., 2011), namely the structure of the trees within the forest and the class cardinalities provided within each node.

While reconstruction attacks have been previously studied (Dwork et al., 2017), to the best of our knowledge, no work could consistently reconstruct an entire dataset from a trained RF. While some information can be extracted from single trees regarding the number of examples with specific combinations of features, the path taken by each individual example in each tree is unknown. Consequently, it is challenging to combine the information provided by different trees to effectively narrow down the potential datasets. To achieve this goal, we formalize the *maximum-likelihood dataset reconstruction problem* and formulate it as a unified Constraint Programming (CP) model over the forest. With this, we can leverage the solution capabilities of modern CP algorithms based on constraint propagation, solution domain reduction, exploration, and backtracking. In an extensive computational campaign, we show that our methodology achieves nearly flawless recovery for RFs trained without bootstrap aggregation but with feature randomization. Even in cases where bootstrap aggregation is employed, our approach successfully recovers the majority of the data. In summary, the main contributions of this study are:

- A formalization of the *maximum-likelihood dataset reconstruction problem* for random forests
- A proof of $\mathcal{NP}$-hardness for this problem. This is, how-

[1]CIRRELT & SCALE-AI Chair in Data-Driven Supply Chains, Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Canada [2]Department of Combinatorics and Optimization, University of Waterloo, Canada [3]Ecole nationale des ponts et chaussées, Paris, France. Correspondence to: Thibaut Vidal <thibaut.vidal@polymtl.ca>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

*https://gdpr-info.eu/

**https://artificialintelligenceact.eu/

ever, a limited safeguard since the relentless progress of generalist combinatorial optimization algorithms (i.e., based on CP or mixed-integer programming) permits solving many $\mathcal{NP}$-hard problems at scale nowadays.
- The proposal of a CP formulation amenable to an efficient solution using state-of-the-art algorithms.
- Extensive computational experiments demonstrating how even a reasonably small number of trees reveal the quasi-totality of the datasets on standard applications. Our source code is openly accessible at `https://github.com/vidalt/DRAFT` in the form of a user-friendly Python module named DRAFT *(Dataset Reconstruction Attack From Trained ensembles)*, under a MIT license.

2. Technical Background

Supervised Machine Learning (ML). Let $\{\mathbf{x}_k; c_k\}_{k=1}^N$ be a training set in which each example k is characterized by a vector $\mathbf{x}_k \in \{0,1\}^M$ of M binary attributes and a class $c_k \in \mathcal{C}$. We let z_{kc} be a one-hot encoding of the classes, which is 1 if $c_k = c$, and 0 otherwise. Moreover, in some situations, several binary features are used to one-hot encode a single original numerical or categorical attribute. In such case, precisely one of these binary features is 1, and the others are 0. We let $vects$ be the list of the different groups (if any) of binary attributes one-hot encoding the same original feature.

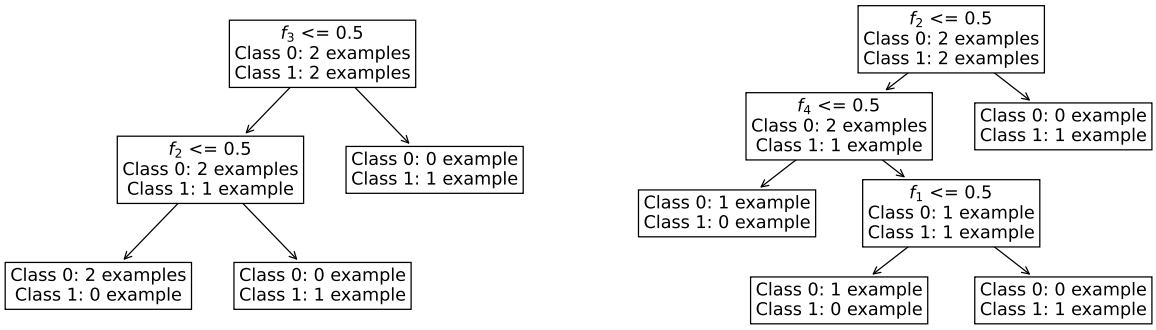
Random Forests (RFs). The training dataset is used to build a random forest $\mathcal{T}$ in which each tree $t \in \mathcal{T}$ is made of a set of internal nodes $\mathcal{V}_t^I$ and a set of leaves $\mathcal{V}_t^L$. Each internal node $v \in \mathcal{V}_t^I$ corresponds to a binary condition over the value of a given attribute. If the condition is satisfied, the example being classified descends towards the left child $l(v)$ of the node, otherwise it descends towards its right child $r(v)$. Once the example reaches a leaf $v \in \mathcal{V}_t^L$ (terminal node), it is classified according to the class associated with this leaf. Such class corresponds to the majority class among the training examples captured by the leaf. To compute it (and eventually assign class probabilities), each leaf contains the per-class number of training examples it captures. In popular ML libraries such as `scikit-learn`, such counts are also provided in the internal nodes, as shown in Figure 1. Then, for every node $v \in \mathcal{V}_t^I \bigcup \mathcal{V}_t^L$, let n_{tvc} denote the number of training examples of class c that went through v.

Training RFs. To encourage diversity between the different trees within an RF, several randomization mechanisms are used during training. For instance, when building each individual tree, only a random subset of the M features is considered to determine the best split at each node. Note that this mechanism is used in all our experiments, although we do not explicitly leverage it. Bootstrap aggregation (*bagging*) is another popular and successful mechanism in RF training (Zhou, 2012). It consists in building $|\mathcal{T}|$ separate training sets, one for each tree, by performing random sampling with replacement from the original training set $\{\mathbf{x}_k; c_k\}_{k=1}^N$. In consequence, not all examples of the original dataset are used for training each tree, while some appear multiple times. Algorithmic implementations for learning RFs are available within popular libraries such as `scikit-learn`. While bagging is not mandatory, it is often used by default, as it lowers variance and enhances generalization. Finally, some support or size constraints are often set when training each tree. In particular, it is possible to set a maximum depth constraint ensuring that each tree has depth at most d_{max}.
In our framework, we leverage both the structure of the trees within the forest and the counts provided within each node to conduct a dataset reconstruction attack. We additionally take advantage of the theoretical probability distributions of the number of occurrences of each example within each tree's training set.

Constraint Programming (CP). CP is a generic approach to finding feasible or optimal solutions to a wide variety of problems, including $\mathcal{NP}$-hard ones. The basic principle is to define a set of *decision variables* – each allowed to take values within a given (discrete) domain – and *constraints* that express relationships between variables. Optionally, an *objective function* may be provided to be maximized or minimized. The types of allowed constraints depend on which specific CP solver is used, but linear and logical/implication constraints are typical examples.
CP solvers then combine several techniques (constraint propagation, backtracking, local search) to efficiently explore the search space and find a feasible/optimal solution. An overview of fundamental ideas/techniques in CP can be found in Rossi et al. (2008). While solving CP models is theoretically $\mathcal{NP}$-hard, state-of-the-art solvers can handle very large-scale problems in practice, and the performance of state-of-the-art solvers has dramatically increased.

3. Related Works

ML methods often exploit private data during training. Consequently, it is crucial to ensure that their outputs – which may be released directly or accessed through a dedicated API – do not leak information regarding their inputs (Dinur & Nissim, 2003). *Inference attacks* against ML models (Rigaki & Garcia, 2020) precisely aim at exploiting the output of a learning algorithm to infer information regarding the training dataset. Different attacks can be distinguished, depending on their specific objective. For instance, *membership inference attacks* aim to infer whether an example was part of a model's training data or not (Shokri et al., 2017; Carlini et al., 2022). In this work, we are interested



(a) Tree t_1 (trained without using attribute f_1)

(b) Tree t_2 (trained without using attribute f_3)

Figure 1: Example decision trees trained using `scikit-learn` on a small dataset (Table 1).

in *dataset reconstruction attacks*, which aim at reconstructing (entirely or partially) a model's training dataset (Dwork et al., 2017). The considered *attack model* is as follows. We specifically target RF models, and consider the *white-box setup*, in which the adversary has complete knowledge of the model's parameters instead of a black-box API access to it (Cristofaro, 2020). He also knows the domains of the different attributes involved in the data. Importantly, he does not intervene during the training process, but rather gets the trained model afterwards.

RECONSTRUCTION ATTACK. *Given a trained RF, find a reconstructed version of its training set – a value for each feature of each example – that is feasible and likely w.r.t. the training process. Ideally, the reconstructed dataset should closely match the actual training data* $\{\mathbf{x}_k; c_k\}_{k=1}^{N}$.

Reconstruction attacks are one of the most ambitious inference attacks against ML models, as they directly aim to recover entire parts of the training data. However, instead of attempting to reconstruct the whole training set, most reconstruction attacks only target retrieving part of it. For instance, the first reconstruction attacks (originally proposed against database access mechanisms) only aimed at retrieving one private binary attribute for all the database examples - assuming all other attributes were publicly known (Dinur & Nissim, 2003; Dwork et al., 2007; 2017). Some other studies only target reconstructing part of one particular example, given some public information about it (Fredrikson et al., 2014; 2015). Other approaches require additional knowledge, such as intermediate gradients computed during collaborative (Phong et al., 2017) or online (Salem et al., 2020) learning, stationary points reached by gradient descent algorithms (Haim et al., 2022) or information regarding the model's fairness (Hu & Lan, 2020; Aalmoes et al., 2022; Hamman et al., 2022; Ferry et al., 2023).

The most closely related works are those of Gambs et al. (2012) and Ferry et al. (2024). More precisely, Gambs et al. (2012) showed that the structure of a single trained decision tree can be leveraged to build a probabilistic dataset encoding the whole set of reconstructions of the training data that are compatible with the provided tree's structure. This approach was later generalized in Ferry et al. (2024) to consider other simple interpretable models. Compared to previous studies, one of the key challenges addressed by our approach is to combine the information provided by several trees to achieve a feasible and accurate reconstruction. This is especially difficult since the number of occurrences (due to *bagging*) and the path taken by each individual example in each tree is unknown. Consequently, we specifically design our method to handle the random selection of examples within each tree and formulate a maximum log-likelihood objective to guide the search.

Table 1: Example binary dataset with $N = 4$ and $M = 4$.

f_1	f_2	f_3	f_4	c
0	0	0	1	0
1	0	0	0	0
0	1	0	0	1
1	0	1	1	1

4. Illustrative Example

We first give an intuition of the reconstruction problem on a small dataset (Table 1) with 4 examples described by 4 binary attributes $f_{i\in\{1..4\}}$ and a binary class c. Figure 1 provides two decision trees trained on this dataset. Tree t_1 was trained without using f_1, while tree t_2 was trained without using f_3 (though this information is unknown to the reconstruction algorithm). For presentation simplicity, bagging is not used here, and therefore each training example is used a single time in each tree. By following the paths from the root to each leaf within t_1 (Figure 1a), one can set the value of some attributes within the reconstructed

dataset by leveraging the performed splits and the per-node cardinalities. For instance, following the leftmost path, we can observe that the two examples belonging to class 0 have value 0 for both f_3 and f_2. Such information permits to fix some attributes' values directly. Similarly, according to t_2, there exists exactly one example of class 1 with value 0 for f_2, and another one with value 1.

The main issue with such ad-hoc reasoning is that, except in some obvious cases (i.e., when *all* examples of a certain class respect a given splitting condition), splits will permit quantifying *how many* examples respect a certain condition without telling *which* are these examples. Therefore, the biggest challenge of dataset reconstruction is to individually link the examples between the different trees and find a compatible dataset that respects all the cardinality constraints. This challenge is exacerbated by the bagging process, as in this case, the cardinalities within the trees' nodes may count some examples several times (and ignore some others).

