Generative Forests

Richard Nock
Google Research
richardnock@google.com

Mathieu Guillame-Bert
Google
gbm@google.com

Abstract

We focus on generative AI for a type of data that still represent one of the most prevalent form of data: tabular data. Our paper introduces two key contributions: a new powerful class of forest-based models fit for such tasks and a simple training algorithm with strong convergence guarantees in a boosting model that parallels that of the original weak / strong supervised learning setting. This algorithm can be implemented by a few tweaks to the most popular induction scheme for decision tree induction (*i.e. supervised learning*) with two classes. Experiments on the quality of generated data display substantial improvements compared to the state of the art. The losses our algorithm minimize and the structure of our models make them practical for related tasks that require fast estimation of a density given a generative model and an observation (even partially specified): such tasks include missing data imputation and density estimation. Additional experiments on these tasks reveal that our models can be notably good contenders to diverse state of the art methods, relying on models as diverse as (or mixing elements of) trees, neural nets, kernels or graphical models.

1 Introduction

There is a substantial resurgence of interest in the ML community around tabular data, not just because it is still one of the most prominent kind of data available [6]: it is recurrently a place of sometimes heated debates on what are the best model architectures to solve related problems. For example, even for well-posed problems with decades of theoretical formalization like supervised learning [39], after more than a decade of deep learning disruption [22], there is still much ink spilled in the debate decision forests vs neural nets [28]. Generative AI[*] makes no exception. Where the consensus has long been established on the best categories of architectures for data like image and text (neural nets), tabular data still flourishes with a variety of model architectures building up – or mixing – elements from knowledge representation, logics, kernel methods, graph theory, and of course neural nets [4, 9, 12, 31, 34, 41, 43, 44] (see Section 2). Because of the remarkable nature of tabular data where a single variable can bring considerable information about a target to model, each of these classes can be a relevant choice at least in *some* cases (such is is the conclusion of [28] in the context of supervised learning). A key differentiator between model classes is then training and the formal guarantees it can provide [31, 44].
In this paper, we introduce new generative models based on sets of trees that we denote as *generative forests* (GF), along with a training algorithm which has two remarkable features: it is extremely simple and brings strong convergence guarantees in a weak / strong learning model that parallels that of the original boosting model of Valiant's PAC learning [18]. These guarantees improve upon the best state of the art guarantees [44, 31]. Our training algorithm, GF.BOOST, supports training from data with missing values and is simple enough to be implementable by a few tweaks on the popular induction scheme for *decision tree induction* with two classes, which is supported by a huge number

[*]We use this now common parlance expression on purpose, to avoid confusion with the other "generative" problem that consists in modelling densities [7, 37].

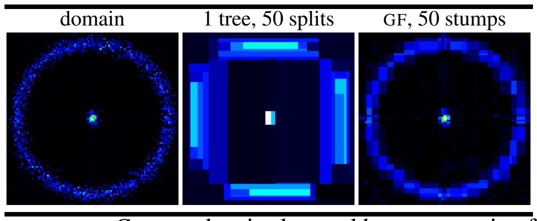


Table 1: *Left*: domain `circgauss`. *Center*: density learned by a generative forest (GF) consisting of a single tree, boosted for a small number (50) of iterations. *Right*: density learned by a GF consisting of 50 boosted tree stumps (*Center* and *Right* learned using GF.BOOST). In a domain $\mathcal{X}$ with dimension d, a single tree with n splits can only partition the domain in $n + 1$ parts. On the other hand, a set of n stumps in a GF can boost this number to $n^{\tilde{\Omega}(d)}$ (the tilda omits $\log$ dependencies), which explains why the right density appears so much better than the central one, even when each tree is just a stump.

of repositories / ML software implementing algorithms like CART or C4.5 [1, 35, 3, 36, 45]. From the model standpoint, generative forests bring a sizeable combinatorial advantage over its closest competitors, generative trees [31] (see Table 1 for an example) and adversarial random forests [44]. Experiments on a variety of simulated or readily available domains display that our models can substantially improve upon state of the art, with models of ours as simple as a set of stumps potentially competing with other approaches building much more complex models.
The models we build have an additional benefit: it is computationally easy to compute the full density given an observation, *even partially specified*; hence, our generative models can also be used for side tasks like missing data imputation or density estimation. Additional experiments clearly display that our approach can be a good contender to the state of the art. To save space and preserve readability, all proofs and additional experiments and results are given in an Appendix.