5. NP-Hardness Result

In this section, we formally define the *dataset reconstruction problem (DRP)* and show its $\mathcal{NP}$-completeness. Part of the input data for (DRP) has already been defined before: the set of classes $\mathcal{C}$, the number N of examples, the number M of binary attributes, and the forest $\mathcal{T}$, where each $t \in \mathcal{T}$ is a binary tree. Also, for each class $c \in \mathcal{C}$, tree $t \in \mathcal{T}$ and each $v \in \mathcal{V}_t^I \cup \mathcal{V}_t^L$, we are given an amount $n_{tvc} \in \mathbb{Z}_+$ of examples of class c that are classified in node v of tree t.

In addition, we are given as input:

- for every $v \in \mathcal{V}_t^I$, an attribute $f_v \in \{1..M\}$.
- a set $\mathcal{B}$ of integer values, that represent how many times a sample may appear in a tree

We assume that the data satisfies the following properties:

- For each class $c \in \mathcal{C}$, tree $t \in \mathcal{T}$ and each $v \in \mathcal{V}_t^I$, we have that $n_{tvc} = n_{tl(v)c} + n_{tr(v)c}$.
- For each tree $t \in \mathcal{T}$, we have $\sum_{v \in \mathcal{V}_t^L} \sum_{c \in \mathcal{C}} n_{tvc} = N$.

For each tree $t \in \mathcal{T}$, we let $\Phi_v^+ \subseteq \{1..M\}$ denote the set of indices of the boolean attributes that must be TRUE for an example to fall into node $v \in \mathcal{V}_t^L \cup \mathcal{V}_t^I$. Similarly, $\Phi_v^- \subseteq \{1..M\}$ is the set of indices of the boolean attributes that must be FALSE for an example to fall into v (hence $\Phi_v^+ \cap \Phi_v^- = \emptyset$). Both represent the splits that are found along the path from the root node of tree $t \in \mathcal{T}$ to v. Formally, if v is the root node of the tree, then $\Phi_v^+ = \Phi_v^- = \emptyset$. For every $v \in \mathcal{V}_t^I$, we can define such sets for its children as $\Phi_{l(v)}^+ = \Phi_v^+$ and $\Phi_{l(v)}^- = \Phi_v^- \cup \{f_v\}$; $\Phi_{r(v)}^+ = \Phi_v^+ \cup \{f_v\}$ and $\Phi_{r(v)}^- = \Phi_v^-$

The goal of (DRP) is to find N vectors $x_1, .., x_N \in \{0,1\}^M$, respective classifications $z_1, .., z_N \in \mathcal{C}$ and node incidences $y_{tvk} \in \mathbb{Z}_+, \forall t \in \mathcal{T}, v \in \mathcal{V}_t^L, k \in \{1..N\}$ such that:

- $\sum_{v \in \mathcal{V}_t^L} y_{tvk} \in \mathcal{B}, \forall t \in \mathcal{T}, k \in \{1..N\}$
- $\sum_{k \in \{1,\dots,N\}: z_k = c} y_{tvk} = n_{tvc}, \forall t \in \mathcal{T}, v \in \mathcal{V}_t^L$
- For all $k \in \{1..N\}$, $t \in \mathcal{T}$, $v \in \mathcal{V}_t^L$, if $y_{tvk} > 0$, then $(x_k)_i = 0$ for all $i \in \Phi_v^-$ and $(x_k)_i = 1$ for all $i \in \Phi_v^+$.

We note that the last constraint implies that for every $t \in \mathcal{T}$ and $k \in \{1..N\}$, at most one variable in the set $\{y_{tvk} : v \in \mathcal{V}_t^L\}$ is nonzero. This is due to the fact that, from the way Φ_v^+ and Φ_v^- are constructed, all other leaves $v' \in \mathcal{V}_t^L \setminus \{v\}$ must have at least one attribute $i \in \Phi_v^+ \cup \Phi_v^-$ switching its required TRUE/FALSE value in v'.

Note that if we set $\mathcal{B} = \{1\}$, we impose that each example must appear exactly once in every tree, which corresponds to the situation when no bagging is used. If we set $\mathcal{B} = \{0, \dots, N\}$, we get that each example can appear any number of times in a tree, which corresponds to the situation when bagging is used.

Theorem 5.1. *The decision version of (DRP) is $\mathcal{NP}$-complete.*

Proof. First, a **YES** certificate of (DRP) is any solution (x, z, y). One can verify if it is feasible in polynomial time, so the decision version of $(DRP) \in \mathcal{NP}$.

Next, consider an instance of the $\mathcal{NP}$-complete problem 3-SAT, given by a set $L = \{1, \dots, |L|\}$ of clauses with three literals each. Each literal is either a variable or its complement, and there are V possible variables. We construct an instance of (DRP) with $|\mathcal{C}| = 1$, by specifying the forest $\mathcal{T}$ and n_{lvc} values. We let $\mathcal{B} = \{1\}$. By assumption on the data, we only need to specify n_{lvc} for $v \in \mathcal{V}_t^L$. Also, recall that the left branch of $v \in \mathcal{V}_t^I$ corresponds to setting f_v to zero, and the right branch sets it to 1. The idea is to construct a perfect binary tree for each $l \in L$. This way, the fixed attributes Φ_v^+ and Φ_v^- of each leaf v of tree l will represent one of each possible assignment of the three literals of l to 0 or 1.

Out of the eight possible leaves of tree l, seven satisfy the corresponding clause, and one does not. We set $n_{lvc} = 1$ for the leaves that satisfy clause l and $n_{lvc} = 0$ otherwise. The goal of our construction is to force the DRP solver to generate an example whose attribute values lead it towards one of the leaves with $n_{lvc} = 1$ in each tree, i.e., a 3-SAT solution, whenever such a solution exists. To achieve this, we include six additional "dummy" examples for each tree to reach the remaining alternatives with $n_{lvc} = 1$. Therefore, our constructed instance has $N = 6|L| + 1$ examples.

We rely on $M = V + |L|$ binary attributes. The first V attributes will represent the 0/1 assignment of values to literals. The remaining $|L|$ attributes will determine whether one example is used or not in the leaves corresponding to the perfect binary tree of that clause. With that, we add to each tree $l = 1, \ldots, |L|$, a root node where we branch on feature $V + l$. The right branch of the tree will contain the perfect binary tree described above. The left branch is a single leaf node with $n_{lvc} = 6|L| - 6$, designed to absorb all dummy examples not destined toward that tree.

We finally construct one extra auxiliary tree. All its left branches end up in leaves. Each node on the right branch at depth d (including the root node) branches on attribute $V + d + 1$. The left leaf at depth 1 has $n_{lvc} = 6|L| - 6$. The left leaf at depth 2 has $n_{lvc} = 6$. All other left leaves have $n_{lvc} = 0$. The rightmost leaf has $n_{lvc} = 1$. With this, the rightmost leaf imposes that a single example reaches one leaf of every perfect tree (the desired 3-SAT solution). The other nodes of the tree just count the extra examples.

Appendix A proves that (DRP) is feasible if and only if the 3-SAT instance is a **YES** instance. It also contains illustrative examples for the construction. □

The optimization version of the problem is to search for the solution that has the largest likelihood, called the *maximum likelihood dataset reconstruction problem* (MLDRP). This problem is $\mathcal{NP}$-hard since even reconstructing one feasible solution is $\mathcal{NP}$-complete. The maximum likelihood objective function will be formally introduced in the next section.

6. Constraint Programming Approach

As seen in Section 4, an inspection of the different trees gives sets of restrictions over feature values that concern a known number of examples of each class. However, it does not tell which example specifically satisfies which condition. Testing feasible combinations by inspection would require extensive trial and error, leading to an intractable process. Instead, we propose to formulate this search problem as a constraint programming (CP) model, permitting the use of efficient out-of-the-shelf solvers for such models. The model we design covers the most general case where bagging is used to train the forest, and includes discretization strategies specifically designed to help the solution process. Note that, while we focus on CP, Mixed-Integer Linear Programming (MILP) could also be employed instead. However, having conducted experiments with both techniques, and as demonstrated in Appendix B, CP generally achieved better performance and permitted to handle bagging much more effectively.

For our mathematical formulation, we define three sets of decision variables. The first one assigns training examples to a corresponding class. The second assigns the training examples to the trees' leaves, and the third connects the attributes' values to the splits leading to their assigned leaf.

- $\forall k \in \{1..N\}, \forall c \in \mathcal{C}$: z_{kc} is 1 if training example k is considered as part of class c, else 0.
- $\forall t \in \mathcal{T}, \forall v \in \mathcal{V}_t^L, \forall k \in \{1..N\}, \forall c \in \mathcal{C}$: $y_{tvkc} \in \mathbb{Z}_+$ is the number of times training example k is classified by leaf v within tree t as class c.
- $\forall k \in \{1..N\}, \forall i \in \{1..M\}$: $x_{ki} \in \{0; 1\}$ is the value of feature i for example k in the reconstruction.

To define the objective function, we will assume that a training example appears at most 7 times in any tree, since higher values are very unlikely. Indeed, bootstrap sampling consists of sampling with replacement N examples from a set of N original examples. At each iteration of the bootstrap sampling process, each example has a probability of $\frac{1}{N}$ of being selected. The probability of an example being selected more than B times can then be computed as:

$$1 - \sum_{b=0}^{B-1} \left(\left(\frac{1}{N}\right)^b \cdot \left(\frac{N-1}{N}\right)^{N-b} \cdot \binom{N}{b} \right).$$

If $N = 100$ as in our experiments, the probability of an example appearing more than 7 times in a bootstrap sampled training set is roughly 10^{-5}. This value remains similar for larger values of N (e.g., around 10^{-5} for $N = 10^{10}$).

With this, we define $\mathcal{B} := \{0, \ldots, 7\}$. Note that if no feasible solution is found using this default value, increasing it and solving the model again is possible. We now define a binary variable to capture how many times an example is used:

- $\forall t \in \mathcal{T}, \forall k \in \{1..N\}, \forall b \in \mathcal{B}$: q_{tkb} is 1 if training example k is used b times in tree t and 0 otherwise

The constraints of our model are as follows.

One-hot encoding:

- $\forall k \in \{1..N\}, \forall w \in vects: \quad \sum_{i \in w} x_{ki} = 1$

Each example is assigned to exactly one class:

- $\forall k \in \{1..N\}: \quad \sum_{c \in \mathcal{C}} z_{kc} = 1$

If an example is not assigned a given class, it cannot be used as that class in any tree:

- $\forall k \in \{1..N\}, \forall c \in \mathcal{C}$:

  if $z_{kc} = 0$ **then** $\sum_{t \in \mathcal{T}, v \in \mathcal{V}_t^L} y_{tvkc} = 0$

Each leaf must capture exactly the defined number of examples from each class:

- $\forall t \in \mathcal{T}, \forall v \in \mathcal{V}_t^L, \forall c \in \mathcal{C} : n_{tvc} = \sum_{k \in \{1..N\}} y_{tvkc}$

If an example is captured by a leaf, the associated conditions must be enforced on its features:

- $\forall t \in \mathcal{T}, \forall k \in \{1..N\}, \forall v \in \mathcal{V}_t^L$:

  if $\sum_{c \in \mathcal{C}} y_{tvkc} \geq 1$ **then** $\left(\bigwedge_{i \in \Phi_v^+} x_{ki} = 1 \right) \wedge \left(\bigwedge_{i \in \Phi_v^-} x_{ki} = 0 \right)$

The number of times a sample is used in a tree is consistent:

- $\forall t \in \mathcal{T}, \forall b \in \mathcal{B}, \forall k \in \{1..N\}$:

  $$\sum_{v \in \mathcal{V}_t^L, c \in \mathcal{C}} y_{tvkc} = b \iff q_{tkb} = 1$$

We implemented this model using the `OR-Tools` CP-SAT solver (Perron & Didier), which requires extra variables and constraints to be introduced to model some of the above conditions. These details are presented in Appendix C.

Maximum log-likelihood objective. Since the above model could have many possible solutions when using bagging, we orient the search towards the solutions (datasets) that are the most likely. For a given tree t, let p_b be the probability that a sample k is chosen exactly b times to train that tree. By defining $p^q_{tkb} = p_b$ if $q_{tkb} = 1$ and $p^q_{tkb} = 1$ otherwise, we can calculate the probability that the samples were chosen for the tree according to the q_{tkb} variables as $\prod_{k \in \{1..N\}} \prod_{b \in \mathcal{B}} p^q_{tkb}$. Therefore, considering the whole RF, the probability of a given solution is:

$$\prod_{t \in \mathcal{T}} \prod_{k \in \{1..N\}} \prod_{b \in \mathcal{B}} p^q_{tkb}.$$

Maximizing this probability is equivalent to maximizing its logarithm; in other words, maximizing:

$$\sum_{t \in \mathcal{T}} \sum_{k \in \{1..N\}} \sum_{b \in \mathcal{B}} \log\left(p^q_{tkb}\right) = \sum_{t \in \mathcal{T}} \sum_{k \in \{1..N\}} \sum_{b \in \mathcal{B}} \log\left(p_b\right) q_{tkb}.$$

Model simplifications when bagging is deactivated. RFs can be trained using random subsets of features for each split but considering all the examples in each tree. In such situations without bagging, the CP model can be significantly simplified. Variables y_{tvkc} will become binary and sum up to 1 for each tree $t \in \mathcal{T}$ and each example $k \in \{1..N\}$, since each example will be used exactly once in each tree. Also, we know in advance how many examples are of each class $c \in \mathcal{C}$. Therefore, to match that data, we may fix the variables z_{kc} in advance. Finally, q_{tkb} are always fixed since every example is used exactly once in each tree. Thus, the objective function becomes constant, and the problem reduces to the search for a feasible solution.

Reconstructing non-binary attributes. To streamline the presentation of the methodology and evaluation metric, we provided our model for the particular case of binary attributes. However, extending it to handle other types of attributes is possible with minimal changes, as detailed in Appendix D and implemented within our publicly available repository. In a nutshell, *categorical* attributes are typically one-hot encoded for tree ensembles and directly handled by our formulation. *Ordinal* features can be modeled as integer variables and only require a slight generalization of the constraints connecting the attributes' values to the assignment of the examples to the leaves. Finally, *numerical* attributes can also be reconstructed: although they take values in a continuous space, the number of splits within the forest is finite, and so is the number of different intervals in which they can lie. Leveraging this observation, we can use ordinal features to model such possible intervals in the reconstruction.

7. Experimental Study

Through extensive experimental analyses, we aim to evaluate the effectiveness and accuracy of the proposed reconstruction attack, named DRAFT *(Dataset Reconstruction Attack From Trained ensembles)*. We first detail the experimental setup before discussing the results.

7.1. Experimental Setup

Datasets. We rely on three popular datasets for binary classification in our experiments. We discretize each dataset's numerical attributes and one-hot encode the categorical ones. To keep a reasonably small number of features, we remove some attributes with the smallest support*. First, the COMPAS dataset (analyzed by Angwin et al. 2016) gathers records about criminal offenders in the Broward County of Florida collected from 2013 and 2014, with the task being

*Our binarized versions of these datasets are available in the supplementary material and will be available on our online repository upon publication.

recidivism prediction. Our preprocessed version includes 7,206 examples described by 15 binary attributes. Second, the UCI Adult Income dataset (Dua & Graff, 2017) contains data regarding the 1994 U.S. census to predict whether a person earns more than \$50K/year. After preprocessing, our dataset includes 48,842 examples and 20 binary features. Finally, we use the Default of Credit Card Client dataset (Yeh & hui Lien, 2009), to predict whether a person will default in payment (the next time they use their credit card). Our preprocessed version includes 29,986 examples and 22 binary attributes.

Reconstruction error evaluation. To assess the attack's success, we first compute the Manhattan distance between each reconstructed and original example. The resulting distance matrix then instantiates a minimum weight matching in bipartite graphs, also known as linear sum assignment problem, which we solve using the `Scipy` (Virtanen et al., 2020) Python library. Once the datasets are aligned, we then measure the proportion of binary attributes that differ between both.

Random reconstruction baseline. As mentioned in Section 3, reconstruction attacks rarely target reconstructing an entire training set, and none of them apply to our setup (*i.e.,* leveraging an RF to rebuild its complete training set). We then consider a baseline adversary with the same knowledge as ours (in particular, the number of examples N, the different attributes M including their one-hot encoding $vects$) except for the RF itself. The adversary then randomly guesses each attribute of each example, remaining consistent with the one-hot encoding information. The reconstruction error is finally assessed, as described in the previous paragraph. We average such computation over 100 random runs and report the average value. By comparing this baseline with the performances of our approach, one can then quantify how much additional information can be extracted from the RF.

Target RFs. To train our target models (*i.e.,* the RFs from which we attempt to reconstruct the training data), we use the popular implementation provided by the `scikit-learn` library. For each dataset, we learn RFs with varying parameters. More precisely, we use a number of trees $|\mathcal{T}| \in \{1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ with maximum depth $d_{max} \in \{\text{None}, 2, 3, 4, 5, 10\}$ (where None stands for no maximum depth constraint). For each experiment, we randomly sample 100 examples from the entire dataset to form a training set, and use the remaining ones as a test set to verify to what extent the models generalize. We repeat the experiment five times using different seeds for the random sampling, and report the average results and their standard deviation across the five runs.

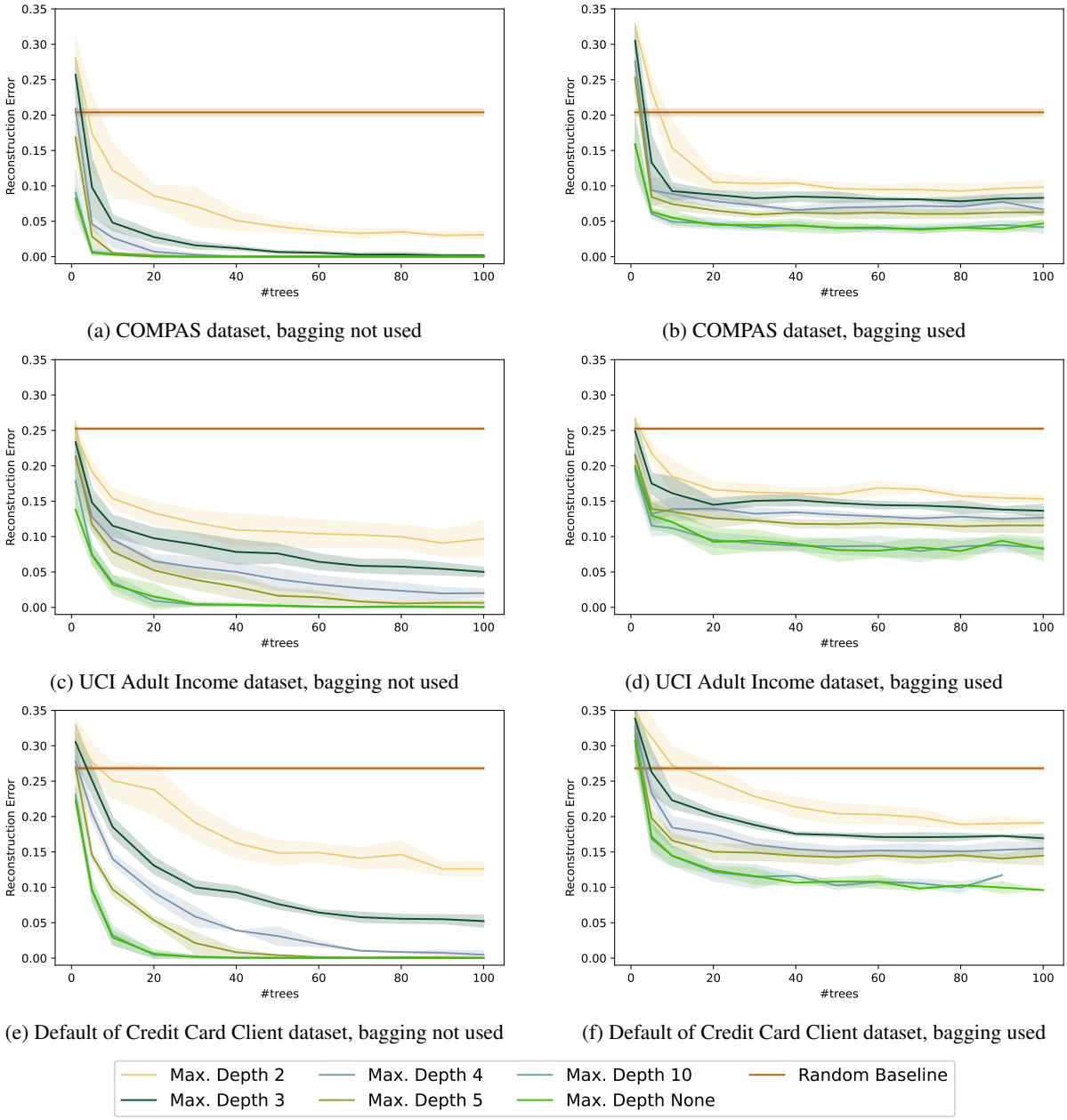
Dataset reconstruction. The proposed CP models described in Section 6 are solved using the `OR-Tools` CP-SAT solver (Perron & Didier) (v9). Each model resolution is limited to a maximum of five hours of CPU time using 16 threads with up to 6 GB of RAM for each thread. Note, however, that while the CP models handling bagging often reached this time limit, they usually were able to find feasible solutions in a much shorter time. All experiments are run on a computing cluster over a set of homogeneous nodes using Intel Platinum 8260 Cascade Lake @ 2.4GHz CPU.

All the material (source code and data sets) needed to reproduce our experiments is accessible at `https://github.com/vidalt/DRAFT` under a MIT license.

7.2. Results

The results of our experiments are reported in Figure 2 for all three datasets, with or without the use of bagging to train the target RFs. More precisely, we plot the average reconstruction error as a function of the number of trees $|\mathcal{T}|$, for several values of the trees' maximum depth d_{max}. We observe several trends that are consistent across all three datasets. In all cases, as expected, increasing the trees' depth or the number of trees in the forest decreases the reconstruction error as it provides more information regarding the training data. When bagging is not used to train the RFs, the reconstruction error reaches 0 in all cases for the deepest forests (recall that the default parameters of the `scikit-learn` library is no maximum depth constraint). This is not the case when using bagging. In such cases, the reconstruction error reaches a threshold and stops improving even for larger forests. In Appendix E, we further investigate the effect of bagging on protecting the training data against reconstruction attacks and conduct additional experiments. Our main finding is that this performance drop precisely comes from the difficulty of guessing how many times each example went through each tree: bagging intrinsically provides a form of protection regarding the training data. This is consistent with theoretical results stating that bagging provides (weak) differential privacy guarantees (Liu et al., 2021b).

We report in Tables 2 and 3 (respectively for a fixed number of trees $|\mathcal{T}| = 100$ and no fixed maximum depth, both corresponding to `scikit-learn`'s default values) the average run times for the reconstruction model without bagging, along with the number of times (#Runs) the solver was unable to find a feasible solution before timeout. When bagging is used, most runs attain the time limit and return a solution but cannot prove optimality. Run times in this context are not informative, so we only report the number of times the solver did not find any feasible solution before the time limit. Note that the few runs that did not produce a feasible solution are excluded from Figure 2. We observe



(a) COMPAS dataset, bagging not used

(b) COMPAS dataset, bagging used

(c) UCI Adult Income dataset, bagging not used

(d) UCI Adult Income dataset, bagging used

(e) Default of Credit Card Client dataset, bagging not used

(f) Default of Credit Card Client dataset, bagging used

Figure 2: Average reconstruction error as a function of the number of trees $|\mathcal{T}|$ within the target forest $\mathcal{T}$, for different maximum depth values d_{max} and for the random baseline.

from Table 2 that the formulation without bagging efficiently handles the problems that are under-constrained (shallow trees) or over-constrained (deep trees). Intermediate cases seem to require more computational effort, and in a few cases, the solver did not find a feasible solution. When using bagging, the size of the models seems to matter the most, as the solver only failed to find feasible solutions with the deepest forests. The same observation holds from Table 3, as the only runs for which the solver did not find a feasible solution are those with the largest numbers of trees. When not using bagging, the solution times scale approximately linearly with the number of trees. We report in Appendix F additional experiments regarding our method's scalability with respect to the number of training examples N. The results demonstrate its ability to reconstruct considerably larger datasets, with the reconstruction error remaining very small. Furthermore, while the size of the CP model's search space increases exponentially with the number of recon-

Table 2: Average run time and number of runs for which the solver did not come up with a feasible solution (**#Runs**), for a fixed number of trees $|\mathcal{T}| = 100$ (default value).