2 Related work

It would not do justice to the large amount of work in the field of "Generative AI" for tabular data to just sample a few of them, so we devote a part of the Appendix to an extensive review of the state of the art. Let us just mention that, unlike for unstructured data like images, there is a huge variety of model types, based on trees [7, 31, 44], neural networks [12, 14, 20, 47], probabilistic circuits [5, 43, 38], kernel methods [4, 34], graphical models [41] (among others: note that some are in fact hybrid models). The closest approaches to ours are [44] and [31], because the models include trees with a stochastic activation of edges to pick leaves, and a leaf-dependent data generation process. While [31] learn a single tree, [44] use a way to generate data from a set of trees – called an adversarial random forest – which is simple: sample a tree, and then sample an observation from the tree. The model is thus simple but not economical: each observation is generated by a single tree only, each of them thus having to represent a good sampler in its own. In our case, generating one observation makes use of *all* trees (Figure 1, see also Section 4). We also note that the theory part of [44] is limited because it resorts to statistical consistency (infinite sample) and Lipschitz continuity of the target density, with second derivative continuous, square integrable and monotonic. Similarly, [31] resort to a wide set of proper losses, but with a symmetry constraint impeding in a generative context. As we shall see, our theoretical framework does not suffer from such impediments.

3 Basic definitions

Perhaps surprisingly at first glance, this Section introduces generative and *supervised* loss functions. Indeed, our algorithm, which trains a data generator and whose overall convergence shall be shown on a generative loss, turns out to locally optimize a *supervised* loss. For the interested reader, the key link, which is of independent interest since it links in our context losses for supervised and generative learning, is established in Lemma A (Appendix).
$\forall k \in \mathbb{N}_{>0}$, let $[k] \doteq \{1, 2, ..., k\}$. Our notations follow [37]. Let $\mathfrak{B} \doteq (\pi, \mathrm{A}, \mathrm{B})$ denote a *binary task*, where A, B (and any other measure defined hereafter) are probability measures with the same support, also called *domain*, $\mathcal{X}$, and $\pi \in [0, 1]$ is a *prior*. $\mathrm{M} \doteq \pi \cdot \mathrm{A} + (1 - \pi) \cdot \mathrm{B}$ is the corresponding mixture measure. For the sake of simplicity, we assume $\mathcal{X}$ bounded hereafter, and note that tricks can be used to remove this assumption [31, Remark 3.3]. In tabular data, each of the $\dim(\mathcal{X})$ features

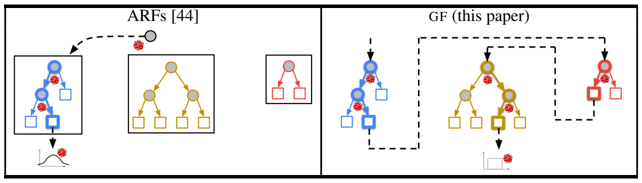


Figure 1: Sketch of comparison of two approaches to generate one observation, using Adversarial Random Forests [44] (left) and using generative forests, GF (right, this paper). In the case of Adversarial Random Forests, a tree is sampled uniformly at random, then a leaf is sampled in the tree and finally an observation is sampled according to the distribution "attached" to the leaf. Hence, only one tree is used to generate an observation. In our case, we leverage the combinatorial power of the trees in the forest: *all trees* are used to generate one observation, as each is contributing to one leaf. Figure 3 provides more details on generation using GF.