	Max. Depth	No Bagging		Bagging
		Avg. T (s)	#Runs	#Runs
COMPAS	2	9.5	0	0
	3	36.5	0	0
	4	45.7	0	0
	5	70.0	0	0
	10	110.9	0	0
	None	100.8	0	0
Adult	2	20.2	0	0
	3	81.5	0	0
	4	1943.5	0	0
	5	1290.7	0	0
	10	346.4	0	1/5
	None	196.3	0	1/5
Default Credit	2	35.5	0	0
	3	300.8	1/5	0
	4	6040.0	2/5	0
	5	1358.5	0	0
	10	382.8	0	0
	None	165.0	0	4/5

Table 3: Average run time and number of runs for which the solver did not come up with a feasible solution (**#Runs**), for no maximum depth constraint (default value).

	$\lvert\mathcal{T}\rvert$	No Bagging		Bagging
		Avg. T (s)	#Runs	#Runs
COMPAS	1	0.7	0	0
	10	8.4	0	0
	30	39.8	0	0
	50	53.0	0	0
	80	89.7	0	0
	100	100.8	0	0
Adult	1	0.8	0	0
	10	74.2	0	0
	30	84.9	0	0
	50	85.1	0	0
	80	119.9	0	0
	100	196.3	0	1/5
Default Credit	1	0.9	0	0
	10	129.7	0	0
	30	47.7	0	0
	50	66.2	0	2/5
	80	161.3	0	4/5
	100	165.0	0	4/5

structed training examples N, in practice, reconstruction time increases polynomially (approximately quadratic or sub-quadratic) with N.

As discussed in Section 3, most works in the reconstruction attacks literature only target reconstructing part of the dataset attributes (generally, a single one), assuming the others are publicly known. In Appendix G, we perform complementary experiments on such partial reconstruction. The results show that our approach successfully leverages knowledge of part of the dataset attributes, which results in lower error rates for the other ones.

8. Discussion and Conclusions

This study has shown that the structure of a trained RF can be exploited to reconstruct most (if not all) of its training data. It introduced a new paradigm of attack, leveraging mathematical programming tools to encode the structure of an RF and relying on a general-purpose CP solver to find the most likely reconstructions of the training data. Due to the high redundancy of RFs built using off-the-shelf ML libraries with their default parameters, the resulting problem is often strongly constrained, resulting in a high reconstruction rate. While theoretical $\mathcal{NP}$-completeness theorems indicate that such an attack may not be computationally tractable at scale, the tremendous progress in CP/MILP solvers has made it practical to solve larger and larger problems over time. Therefore, it may just be a question of time until data breaches happen for large datasets.

The fact that the proposed framework is based on mathematical programming techniques opens the door to many promising research perspectives. The approach could be tested on various types of attributes (numerical, categorical) without the need for feature binarization. Performance improvements could also be achieved through different problem reformulations or additional valid inequalities. Notably, one could leverage the information gain criterion used to select the splits while building the decision tree to eliminate combinations of attributes' values leading to different splits.

Our framework can also be used as a building block for other types of inference attacks, such as membership inference or property inference. Furthermore, we considered canonical RFs trained without privacy-preserving techniques, representing most of what popular libraries do by default. Investigating the effectiveness of common privacy-preserving mechanisms, such as the widely used differential privacy (Dwork et al., 2014), would bring additional insights. Though this may lead to difficult models, the proposed CP (or MILP) formulations could be extended to infer the noise added by the protection mechanisms on the released per-node counts (Fletcher & Islam, 2019; Dinur & Nissim, 2003). On the same line, adapting the formulation to work without the knowledge of the per-leaf per-class counts (hence only supposing that each leaf contains at least one example from the predicted class), or considering other gray-box setups, are interesting directions. Finally, another interesting direction is to apply the proposed methodology to other types of ensembles and ML models.

Impact Statement

ML models are commonly trained using large amounts of data, often including personal or private information. The flourishing literature on inference attacks against ML models showed that models might jeopardize their training data even when accessed in a black-box manner (*i.e.,* through a prediction API). Furthermore, transparency requirements encourage practitioners to either provide additional explanations for their model's decisions or to entirely release such models, potentially opening up to new attacks.

In this study, we have demonstrated that the structure of a trained RF can be leveraged to reconstruct most (if not all) of its training data. Importantly, our proposed method only leverages the information provided by popular libraries such as `scikit-learn`. While NP-harness theorems and scalability issues limit the current applicability of our approach, our results already demonstrate its effectiveness on datasets of practical significance. These findings underscore a critical vulnerability inherent to widely adopted ensemble methods, warranting attention and mitigation. The methods and experiments developed in this study have two main implications: (i) raising awareness against the privacy vulnerabilities of ensemble methods and (ii) providing promising research paths to stress test privacy-preserving mechanisms, aiming to protect such models before releasing them.

References

Aalmoes, J., Duddu, V., and Boutet, A. Dikaios: Privacy auditing of algorithmic fairness via attribute inference attacks. *arXiv preprint arXiv:2202.02242*, 2022.

Angwin, J., Larson, J., Mattu, S., and Kirchner, L. Machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. propublica (2016). *ProPublica, May*, 23, 2016.

Aziz, H., Cseh, Á., Dickerson, J. P., and McElfresh, D. C. Optimal kidney exchange with immunosuppressants. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 21–29. AAAI Press, 2021. doi: 10.1609/AAAI.V35I1.16073. URL https://doi.org/10.1609/aaai.v35i1.16073.

Carlini, N., Chien, S., Nasr, M., Song, S., Terzis, A., and Tramèr, F. Membership inference attacks from first principles. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pp. 1897–1914. IEEE, 2022. doi: 10.1109/SP46214.2022.9833649. URL https://doi.org/10.1109/SP46214.2022.9833649.

Cristofaro, E. D. An overview of privacy in machine learning. *CoRR*, abs/2005.08679, 2020. URL https://arxiv.org/abs/2005.08679.

Dinur, I. and Nissim, K. Revealing information while preserving privacy. In Neven, F., Beeri, C., and Milo, T. (eds.), *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pp. 202–210. ACM, 2003. doi: 10.1145/773153.773173. URL https://doi.org/10.1145/773153.773173.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Dwork, C., McSherry, F., and Talwar, K. The price of privacy and the limits of lp decoding. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pp. 85–94, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936318. doi: 10.1145/1250790.1250804. URL https://doi.org/10.1145/1250790.1250804.

Dwork, C., Roth, A., et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Dwork, C., Smith, A., Steinke, T., and Ullman, J. Exposed! a survey of attacks on private data. *Annual Review of Statistics and Its Application*, 4(1):61–84, 2017. doi: 10.1146/annurev-statistics-060116-054123. URL https://doi.org/10.1146/annurev-statistics-060116-054123.

Ferry, J., Aïvodji, U., Gambs, S., Huguet, M.-J., and Siala, M. Exploiting fairness to enhance sensitive attributes reconstruction. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, 2023. URL https://openreview.net/forum?id=tOVr0HLaFz0.

Ferry, J., Aïvodji, U., Gambs, S., Huguet, M.-J., and Siala, M. Probabilistic Dataset Reconstruction from Interpretable Models. In *2nd IEEE Conference on Secure and Trustworthy Machine Learning*, Toronto, Canada, April 2024. URL https://hal.science/hal-04189566.

Fletcher, S. and Islam, M. Z. Decision tree classification with differential privacy: A survey. *ACM Comput. Surv.*, 52(4), aug 2019. ISSN 0360-0300. doi: 10.1145/3337064. URL https://doi.org/10.1145/3337064.

Fredrikson, M., Lantz, E., Jha, S., Lin, S. M., Page, D., and Ristenpart, T. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In Fu, K. and Jung, J. (eds.), *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pp. 17–32. USENIX Association, 2014. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew.

Fredrikson, M., Jha, S., and Ristenpart, T. Model inversion attacks that exploit confidence information and basic countermeasures. In Ray, I., Li, N., and Kruegel, C. (eds.), *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pp. 1322–1333. ACM, 2015. doi: 10.1145/2810103.2813677. URL https://doi.org/10.1145/2810103.2813677.

Gambs, S., Gmati, A., and Hurfin, M. Reconstruction attack through classifier analysis. In Cuppens-Boulahia, N., Cuppens, F., and García-Alfaro, J. (eds.), *Data and Applications Security and Privacy XXVI - 26th Annual IFIP WG 11.3 Conference, DBSec 2012, Paris, France, July 11-13,2012. Proceedings*, volume 7371 of *Lecture Notes in Computer Science*, pp. 274–281. Springer, 2012. doi: 10.1007/978-3-642-31540-4_21. URL https://doi.org/10.1007/978-3-642-31540-4_21.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Haim, N., Vardi, G., Yehudai, G., Shamir, O., and Irani, M. Reconstructing training data from trained neural networks. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Hamman, F., Chen, J., and Dutta, S. Can querying for bias leak protected attributes? achieving privacy with smooth sensitivity. In *NeurIPS 2022 Workshop on Algorithmic Fairness through the Lens of Causality and Privacy*, 2022.

Hu, H. and Lan, C. Inference attack and defense on the distributed private fair learning framework. In *The AAAI Workshop on Privacy-Preserving Artificial Intelligence*, 2020.

Liu, B., Ding, M., Shaham, S., Rahayu, W., Farokhi, F., and Lin, Z. When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(2): 1–36, 2021a.

Liu, H., Jia, J., and Gong, N. Z. On the intrinsic differential privacy of bagging. In Zhou, Z. (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 2730–2736. ijcai.org, 2021b. doi: 10.24963/IJCAI.2021/376. URL https://doi.org/10.24963/ijcai.2021/376.

Parmentier, A. and Vidal, T. Optimal counterfactual explanations in tree ensembles. In *International Conference on Machine Learning*, pp. 8422–8431. PMLR, 2021.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Perron, L. and Didier, F. CP-SAT. URL https://developers.google.com/optimization/cp/cp_solver/.

Phong, L. T., Aono, Y., Hayashi, T., Wang, L., and Moriai, S. Privacy-preserving deep learning: Revisited and enhanced. In Batten, L., Kim, D. S., Zhang, X., and Li, G. (eds.), *Applications and Techniques in Information Security - 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6-7, 2017, Proceedings*, volume 719 of *Communications in Computer and Information Science*, pp. 100–110. Springer, 2017. doi: 10.1007/978-981-10-5421-1_9. URL https://doi.org/10.1007/978-981-10-5421-1_9.

Rigaki, M. and Garcia, S. A survey of privacy attacks in machine learning. *CoRR*, abs/2007.07646, 2020. URL https://arxiv.org/abs/2007.07646.

Rossi, F., Van Beek, P., and Walsh, T. Constraint programming. *Foundations of Artificial Intelligence*, 3:181–211, 2008.

Salem, A., Bhattacharya, A., Backes, M., Fritz, M., and Zhang, Y. Updates-leak: Data set inference and reconstruction attacks in online learning. In Capkun, S. and Roesner, F. (eds.), *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pp. 1291–1308. USENIX Association, 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/salem.

Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 3–18. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.41. URL https://doi.org/10.1109/SP.2017.41.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Yeh, I.-C. and hui Lien, C. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2, Part 1):2473–2480, 2009. ISSN 0957-4174. doi: 10.1016/j.eswa.2007.12.020.

Zhou, Z.-H. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition, 2012. ISBN 1439830037.

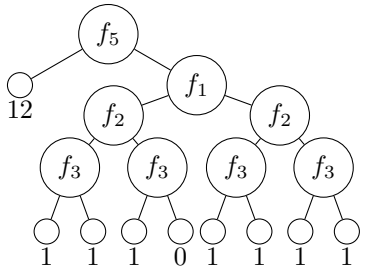
A. Proof of NP-Hardness (Theorem 5.1)

The description of the construction of the instance of (DRP) can be found in the main text. We start by providing an example to clarify the construction.

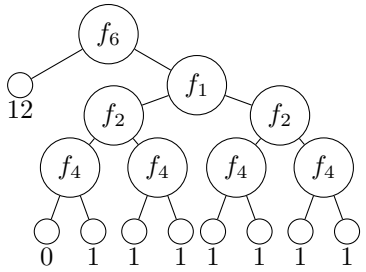
Consider the following 3-SAT instance with $|L| = 3$ clauses and $V = 4$ variables:

$$(u_1 \vee \bar{u_2} \vee \bar{u_3}) \wedge (u_1 \vee u_2 \vee u_4) \wedge (\bar{u_2} \vee \bar{u_3} \vee \bar{u_4}) \tag{1}$$

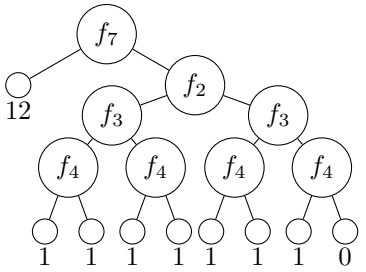
Figure 3 shows the constructed (DRP) instance arising from it.



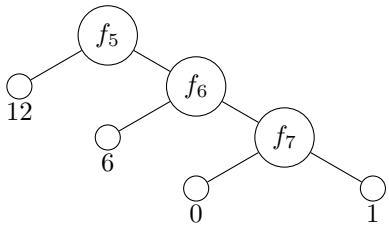
(a) Tree corresponding to clause 1



(b) Tree corresponding to clause 2



(c) Tree corresponding to clause 3



(d) Auxiliary tree

Figure 3: Example of (DRP) instance originating from (1). The left branches correspond to setting the feature to 0. The right ones set the feature to 1. The numbers below are the n_{lvc} values.

Theorem 5.1 follows from Claims A.1 and A.2.

Claim A.1. *If the 3-SAT instance is a YES instance, then (DRP) is feasible.*

Proof. Suppose that there is an assignment of values to the V variables of 3-SAT so that all clauses L are satisfied.

Then for each clause $l \in L$, pick the leaf v of the perfect binary subtree that corresponds to the assignment of variables in 3-SAT. We set $y_{lv(6|L|+1)} = 1$ and $y_{lv'(6|L|+1)} = 0$ for all $v' \in \mathcal{V}_l^L$, $v' \neq v$. We set the first V attributes of $x_{6|L|+1}$ to match the assignment of the V variables that satisfy 3-SAT. We set the attributes $V + 1, \ldots, V + |L|$ to 1.

For the remaining examples, we set the attributes $V + l$ of examples $6(l - 1) + 1, \ldots, 6l$ to 1, and all other attributes in $V + 1, \ldots, V + |L|$ to 0. This is done for all $l = 1, \ldots, |L|$. In addition, we set the remaining attributes to match one of the leaves that have $n_{lvc} = 1$ but don't correspond to the assignment of variables in 3-SAT and set the corresponding y_{lvk} to 1.

This can be easily checked to be feasible for (DRP).

For example, in Figure 3, if the 3-SAT assignment is $u_1 = \text{TRUE}$, $u_2 = u_3 = u_4 = \text{FALSE}$, then we would have the solution to (DRP) shown in Table 4.

□

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
x_1	0	0	0	**1**	1	0	0
x_2	0	0	1	**1**	1	0	0
x_3	0	1	0	**1**	1	0	0
x_4	1	0	1	**1**	1	0	0
x_5	1	1	0	**1**	1	0	0
x_6	1	1	1	**1**	1	0	0
x_7	0	0	**1**	1	0	1	0
x_8	0	1	**1**	0	0	1	0
x_9	0	1	**1**	1	0	1	0
x_{10}	1	0	**1**	1	0	1	0
x_{11}	1	1	**1**	0	0	1	0
x_{12}	1	1	**1**	1	0	1	0
x_{13}	**1**	0	0	1	0	0	1
x_{14}	**1**	0	1	0	0	0	1
x_{15}	**1**	0	1	1	0	0	1
x_{16}	**1**	1	0	0	0	0	1
x_{17}	**1**	1	0	1	0	0	1
x_{18}	**1**	1	1	0	0	0	1
x_{19}	*1*	*0*	*0*	*0*	1	1	1

Table 4: Solution to (DRP) constructed from 3-SAT solution. The entries in **bold** are arbitrary. The *italicized* entries encode the solution of the corresponding 3-SAT problem.

Claim A.2. *If (DRP) is feasible then the 3-SAT instance is a YES instance.*

Proof. If (DRP) is feasible, then there exists one example which has features $V+1, \ldots, V+|L|$ equal to 1. This comes from the rightmost node of the auxiliary tree.

Without loss of generality, assume that such example is $x_{6|L|+1}$ (*e.g.,* x_{19} in Table 4).

Then in each of the trees $l \in L$, $x_{6|L|+1}$ must have gone to the right branch at the root. In this case, we know that $x_{6|L|+1}$ must fall into one of the leaves of the perfect binary tree that corresponds to a truth assignment that makes the clause l satisfied.

A solution to 3-SAT can then be constructed by looking at the first V components of $x_{6|L|+1}$. □

B. Mixed-Integer Linear Programming Formulation

We show how the reconstruction problem can be alternatively formulated as a Mixed-Integer Linear Program (MILP), permitting the use of alternative solution algorithms. In a MILP, all variables can be continuous or integers, but all constraints and the (optional) objective function must be linear in the decision variables. This restriction is not imposed in Constraint Programming (CP). Consequently, we must *linearize* some of the expressions required to model our reconstruction problem using additional variables. We describe the MILP formulation for the scenario where bagging is not used to train the target random forests, before performing some empirical evaluation of its performance.

B.1. Model Formulation (Without Bagging)

We present here a MILP model for the DRP. Our MILP model for reconstructing the training set of a given random forest extends the OCEAN framework (Parmentier & Vidal, 2021), which was proposed to generate optimal counterfactual explanations for tree ensembles. In a nutshell, OCEAN leverages MILP to encode the structure of the trees within the forest, and aims at finding an example as close as possible from a query example $\mathbf{x}_k$ but with a different classification. Rather than determining the attributes' vector of a single example (the generated counterfactual), we aim to reconstruct the features' vector of all the N training examples simultaneously.

We now introduce some additional notation. For each tree $t \in \mathcal{T}$, we define $\mathcal{D}_t$ as the set of all the depths reached in t, and $\forall d \in \mathcal{D}_t$, $\mathcal{V}^I_{td}$ is the set of internal nodes at depth d in t. As mentioned in Section 6, without bagging, one can fix in advance the set of decisions z_{kc} (*i.e.,* if an example k is from class c). Let $Z_c = \{k \in \{1..N\} : z_{kc} = 1\}$ be the set of indices of examples belonging to class c, and $\mathcal{V}^I_{ti}$ be the set of nodes within tree t splitting on feature i. W.l.o.g., we assume that the indices in Z_c are consecutive.

We first define decision variables that will model the path of each example through each tree:

- $\forall t \in \mathcal{T}, \forall d \in \mathcal{D}_t, \forall k \in \{1..N\} : \lambda_{tdk} \in \{0, 1\}$ takes value 1 if example k takes the left path at depth d of the tree t, and 0 otherwise. The value is free if the path doesn't go this deep.
- $\forall t \in \mathcal{T}, \forall v \in \mathcal{V}^I_t \bigcup \mathcal{V}^L_t, \forall k \in \{1..N\}$: $y_{tvk} \in [0; 1]$ takes value 1 if example k reaches node v of the tree t, 0 otherwise (note that the integrality is forced by the previous variables)
- $\forall k \in \{1..N\}, \forall i \in \{1..M\}$: $x_{ki} \in \{0; 1\}$ is the value of feature i for example k in the reconstruction

First, the following constraints correspond to the one-hot encoding of the features:

$$\sum_{i \in w} x_{ki} = 1 \qquad \forall k \in \{1..N\}, \forall w \in vects$$

We then use the following constraints to model the flow of the examples through the trees:

$$y_{t1k} = 1 \qquad \forall t \in \mathcal{T}, \forall k \in \{1..N\} \quad (2)$$

$$y_{tvk} = y_{tl(v)k} + y_{tr(v)k} \qquad \forall t \in \mathcal{T}, \forall v \in \mathcal{V}^I_t, \forall k \in \{1..N\} \quad (3)$$

$$\sum_{v \in \mathcal{V}^I_{td}} y_{tl(v)k} \leq \lambda_{tdk} \qquad \forall t \in \mathcal{T}, \forall d \in \mathcal{D}_t, \forall k \in \{1..N\} \quad (4)$$

$$\sum_{v \in \mathcal{V}^I_{td}} y_{tr(v)k} \leq 1 - \lambda_{tdk} \qquad \forall t \in \mathcal{T}, \forall d \in \mathcal{D}_t, \forall k \in \{1..N\} \quad (5)$$

In a nutshell, because we consider the case without the use of bagging, each example has one associated unit of flow at the root of each tree. This flow is encoded by continuous variables (which are easier to handle for the solver than integer/binary ones). All the flow is then directed through the tree, by going either left or right at each split node, until it reaches a leaf.

We then link these flows to the values taken by the features of the examples through the following constraints:

$$x_{ki} \leq 1 - y_{tl(v)k} \qquad \forall k \in \{1..N\}, \forall i \in \{1..M\}, \forall t \in \mathcal{T}, \forall v \in \mathcal{V}^I_{ti} \quad (6)$$

$$y_{tr(v)k} \leq x_{ki} \qquad \forall k \in \{1..N\}, \forall i \in \{1..M\}, \forall t \in \mathcal{T}, \forall v \in \mathcal{V}^I_{ti} \quad (7)$$

Finally, we connect these flows to the support of each node within the trees (recall that because we consider the case without bagging, z_{kc} is a prefixed constant, and hence the computation is linear in the decision variables y_{tvk}):

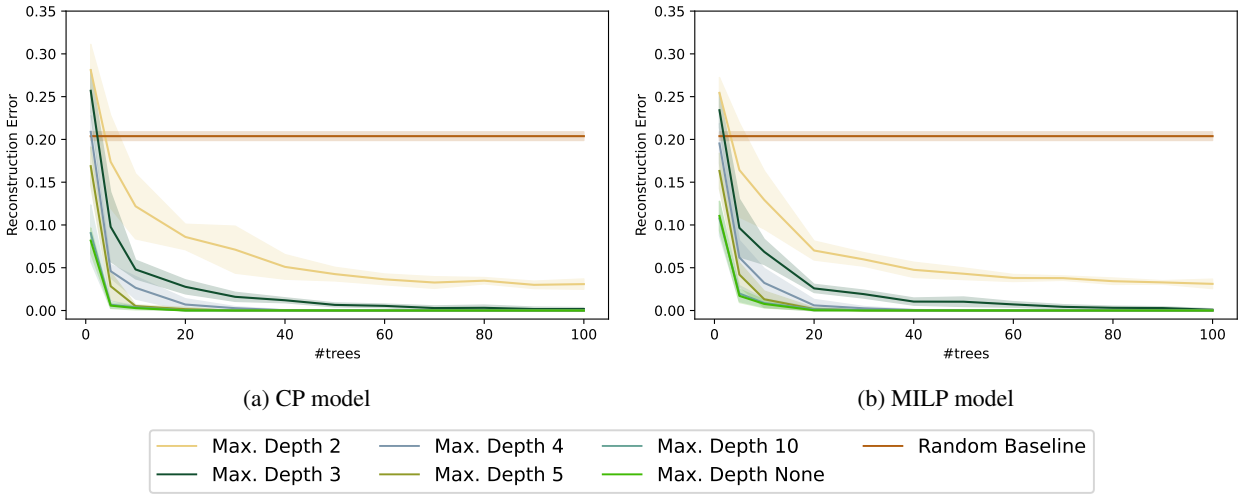
$$n_{tvc} = \sum_{k \in \{1..N\}} y_{tvk} z_{kc}, \qquad \forall t \in \mathcal{T}, \forall v \in \mathcal{V}^I_t \bigcup \mathcal{V}^L_t, \forall c \in \mathcal{C} \quad (8)$$

Note that we additionally use the following constraints for symmetry breaking in each class:

$$\sum_{i \in \{1..M\}} 2^{i-1} x_{ki} \leq \sum_{i \in \{1..M\}} 2^{i-1} x_{(k+1)i} \qquad \forall c \in \mathcal{C}, \forall k \in Z_c \setminus \{\ell_c\} \quad (9)$$

where ℓ_c is the last index in Z_c.