can be of various *types*, including categorical, numerical, etc., and associated to a natural measure (counting, Lebesgue, etc.) so we naturally associate $\mathcal{X}$ to the product measure, which can thus be of mixed type. We also write $\mathcal{X} \doteq \times_{i=1}^{d} \mathcal{X}_i$, where $\mathcal{X}_i$ is the set of values that can take on variable i. Several essential measures will be used in this paper, including U, the uniform measure, G, the measure associated to a generator that we learn, R, the *empirical* measure corresponding to a training sample of observations. Like in [31], we do not investigate generalisation properties.
Loss functions There is a natural problem associated to binary task $\mathfrak{B}$, that of estimating the probability that an arbitrary observation $\boldsymbol{x} \in \mathcal{X}$ was sampled from A – call such *positive* – or B – call these *negative* –. To learn a *supervised* model $\mathcal{X} \rightarrow [0,1]$ for such a *class probability estimation* (CPE) problem, one usually has access to a set of examples where each is a couple (observation, class), the class being in set $\mathcal{Y} \doteq \{-1, 1\}$ (={negative, positive}). Examples are drawn i.i.d. according to $\mathfrak{B}$. Learning a model is done by minimizing a loss function: when it comes to CPE, any such CPE loss [2] is some $\ell : \mathcal{Y} \times [0,1] \rightarrow \mathbb{R}$ whose expression can be split according to *partial* losses ℓ_1, ℓ_{-1}, $\ell(y, u) \doteq [\![y = 1]\!] \cdot \ell_1(u) + [\![y = -1]\!] \cdot \ell_{-1}(u)$. Its (pointwise) *Bayes risk* function is the best achievable loss when labels are drawn with a particular positive base-rate,

$$\underline{L}(p) \doteq \inf_u \mathsf{E}_{\mathsf{Y} \sim \mathrm{B}(p)} \ell(\mathsf{Y}, u), \tag{1}$$

where $\mathrm{B} \doteq$ Bernoulli random variable for class 1. A fundamental property for a CPE loss is ***properness***, encouraging to guess ground truth: ℓ is proper iff $\underline{L}(p) = \mathsf{E}_{\mathsf{Y} \sim \mathrm{B}(p)} \ell(\mathsf{Y}, p), \forall p \in [0,1]$, and *strictly* proper if $\underline{L}(p) < \mathsf{E}_{\mathsf{Y} \sim \mathrm{B}(p)} \ell(\mathsf{Y}, u), \forall u \neq p$. Strict properness implies strict concavity of Bayes risk. For example, the square loss has $\ell_1^{\mathrm{SQ}}(u) \doteq (1-u)^2$, $\ell_{-1}^{\mathrm{SQ}}(u) \doteq u^2$, and, being strictly proper, Bayes risk $\underline{L}^{\mathrm{SQ}}(u) \doteq u(1-u)$. Popular strictly proper ML include the log and Matusita's losses. All these losses are ***symmetric*** since $\ell_1^{\mathrm{SQ}}(u) = \ell_{-1}^{\mathrm{SQ}}(1-u), \forall u \in (0,1)$ and ***differentiable*** because both partial losses are differentiable.
In addition to CPE losses, we introduce a set of losses relevant to generative approaches, that are popular in density ratio estimation [29, 40]. For any differentiable and convex $F : \mathbb{R} \rightarrow \mathbb{R}$, the Bregman divergence with generator F is $D_F(z \| z') \doteq F(z) - F(z') - (z - z')F'(z')$. Given function $g : \mathbb{R} \rightarrow \mathbb{R}$, the generalized perspective transform of F given g is $\check{F}(z) \doteq g(z) \cdot F(z/g(z))$, g being implicit in notation $\check{F}$ [26, 27, 32]. The *Likelihood ratio risk* of B with respect to A for loss ℓ is

$$\mathbb{D}_\ell(\mathrm{A}, \mathrm{B}) \quad \doteq \quad \pi \cdot \mathbb{E}_{\mathrm{U}} \left[D_{(\widecheck{-\underline{L}})} \left(\frac{\mathrm{dA}}{\mathrm{dU}} \middle\| \frac{\mathrm{dB}}{\mathrm{dU}} \right) \right], \tag{2}$$

with $g(z) \doteq z + (1-\pi)/\pi$ in the generalized perspective transform. The prior multiplication is for technical convenience. $\mathbb{D}_\ell$ is non-negative; strict properness is necessary for a key property of $\mathbb{D}_\ell$: $\mathbb{D}_\ell = 0$ iff $\mathrm{A} = \mathrm{B}$ almost everywhere [31].

4 Generative forests: models and data generation

Architecture We first introduce the basic building block of our models, *trees*.
Definition 4.1. *A **tree** Υ is a binary directed tree whose internal nodes are labeled with an observation variable and arcs are* consistently *labeled with subsets of their tail node's variable domain.*

Consistency is an important generative notion; informally, it postulates that the arcs' labels define a partition of the measure's support. To make this notion formal, we proceed need a key definition.

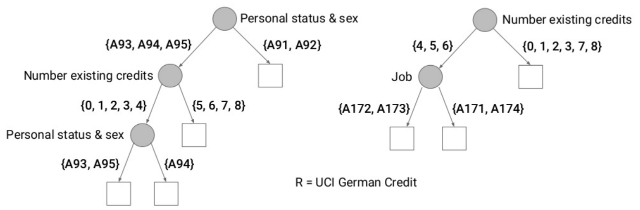


Figure 2: A GF ($T = 2$) associated to UCI German Credit. Constraint **(C)** (see text) implies that the domain of "Number existing credits" is $\{0, 1, ..., 8\}$, that of "Job" is $\{$A171, A172, A173, A174$\}$, etc. .

Algorithm 1 INIT($\{\Upsilon_t\}_{t=1}^T$)

```
Input: Trees $\{\Upsilon_t\}_{t=1}^T$ of a GF;
Step 1 : for $t \in [T]$
        $\Upsilon_t.\nu^\star \leftarrow \mathrm{root}_t$;
        $\Upsilon_t.\mathtt{done} \leftarrow [\![\Upsilon_t.\nu^\star \in \Lambda(\Upsilon_t)]\!]$;
```

Algorithm 2 STARUPDATE($\Upsilon, \mathcal{C}, \mathrm{R}$)

```
Input: Tree $\Upsilon$, subset $\mathcal{C} \subseteq \mathcal{X}$, measure $\mathrm{R}$;
Step 1 : if $\mathrm{B}(p_{\mathrm{R}}[\mathcal{X}_{\Upsilon.\nu_{\mathtt{t}}^\star} \cap \mathcal{C}|\mathcal{C}]) = 1$ then
            $\mathcal{C} \leftarrow \mathcal{X}_{\Upsilon.\nu_{\mathtt{t}}^\star} \cap \mathcal{C}$; $\Upsilon.\nu^\star \leftarrow \Upsilon.\nu_{\mathtt{t}}^\star$;
         else
            $\mathcal{C} \leftarrow \mathcal{X}_{\Upsilon.\nu_{\mathtt{f}}^\star} \cap \mathcal{C}$; $\Upsilon.\nu^\star \leftarrow \Upsilon.\nu_{\mathtt{f}}^\star$;
Step 2 : if $\Upsilon.\nu^\star \in \Upsilon.\Lambda$ then $\Upsilon.\mathtt{done} \leftarrow \mathtt{true}$;
```

For any node $\nu \in \mathcal{N}(\Upsilon)$ (the whole set of nodes of Υ, including leaves), we denote $\mathcal{X}_\nu \subseteq \mathcal{X}$ the *support* of the node. The root has $\mathcal{X}_\nu = \mathcal{X}$. To get $\mathcal{X}_\nu$ for any other node, we initialize it to $\mathcal{X}$ and then descend the tree from the root, progressively updating $\mathcal{X}_\nu$ by intersecting an arc's observation variable's domain in $\mathcal{X}_\nu$ with the sub-domain labelling the arc until we reach ν. Then, a labeling of arcs in a tree is *consistent* iff it complies with one constraint **(C)**:

(C) for each internal node ν and its left and right children $\nu_{\mathtt{f}}, \nu_{\mathtt{t}}$ (respectively), $\mathcal{X}_\nu = \mathcal{X}_{\nu_{\mathtt{f}}} \cup \mathcal{X}_{\nu_{\mathtt{t}}}$ and the measure of $\mathcal{X}_{\nu_{\mathtt{f}}} \cap \mathcal{X}_{\nu_{\mathtt{t}}}$ with respect to G is zero.

For example, the first split at the root of a tree is such that the union of the domains at the two arcs equals the domain of the feature labeling the split (see Figure 2). We define our generative models.

Definition 4.2. *A* ***generative forest*** (GF)*,* G*, is a set of trees,* $\{\Upsilon_t\}_{t=1}^T$*, associated to measure* R *(implicit in notation).*

Figure 2 shows an example of GF. Following the consistency requirement, any single tree defines a recursive partition of $\mathcal{X}$ according to the splits induced by the inner nodes. Such is *also* the case for a set of trees, where *intersections* of the supports of tuples of leaves (1 for each tree) define the subsets:

$$\mathcal{P}(\mathrm{G}) \doteq \left\{\cap_{i=1}^T \mathcal{X}_{\lambda_i} \text{ s.t. } \lambda_i \in \Lambda(\Upsilon_i), \forall i\right\} \quad (3)$$