In the next subsection, we empirically evaluate our proposed MILP model and compare it to the CP formulation introduced in the main paper. Note that extending the proposed MILP to handle bootstrap sampling is possible, but the number of required variables increases prohibitively in order to preserve linearity, limiting the scalability of the approach.



(a) CP model

(b) MILP model

Figure 4: Average reconstruction error as a function of the number of trees $|\mathcal{T}|$ within the attacked forest $\mathcal{T}$, for different maximum depth values d_{max} and for the random baseline. For the experiments on the COMPAS dataset, not using bagging, we report the results obtained using either the CP model (Section 6) or the MILP one (Section B.1).

B.2. Empirical Evaluation

We run the reconstruction experiments on the COMPAS dataset without bagging as described in Section 7.1, using our MILP formulation, and compare the results with those obtained using our CP model (which are reported in Section 7.2). The MILP models are solved using the `Gurobi` solver (Gurobi Optimization, LLC, 2023) through its Python binding[†], all the other experimental parameters remaining unchanged.

The results are reported in Figure 4, and their run times are compared in Table 5. Note that the results for the CP model are those presented in Figure 2a, repeated here to ease comparison. Comparing the different curves (which correspond to different maximum depth constraints) between Figures 4a and 4b, we see that both approaches successfully solve the dataset reconstruction problem on COMPAS without the use of bagging to train the target random forests. Intuitively, the two feasibility models encode the same information, and define the same set of feasible reconstructions. Because they use different techniques to represent and explore it, they may end up with different reconstructions, but there is no *a priori* reason for one to outperform the other systematically, and as observed in our experiments, their reconstruction performances are generally similar.

Nevertheless, Table 5 highlights significant solution-time differences between the CP and MILP approaches. The solution times of both approaches are of the same order of magnitude for shallow trees. However, as the depth of the trees grows, the solution time increases more quickly with the MILP than with the CP model. For instance, on average, the MILP formulation requires over three times more CPU time than the CP one when no maximum depth constraint is set. More importantly, the solution times are considerably less stable when using the MILP, resulting in larger maximum run times. In the most extreme case, the MILP exceeds 75 minutes, contrasting sharply with the CP model's consistently modest durations, never surpassing three minutes. As discussed in the previous subsection, the MILP is also less prone to be extended to the setup where bagging is used to train the target random forests. These observations led us to rely on the CP model in the main paper.

C. Implementation Details for the CP Model

As mentioned in Section 6, in order to implement the CP model using the `OR-Tools` CP-SAT solver, some other variables and constraints are needed due to the specificities of the solution software. We discuss these technical aspects in this appendix.

First, CP-SAT only allows implication constraints with a literal being the cause of the implication. Therefore, we had to rely

[†] https://pypi.org/project/gurobipy/

Table 5: Reconstruction times for the experiments on the COMPAS dataset, without the use of bagging to train the target random forests. For both the CP and the MILP models, we consider all the measured runtimes (*i.e.,* for the 5 seeds and the 12 different numbers of trees within the forests) for a given maximum depth constraint. We report their average value, along with the standard deviation, the minimum time and the maximum one.

Max. Depth	Method	Reconstruction Times (s)			
		Avg	Std	Min	Max
2	CP	4.7	3.8	0.1	17.0
	MILP	5.4	6.1	0.1	31.2
3	CP	14.1	12.9	0.1	48.4
	MILP	10.3	21.9	0.2	162.5
4	CP	25.2	18.9	0.2	58.9
	MILP	24.7	52.6	0.2	302.7
5	CP	34.3	23.4	0.3	85.8
	MILP	26.4	60.4	0.3	418.6
10	CP	53.7	39.2	0.5	160.6
	MILP	77.6	312.7	2.6	2471.2
None	CP	49.7	35.5	0.6	142.0
	MILP	188.3	791.2	3.3	4521.8

on auxiliary binary variables:

- For all $t \in \mathcal{T}$, $v \in \mathcal{V}_t^L$, $k \in \{1..N\}$: w_{tvk} is 1 if example k is classified by leaf v of tree t; 0 otherwise.

Second, the relationship between the q_{tkb} and y_{tvkc} also cannot be enforced directly. It can only be done via another set of auxiliary variables:

- $\eta_{tk} \in \mathbb{Z}_+$ represents the number of times example k is used in tree t

To model the relationship between w and y, we add the constraints:

- **if** $w_{tvk} = 0$ **then** $y_{tvkc} = 0, \forall t \in \mathcal{T}, v \in \mathcal{V}_t^L, k \in \{1..N\}, c \in \mathcal{C}$
- **if** $w_{tvk} = 1$ **then** $\sum_{c \in \mathcal{C}} y_{tvkc} \geq 1, \forall t \in \mathcal{T}, v \in \mathcal{V}_t^L, k \in \{1..N\}$

These are explicitly added in CP-SAT using the `OnlyEnforceIf` function that allows a linear constraint only to be enforced if a boolean variable is TRUE.

With these variables, the constraints that were presented before as

$$\textbf{if } \sum_{c \in \mathcal{C}} y_{tvkc} \geq 1 \textbf{ then } \quad \left(\bigwedge_{i \in \Phi_v^+} x_{ki} = 1 \right) \wedge \left(\bigwedge_{i \in \Phi_v^-} x_{ki} = 0 \right)$$

will now be implemented as:

- $\forall t \in \mathcal{T}, \forall k \in \{1..N\}, \forall v \in \mathcal{V}_t^L$: **if** $w_{tvk} = 1$ **then** $\quad \left(\bigwedge_{i \in \Phi_v^+} x_{ki} = 1 \right) \wedge \left(\bigwedge_{i \in \Phi_v^-} x_{ki} = 0 \right)$

These also can be explicitly added in CP-SAT using the `OnlyEnforceIf` function.

To model the correct relationship between y_{tvkc} and η_{tk} variables, we add the constraints:

- For all $k \in \{1..N\}$, $t \in \mathcal{T}$: $\eta_{tk} = \sum_{c \in \mathcal{C}} \sum_{v \in \mathcal{V}_t^L} y_{tvkc}$

Now, the constraints

$$\sum_{v \in \mathcal{V}_t^L, c \in \mathcal{C}} y_{tvkc} = b \iff q_{tkb} = 1$$

can be implemented in CP-SAT using the constraints

- For all $t \in \mathcal{T}$, $k \in \{1..N\}$: `AddMapDomain`$(\eta_{tk}, [q_{tkb}]_{b \in \mathcal{B}})$

These constraints receive the integer variable η_{tk} and the vector of binary variables $[q_{tkb}]_{b \in \mathcal{B}}$ and enforce that $\eta_{tk} = b$ if and only if $q_{tkb} = 1$.

D. Extending the CP Model to Handle Non-Binary Attributes

The Constraint Programming (CP) model presented in Section 6 is able to reconstruct binary attributes. While we focused on this case to streamline the presentation of the methodology and evaluation metrics, our framework can be extended to handle other types of attributes, as explained in this appendix section.

Discrete attributes take values in a finite domain. If these values can be ordered, the attribute is coined as **ordinal**, and if they can not (*i.e.,* if they represent categories), it is called **categorical**. These two types of discrete attributes can be handled by DRAFT as detailed hereafter.

Categorical Attributes. Because the different possible values of a categorical attribute can not be ordered, it wouldn't make sense to verify whether they are greater or smaller than a given split value, even if the different categories can be represented using different integer values. Indeed, such attributes must usually be one-hot encoded (*i.e.,* with one separate binary attribute for each possible category, all the created binary attributes summing up to one) and are hence directly and efficiently handled using the formulation described in Section 6.

Ordinal Attributes. Ordinal attributes can be used directly in tree ensembles (without one-hot encoding) since an order relation permits defining meaningful splits. They can be handled naturally using DRAFT. More precisely, using the CP formulation provided in Section 6, the reconstruction variables $\{x_{ki}\}_{k \in \{1..N\}}$ associated to each ordinal attribute i must be declared as integers (which are directly supported in Constraint Programming). Furthermore, the constraint enforcing the conditions associated to a branch leading to a leaf v if an example k is assigned to that leaf in tree t must be slightly generalized. We now define Φ_v^+ as the set of attribute-value tuples (i, a) such that attribute i must be greater than a for an example to fall into leaf v. Similarly, Φ_v^- is now the set of attribute-value tuples (i, a) such that attribute i must be smaller or equal to a for an example to fall into v. Note that this slight generalization also encompasses the binary attribute case, where the split value a is usually fixed to 0.5. The generalized constraint then becomes:

$$\forall t \in \mathcal{T}, \forall k \in \{1..N\}, \forall v \in \mathcal{V}_t^L : \textbf{if } \sum_{c \in \mathcal{C}} y_{tvkc} \geq 1 \textbf{ then } \quad \left(\bigwedge_{(i,a) \in \Phi_v^+} x_{ki} > a \right) \wedge \left(\bigwedge_{(i,a) \in \Phi_v^-} x_{ki} \leq a \right)$$

All the other variables and constraints remaining unchanged, the model provided in Section 6 can effectively be used to reconstruct discrete (categorical or ordinal) features.

Contrary to discrete attributes, **numerical** ones take values in a continuous space. If the underlying mathematical programming framework can encode continuous variables (which is, for instance, the case of Mixed-Integer Linear Programming), then numerical attributes can be handled just like ordinal ones, using the methodology described in the previous paragraph. This is not the case in Constraint Programming, but numerical attributes can still be reconstructed effectively, as discussed hereafter.

Numerical Attributes. While numerical attributes take values in a continuous space, the number of nodes within a decision tree (hence within a random forest) is finite, and so the number of split values regarding any specific attribute is also finite. Then, the number of possible values or intervals for a given reconstructed numerical attribute is also discrete. Indeed, the knowledge acquired from a random forest can indicate that an example's numerical attribute lies within a given interval (between two split values), but in the general case, it does not indicate which particular value within this interval it should take. We leverage such discretization to reconstruct numerical attributes as follows:

1. We parse all the trees in the forest and build the ordered list of all the different split values regarding each numerical feature i:
$$\mathcal{A}_i = \texttt{sorted}\left(\left\{a : (i,a) \in (\Phi_v^+ \cup \Phi_v^-)_{t\in\mathcal{T}, v\in\mathcal{V}_t^L}\right\}\right)$$
2. We concatenate this ordered list of split values to the (possibly infinite) lower and upper bounds on the domain of attribute i:
$$\mathcal{I}_i = \{\texttt{lower\_bound}(i)\} \cup \mathcal{A}_i \cup \{\texttt{upper\_bound}(i)\}$$
Intuitively, $\mathcal{I}_i$ defines the possible intervals for attribute i given the splits within the forest.
3. To build the reconstruction model, we encode each numerical feature i as an ordinal (integer) one i', taking values in $\{1..(|\mathcal{A}_i|+1)\}$. The value of ordinal attribute i' in the reconstruction performed by the CP model (variables $\{x_{ki'}\}_{k\in\{1..N\}}$) will then be used to retrieve the interval in which numerical feature i lies. Note that the first $|\mathcal{A}_i|$ values correspond to the different split values for attribute i while the $(|\mathcal{A}_i|+1)$ one encodes the situation where i is strictly greater than its largest split value. Note that since the constraints associated to the splits are either "strictly greater than" or "smaller or equal to", it is not possible to forbid the smallest split value, and so we do not need to insert an additional value before it.
4. For each split-value tuple (i,a) associated to numerical feature i in $(\Phi_v^+ \cup \Phi_v^-)$, we create for the corresponding integer feature i' a split-value tuple (i',a') such that a' is the index of a in the ordered list $\mathcal{A}_i$. Using such split-value tuple, attribute i' can then be handled just like other ordinal features, as aforementioned.
5. Once the reconstruction is done, we have to connect the value of i' to that of the actual (continuous) numerical feature i. If for $k \in \{1..N\}$, $x_{ki'} = a' \in \{1..(|\mathcal{A}_i|+1)\}$ in the reconstruction performed by the CP model, we set the value of the corresponding reconstructed (continuous) attribute to the mean between the $a'-1$ and a' split values (*i.e.,* to $\frac{\mathcal{I}_i[a'] + \mathcal{I}_i[a'+1]}{2}$ - the difference in indices comes from the fact that $\mathcal{I}_i$ starts with an additional element, corresponding to the attribute's lower bound). Note that if the lower and (or) upper bounds of i are infinite, we can choose any arbitrary value compatible with the splits' information.