($\Lambda(\Upsilon) \subseteq \mathcal{N}(\Upsilon)$ is the set of leaves of Υ). Notably, we can construct the elements of $\mathcal{P}(\mathrm{G})$ using the same algorithm that would compute it for 1 tree. First, considering a first tree Υ_1, we compute the support of a leaf, say $\mathcal{X}_{\lambda_1}$, using the algorithm described for the consistency property above. Then, we start again with a second tree Υ_2 *but* replacing the initial $\mathcal{X}$ by $\mathcal{X}_{\lambda_1}$, yielding $\mathcal{X}_{\lambda_1} \cap \mathcal{X}_{\lambda_2}$. Then we repeat with a third tree Υ_3 replacing $\mathcal{X}$ by $\mathcal{X}_{\lambda_1} \cap \mathcal{X}_{\lambda_2}$, and so on until the last tree is processed. This yields one element of $\mathcal{P}(\mathrm{G})$.

Generating one observation Generating one observation relies on a *stochastic* version of the procedure just described. It ends up in an element of $\mathcal{P}(\mathrm{G})$ of positive measure, from which we sample uniformly one observation, and then repeat the process for another observation. To describe the process at length, we make use of two key routines, INIT and STARUPDATE, see Algorithms 1 and 2. INIT initializes "special" nodes in each tree, that are called *star nodes*, to the root of each tree (notation for a variable v relative to a tree Υ is $\Upsilon.v$). Stochastic activation, performed in STARUPDATE, progressively makes star nodes descend in trees. When all star nodes have reached a leaf in their respective tree, an observation is sampled from the intersection of the leaves' domains (which is an element of $\mathcal{P}(\mathrm{G})$). A Boolean flag, `done` takes value `true` when the star node is in the leaf set of the tree, indicating no more STARUPDATE calls for the tree ($[\![.]\!]$ is Iverson's bracket, [21]).
STARUPDATE is called with a tree of the GF for which `done` is false, a subset $\mathcal{C}$ of the whole domain and measure R. The first call of this procedure is done with $\mathcal{C} = \mathcal{X}$. When all trees are marked `done`, $\mathcal{C}$ has been "reduced" to some $\mathcal{C}_{\mathrm{s}} \in \mathcal{P}(\mathrm{G})$, where the index reminds that this is the last $\mathcal{C}$ we obtain, from which we **s**ample an observation, uniformly at random in $\mathcal{C}_{\mathrm{s}}$. Step 1 in STARUPDATE is

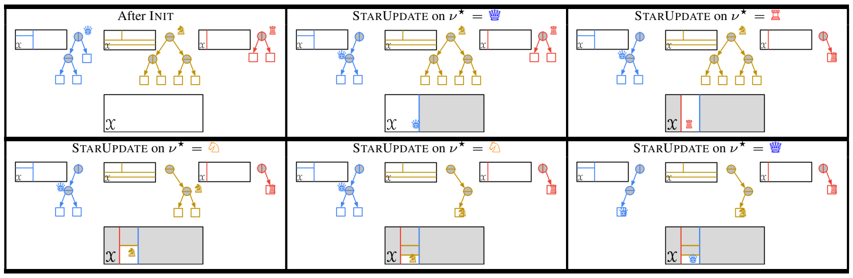

After INIT
STARUPDATE on $\nu^\star$ = ♛
STARUPDATE on $\nu^\star$ = ♜
STARUPDATE on $\nu^\star$ = ♘
STARUPDATE on $\nu^\star$ = ♘
STARUPDATE on $\nu^\star$ = ♛
$\mathcal{X}$

Figure 3: From left to right and top to bottom: updates of the argument $\mathcal{C}$ of STARUPDATE through a sequence of run of STARUPDATE in a generative forest consisting of three trees (the partition of the domain induced by each tree is also depicted, alongside the nature of splits, vertical or horizontal, at each internal node) whose star nodes are indicated with chess pieces (♛, ♜, ♘). In each picture, $\mathcal{C}$ is represented at the bottom of the picture (hence, $\mathcal{C} = \mathcal{X}$ after INIT). In the bottom-right picture, all star nodes are leaves and thus $\mathcal{C}_s = \mathcal{C}$ displays the portion of the domain in which an observation is sampled. Remark that the last star node update produced no change in $\mathcal{C}$.

fundamental: it relies on tossing an unfair coin (a Bernoulli event noted $\mathrm{B}(p)$), where the head probability p is just the mass of R in $\mathcal{X}_{\Upsilon.\nu_t^\star} \cap \mathcal{C}$ *relative to* $\mathcal{C}$. Hence, if $\mathcal{X}_{\Upsilon.\nu_t^\star} \cap \mathcal{C} = \mathcal{C}$, $p = 1$. There is a simple but important invariant (proof omitted).

Lemma 4.3. *In* STARUPDATE, *it always holds that the input* $\mathcal{C}$ *satisfies* $\mathcal{C} \subseteq \mathcal{X}_{\Upsilon.\nu^\star}$.

We have made no comment about the *sequence* of tree choices over which STARUPDATE is called. Let us call *admissible* such a sequence that ends up with *some* $\mathcal{C}_s \subseteq \mathcal{X}$. T being the number of trees (see INIT), for any sequence $\boldsymbol{v} \in [T]^{D(\mathcal{C}_s)}$, where $D(\mathcal{C}_s)$ is the sum of the depths of all the star *leaves* whose support intersection is $\mathcal{C}_s$, we say that $\boldsymbol{v}$ is *admissible for* $\mathcal{C}_s$ if there exits a sequence of branchings in Step 1 of STARUPDATE, whose corresponding sequence of trees follows the indexes in $\boldsymbol{v}$, such that at the end of the sequence all trees are marked `done` and the last $\mathcal{C} = \mathcal{C}_s$. Crucially, the probability to end up in $\mathcal{C}_s$ using STARUPDATE, given any of its admissible sequences, is the *same* and equals its mass with respect to R.

Lemma 4.4. *For any* $\mathcal{C}_s \in \mathcal{P}(\mathrm{G})$ *and admissible sequence* $\boldsymbol{v} \in [T]^{D(\mathcal{C}_s)}$ *for* $\mathcal{C}_s$, $p_{\mathrm{G}}[\mathcal{C}_s|\boldsymbol{v}] = p_{\mathrm{R}}[\mathcal{C}_s]$.

The Lemma is simple to prove but fundamental in our context as the way one computes the sequence – and thus the way one picks the trees – does not bias generation: the sequence of tree choices could thus be iterative, randomized, concurrent (*e.g.* if trees were distributed), etc., this would not change generation's properties from the standpoint of Lemma 4.4. We defer to Appendix (Section II) three examples of sequence choice. Figure 3 illustrates a sequence and the resulting $\mathcal{C}_s$.

Missing data imputation and density estimation with GFs A generative forest is not just a generator: it models a density and the exact value of this density at any observation is easily available. Figure 4 (top row) shows how to compute it. Note that if carried out in parallel, the complexity to get this density is cheap, of order $O(\max_t \mathrm{depth}(\Upsilon_t))$. If one wants to prevent predicting zero density, one can stop the descent if the next step creates a support with zero empirical measure. A GF thus also models an easily tractable density, but this fact alone is not enough to make it a *good* density estimator. To get there, one has to factor the loss minimized during training. In our case, as we shall see in the following Section, we train a GF by minimizing an information-theoretic loss directly formulated on this density (2). So, using a GF also for density estimation can be a reasonable additional benefit of training GFs.
The process described above that finds leaves reached by an observation can be extended to missing values in the observation using standard procedures for classification using decision trees. We obtain a simple procedure to carry out *missing data imputation* instead of density estimation: once a tuple of leaves is reached, one uniformly samples the missing features in the corresponding support. This is fast, but at the expense of a bit more computations, we can have a much better procedure, as explained in Figure 4 (bottom row). In a first step, we compute the density of each support subset corresponding to *all* (not just 1 as for density estimation) tuples of leaves reached in each tree. This provides us with the *full density* over the missing values *given* (i) a partial assignment of the tabular domain's variables and (ii) the GF. We then keep the elements corresponding to the maximal density value and finally simultaneously sample all missing features uniformly in the corresponding domain. Overall, the whole procedure is $O(d \cdot (\sum_t \mathrm{depth}(\Upsilon_t))^2)$.