E. The Impact of Bagging on Data Protection

Our results (reported in Section 7) show that if bagging is not used, then all of the data can be recovered with just a few trees in the RF. However, with bagging, the CP model from Section 6 can recover around 90-95% of the data, even with many trees. In this appendix, we present experiments designed to understand why we could not recover 100% of the data with bagging.

One of the complicating aspects of bagging is that the knowledge of how many times a sample has been classified within a given leaf v of a given tree t is lost. With this in mind, we posed the following question:

- Considering the CP model from Section 6, if we know in advance the values of y_{tvkc} (that is, how many times sample k is classified within a given leaf v of tree t as part of class c) how much reduction can be observed in the reconstruction error?

Note that, if the values of y_{tvkc} are given, then the values of z_{kc} and q_{tkb} can be deducted. So the only remaining issue is to determine the x_{ki} values and the only constraints that need to be enforced on those are the one-hot encoding constraints and the *leaf-consistency* constraints:

- $\forall t \in \mathcal{T}, \forall k \in \{1..N\}, \forall v \in \mathcal{V}_t^L$: **if** $\sum_{c\in\mathcal{C}} y_{tvkc} \geq 1$ **then** $\left(\bigwedge_{i\in\Phi_v^+} x_{ki} = 1\right) \wedge \left(\bigwedge_{i\in\Phi_v^-} x_{ki} = 0\right)$

Let $\phi^y_{tk} := \left\{ v \in \mathcal{V}^L_t : \sum_{c \in \mathcal{C}} y_{tvkc} \geq 1 \right\}$ be the (possibly empty) set of leaves of tree t for which example k has been used. While the leaf-consistency constraints fix all attributes i in $\Phi^+_v \cup \Phi^-_v$ for $v \in \phi^y_{tk}$, any feature that does not appear in any such sets (call them *free attributes*) can be arbitrarily set without changing the likelihood of the solution. And so, the fact that a free attribute is guessed correctly can be attributed to luck and should not be seen as a positive aspect of the CP model.

Formally, the fixed attributes for example $k \in \{1..N\}$ are

$$\mathcal{F}_k := \bigcup_{t \in \mathcal{T}} \bigcup_{v \in \phi^y_{tk}} \left(\Phi^+_v \cup \Phi^-_v \right)$$

and the free attributes are $\bar{\mathcal{F}}_k := \{1..M\} \setminus \mathcal{F}_k$.

Let $\{\mathbf{x}_k; c_k\}_{k=1}^N$ be the training set which was used to train the random forest (and which we are trying to recover). With this we define x^y_k as follows:

$$x^y_{ki} := \begin{cases} 1, & \text{if } i \in \mathcal{F}_k \cap \Phi^+_v \text{ for some } t \in \mathcal{T}, v \in \phi^y_{tk} \\ 0, & \text{if } i \in \mathcal{F}_k \cap \Phi^-_v \text{ for some } t \in \mathcal{T}, v \in \phi^y_{tk} \\ 1 - \mathbf{x}_{ki}, & \text{otherwise} \end{cases}$$

x^y_k can be thought of as the solution that is consistent with the y variables on all fixed attributes and incorrectly guesses the values of all free attributes, so the worst possible solution that is consistent with y.

Our *benchmark* experiment can now be described as follows:

- Run the CP model of Section 6 with $x_{ki} = \mathbf{x}_{ki}$ and $z_{kc_k} = 1$ for all $k \in \{1..N\}, i \in \{1..M\}$.
- Obtain from the solution of such model the values of the y_{tvkc} variables, for all $t \in \mathcal{T}$, $v \in \mathcal{V}^L_t$, $k \in \{1..N\}$, $c \in \mathcal{C}$.
- Output the set of solutions $\{x^y_k\}_{k=1}^N$.

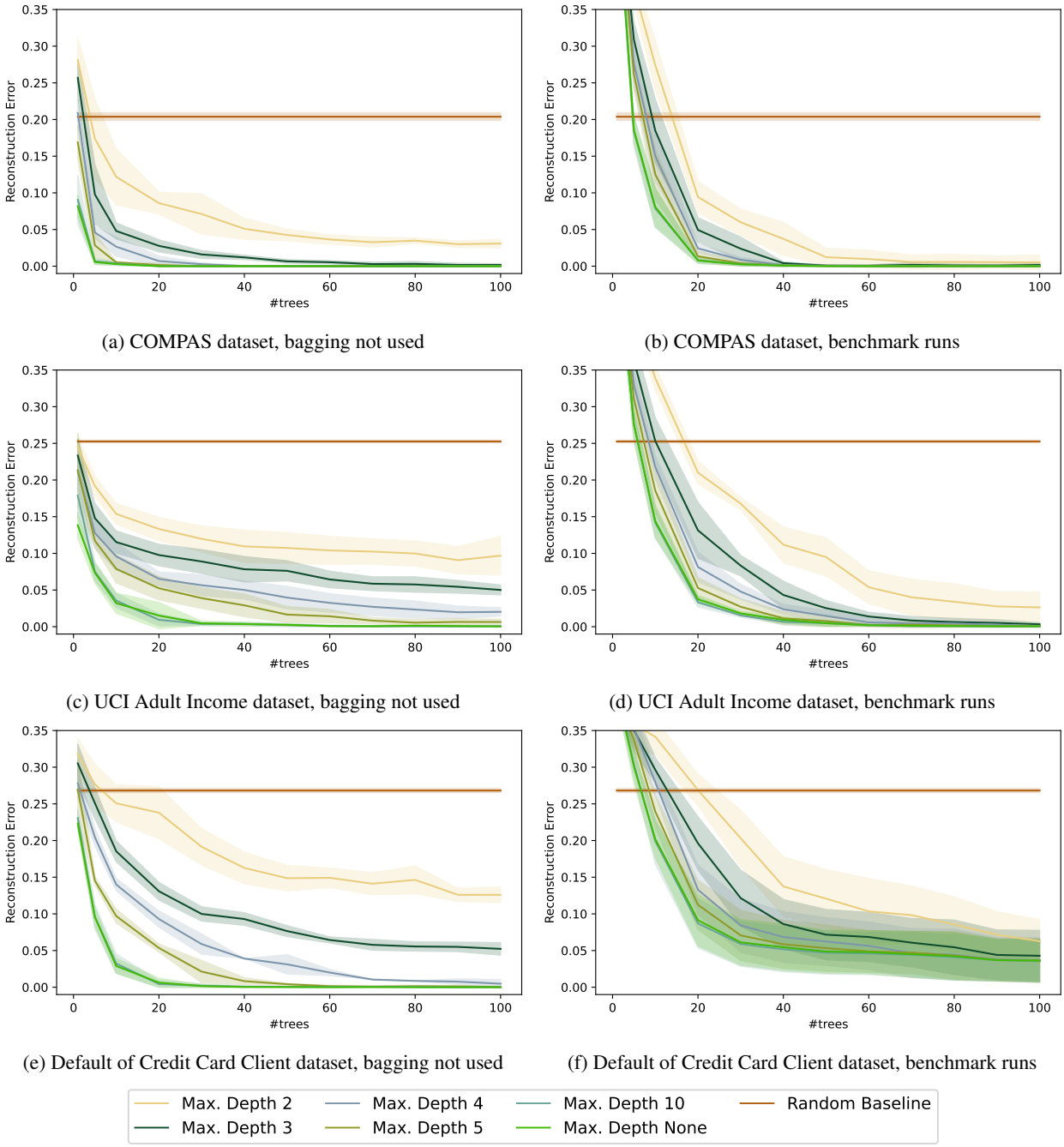
Intuitively, we get the best possible guess for the y_{tvkc} variables by solving the maximum likelihood problem when the training set is given. Subsequently, we get the worst possible solution that is consistent with that guess. It is worth noting that the knowledge of the training set is used in an advantageous way only to obtain the best possible guess for the y_{tvkc} variables.

The results of the benchmark experiments for the three considered datasets are shown in Figures 5b, 5d and 5f. The results without bagging are also repeated in Figures 5a, 5c and 5e (from Figures 2a, 2c and 2e) for reference and easy comparison.

The results show that, if one can correctly guess the y_{tvkc} variables, one can get much closer to recovering 100% of the data, as in the situation without bagging. Accordingly, the key difficulty in recovering the data is guessing which examples were used in each tree. This corroborates the fact that bagging can help prevent data reconstruction. It also answers the question posed at the beginning of this section. Note that bagging was theoretically shown to intrinsically provide some differential privacy guarantees (Liu et al., 2021b), which is consistent with our findings.

It is also interesting to note that the number of trees needed to recover the data without bagging seems to be lower than in the benchmark runs, except for very shallow trees. This makes sense since, without bagging, every tree t provides some information about every example k via the sets ϕ^y_{tk}, while this is not true with bagging.

One can observe another surprising trend when comparing the curves corresponding to shallow trees (*e.g.,* maximum depth of 2). Indeed, without bagging, the reconstruction error decreases until a certain value and remains more or less constant, even when increasing the number of trees further. This does not happen in the benchmark runs, and even with very shallow trees, the reconstruction error (which in this experiment is the worst we can expect) converges close to 0. In fact, a large number of trees trained with bagging seems to provide more information (with the knowledge of the values of y_{tvkc}) than the same number of trees trained without bagging. An explanation for this behavior could lie in the trees' intrinsic diversity and in the fact that each of them contains more information about some training samples, namely those that appeared several times in their training data.



(a) COMPAS dataset, bagging not used

(b) COMPAS dataset, benchmark runs

(c) UCI Adult Income dataset, bagging not used

(d) UCI Adult Income dataset, benchmark runs

(e) Default of Credit Card Client dataset, bagging not used

(f) Default of Credit Card Client dataset, benchmark runs

Figure 5: Comparison of the benchmark results (using bagging, worst possible reconstruction error using our set of constraints if the number of occurrences of each example within each tree are known) with the "no-bagging" ones

F. Additional Experiments on Scalability

This appendix section aims to investigate how our reconstruction attack performs in terms of both reconstruction error and time when the size N of the reconstructed training set varies. We additionally investigate whether or not generalization error affects our reconstruction process. To this end, we re-run our experiments without the use of bagging, for the three datasets and `scikit-learn`'s default configuration (*i.e.,* with a fixed number of trees $|\mathcal{T}| = 100$ and no fixed maximum depth). The setup is as described in Section 7.1, but we vary the size of the sub-sampled training set between 25 and 1,500 examples

and set the reconstruction time limit to 6 hours. The results are provided within Tables 6, 7, and 8 for the three datasets. More precisely, we report for each training set size, the performances (train and test accuracy) of the trained random forests (averaged over the 5 different random samplings), the reconstruction error, as well as the minimum, maximum, average, and standard deviation of the reconstruction times. As can be seen in the tables, for the UCI Adult Income (respectively, the Default of Credit Card Client) dataset, we report no result for $N > 750$ (respectively, for $N > 500$). This is due to a technical limitation in the solver we use in our experiments. More precisely, the Python wrapper of `OR-Tools` is limited in the amount of data it can send to its C++ core. This means that large CP models (above 2GB) can not be solved using this wrapper[‡].

Scalability. We observe in Tables 6, 7, and 8 that the reconstruction error remains very small (0 or very close to 0) for all the considered values of N, i.e., the success of our attack is not affected by the size of the reconstructed training set. However, reconstruction time consistently increases with N. This can be explained by the fact that reconstructing more training examples requires exploring a considerably larger search space. Indeed, increasing N leads to a linear increase in the number of $\{x_{ki}\}_{k\in\{1..N\},\ i\in\{1..M\}}$ and $\{y_{tvk}\}_{t\in\mathcal{T},\ v\in\mathcal{V}_t^L,\ k\in\{1..N\}}$ variables, but to an exponential increase of the number of possible solutions of the constraint programming model (*i.e.,* search space size). Fortunately, the solution process of the CP solver, using domain reduction and other strategies, does not require examining all the solutions. Empirically, the growth of reconstruction time as a function of N is not exponential but polynomial —approximately quadratic on the considered datasets according to a power-law regression. Another interesting side effect is that larger training sets often lead to deeper trees: while small datasets can be separated using shallow trees, larger ones often require more splits to be performed. This leads to an increase in the number of $\{y_{tvk}\}_{t\in\mathcal{T},\ v\in\mathcal{V}_t^L,\ k\in\{1..N\}}$ variables, again increasing the size of the search space. However, this also provides more information regarding the values of the attributes of the reconstructed examples, partly explaining why the reconstruction error remains small, even if the number of possible reconstructions increases significantly.

Generalization Error. Another interesting observation is that the generalization error does not affect our reconstruction approach. This was expected: what matters for reconstruction is the information provided by the forest regarding the training data (*i.e.*, the number of samples passing through each branch fulfilling certain split conditions). Even a badly performing random forest can lead to accurate reconstructions if it encodes enough diverse information regarding its training data within the trees (regardless of unseen test data). This was already visible in all our experiments, where no correlation could be drawn between a random forest's generalization error and the success of our reconstruction attack. More precisely, we investigated for a possible correlation between the reconstruction error and the generalization error (or train error or test error), but none of these analyses led to any visible trend. Finally, as it stands, having good or bad generalization capabilities does not appear to be a prerequisite for the success of our reconstruction approach.

G. Additional Experiments on Partial Reconstruction

In this appendix section, we perform complementary experiments on partial dataset reconstruction. More precisely, we consider the scenario where part of the training set attributes are known (for each training example). This scenario corresponds to the case where some of the attributes are publicly known, and the adversary's objective is only to retrieve the unknown (private) ones. As discussed in Section 3, this setup corresponds to most of the reconstruction attacks found in the literature, where many works only attempt to reconstruct a single private attribute with knowledge of all the remaining ones (Dinur & Nissim, 2003; Dwork et al., 2017).

For each of the three datasets considered in our experiments (introduced in Section 7.1), we vary the number of known attributes between 0 and $M-1$. The former case corresponds to the setup studied in Section 7 (in which the adversary reconstructs the whole dataset), while in the latter case, only one attribute is unknown. Between these two situations, our objective is also to characterize whether the knowledge of a number of attributes helps reconstruct the others, and to what extent. Note that because binary attributes that are a one-hot encoding of the same original feature are not independent from each other, knowledge of one of them can fix the value of the others, which could bias the reconstruction results. For this reason, we consider each set of binary attributes that one-hot encode the same original feature as a single one. Then, the COMPAS dataset has 7 such original features, the UCI Adult Income dataset has 14, and the Default of Credit Card Client dataset has 16. For each $M' \in \{0..M - \sum_{w\in vects}(|w|-1)\}$, we randomly pick M' original attributes (*i.e.,* either a binary attribute or a group of binary attributes one-hot encoding the same feature) that we assume are known. For such known

[‡]This limitation is discussed on the solver's repository: https://github.com/google/or-tools/issues/3861.

#Examples N	RF Accuracy		Reconstruction error	Reconstruction Time (s)			
	Train	Test	Avg	Avg	Std	Min	Max
25	0.896	0.559	0.0	5.8	1.1	4.1	7.4
50	0.860	0.556	0.0	30.5	5.8	26.6	42.0
100	0.800	0.582	0.0	84.1	12.0	65.8	99.1
200	0.770	0.617	0.0	260.9	28.6	231.2	300.2
300	0.759	0.629	0.0	467.9	80.2	386.1	621.9
400	0.741	0.632	0.0	699.9	52.5	602.3	754.1
500	0.734	0.639	0.0	1071.9	197.0	883.9	1448.7
750	0.725	0.643	0.0	2219.4	428.4	1711.9	2818.9
1000	0.714	0.642	0.0	3678.8	231.0	3300.8	4005.0
1500	0.704	0.650	0.0	7362.2	1097.0	6485.9	9519.5

Table 6: Experiments (non-bagging case) on the reconstruction method's scalability, COMPAS dataset. Applying a simple power law regression, we observe that running times are in $\Theta(N^{1.7})$.

#Examples N	RF Accuracy		Reconstruction error	Reconstruction Time (s)			
	Train	Test	Avg	Avg	Std	Min	Max
25	0.952	0.709	0.0	9.1	2.9	6.2	14.6
50	0.964	0.748	0.0	37.0	4.4	29.2	42.8
100	0.964	0.770	0.0	188.2	58.0	117.4	278.3
200	0.949	0.767	0.0	631.7	110.8	513.2	838.6
300	0.935	0.771	0.1	4860.4	2457.7	1430.2	7695.8
400	0.925	0.778	0.1	6119.3	2365.6	2304.4	8054.8
500	0.913	0.779	0.1	6523.6	1558.4	4695.4	8429.2
750	0.905	0.780	0.0	13911.5	5146.7	8764.9	19058.2

Table 7: Experiments (non-bagging case) on the reconstruction method's scalability, UCI Adult Income dataset. Applying a simple power law regression, we observe that running times are in $\Theta(N^{1.4})$.

#Examples N	RF Accuracy		Reconstruction error	Reconstruction Time (s)			
	Train	Test	Avg	Avg	Std	Min	Max
25	0.968	0.733	0.0	6.7	1.5	4.0	8.1
50	0.976	0.723	0.0	42.1	3.5	37.8	47.3
100	0.972	0.740	0.0	148.3	16.6	127.6	169.4
200	0.965	0.751	0.0	905.6	528.1	584.1	1958.1
300	0.957	0.763	0.0	2369.8	1167.5	1316.7	4524.4
400	0.952	0.760	0.0	2733.2	581.5	2065.6	3706.5
500	0.947	0.765	0.0	6119.9	1880.1	3902.8	8671.4

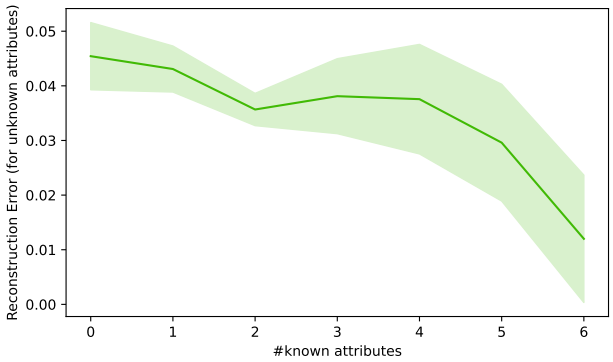
Table 8: Experiments (non-bagging case) on the reconstruction method's scalability, Default of Credit Card Client dataset. Applying a simple power law regression, we observe that running times are in $\Theta(N^{2.4})$.

attributes, their values for all the training set examples are fixed in the CP model introduced in Section 6. In other words, for each known attribute i, we assign the corresponding variables x_{ki} ($\forall k \in \{1..N\}$) to their true value. The solver's task is then to find the value of the other attributes only.

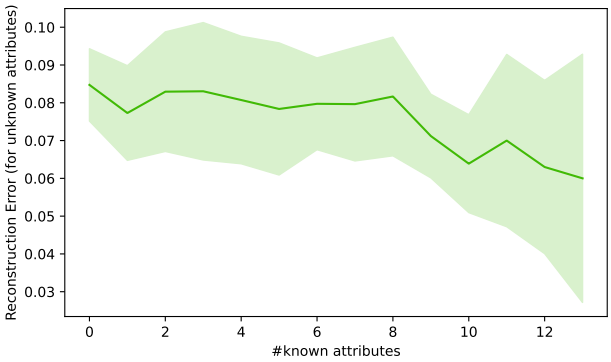
To evaluate the proposed reconstruction, we first perform the examples' matching (with the actual training set) as described in Section 7.1, using all the attributes. Then, the resulting reconstruction error is measured only on the unknown attributes. Note that performing the matching only using the unknown attributes would (artificially) result in lower reconstruction error rates, but would not make sense, as it would only evaluate whether the correct values for the unknown features are found (and not whether they are assigned to the correct example as indicated by the known attributes). For these experiments, we focus on `scikit-learn`'s default configuration (*i.e.,* $|\mathcal{T}| = 100$ trees and no maximum depth constraint). Moreover, we restrict our attention to the general case where bagging is used, as the (simpler) case without bagging is already successfully handled even without knowledge of any attribute. The experimental parameters are as described in Section 7.1, and in particular, each run is averaged over five different random seeds. Finally, as already observed in Section 7, in a few experiments, the solver does not find any feasible reconstruction within the given time frame. Thus, we removed the experiments for which less than three runs were completed. This occurs in two cases, *i.e.,* on the Default of Credit Card Client dataset, when the number of fixed attributes is at most 2.

The reconstruction error (measured on the unknown attributes as aforementioned) is reported in Figure 6 for all three datasets. The results consistently show that knowledge of some attributes helps reconstructing the others. Moreover, the more attributes are known, the lower the error on the remaining (unknown) ones. This suggests that considering the scenarios commonly used in the reconstruction literature only improves the results of our attack, as it successfully leverages knowledge of part of the dataset attributes.

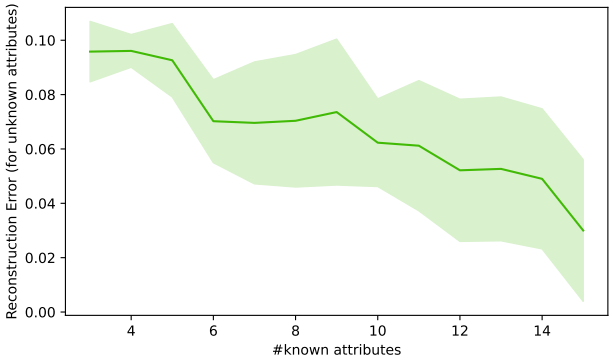
It is worth noting that we also performed partial reconstruction experiments (not reported here) in which part of the training set examples (rather than attributes) are known by the attacker. Interestingly, we observed a different trend as knowledge of some examples did not really improve the reconstruction error for the others. A possible explanation for that (related to our findings of Section E) lies in the Differential Privacy (DP) protection intrinsically offered by bagging (Liu et al., 2021b). Indeed, DP ensures that the trained forest does not depend too strongly on any single example, hence protecting each individual row within the training set. On the contrary, it does not directly protect the training set columns.



(a) COMPAS dataset



(b) UCI Adult Income dataset



(c) Default of Credit Card Client dataset

Figure 6: Results of reconstruction experiments with knowledge of some of the attributes. We report the reconstruction error (for the unknown attributes) as a function of the number of known attributes in the forest's training set. For these experiments, all forests are learnt using `scikit-learn`'s default configuration (*i.e.*, $|\mathcal{T}| = 100$ and no maximum depth constraint). Reconstruction errors are averaged over 5 different random seeds and we also report the standard deviation.