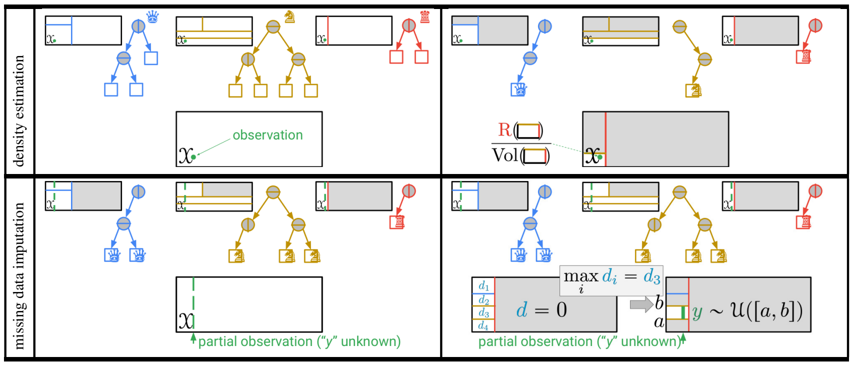


Figure 4: (**Top row**) Density estimation using a GF, on an observation indicated by ● (*Left*). In each tree, the leaf reached by the observation is found and the intersection of all leaves' supports is computed. The estimated density at ● is computed as the empirical measure in this intersection over its volume (*Right*). (**Bottom row**) Missing data imputation using the same GF, and an observation with one missing value (if $\mathcal{X} \subset \mathbb{R}^2$, then y is missing). We first proceed like in density estimation, finding in each tree *all* leaves *potentially* reached by the observation if y were known (*Left*); then, we compute the density in *each* non-empty intersection of all leaves' supports; among the corresponding elements with maximal density, we get the missing value(s) by uniform sampling (*Right*).

Algorithm 3 GF.BOOST(R, J, T)

```
Input: measure R, #iters $J$, #trees $T$;
Output: trees $\{\Upsilon_t\}_{t=1}^T$ of GF G;
Step 1 : $\mathcal{T} \leftarrow \{\Upsilon_t = (\mathrm{root})\}_{t=1}^T$;
Step 2 : for $j = 1$ to $J$
    Step 2.1 : $\Upsilon_* \leftarrow$ tree ($\mathcal{T}$);
    Step 2.2 : $\lambda_* \leftarrow$ leaf ($\Upsilon_*$);
    Step 2.3 : p ← splitPred ($\lambda_*, \mathcal{T}$, R);
    Step 2.4 : split($\Upsilon_*, \lambda_*$, p);
return $\mathcal{T}$;
```

5 Learning generative forests using supervised boosting

From the GAN framework to a fully supervised training of generative models It can appear quite unusual to train generative models using a supervised learning framework, so before introducing our algorithm, we provide details on its filiation in the generative world, starting from the popular GAN training framework [12]. In this framework, one trains a generative model against a discriminator model which has no purpose other than to parameterize the generative loss optimized. As shown in [33], there is a generally inevitable slack between the generator and the discriminator losses with neural networks, which translates into uncertainty for training. [31] show that the slack disappears for calibrated models, a property satisfied by their generative trees (and also by our generative forests). Moreover, [31] also show that the GAN training can be simplified and made much more efficient for generative trees by having the discriminator (a decision tree) copy the tree structure of the generator, a setting defined as *copycat*. Hence, one gets reduced uncertainty and more efficient training. Training still involves two models but it becomes arguably very close to the celebrated boosting framework for top-down induction of decision trees [19], up to the crucial detail that the generator turns out to implements boosting's leveraged "hard" distribution. This training gives guarantees on the likelihood ratio risk we define in (2). Our paper closes the gap with boosting: copycat training can be equivalently simplified to training a *single generative model* in a *supervised* (2 classes) framework. The two classes involved are the observed data and the uniform distribution. Using the uniform distribution is allowed by the assumption that the domain is closed, which is reasonable for tabular data but can also be alleviated by reparameterization [31, Remark 3.3]. The (supervised) loss involved for training reduces to the popular concave "entropic"-style losses used in CART, C4.5 and in popular packages like `scikit-learn`. And of course, minimizing such losses *still* provides guarantee on the generative loss (2). While it is out of scope to show how copycat training does ultimately simplify, we provide all components of the "end product": algorithm, loss optimized and the link between the minimization of the supervised and generative losses via boosting.

The algorithm To learn a GF, we just have to learn its set of trees. Our training algorithm, GF.BOOST (Algorithm 3), performs a greedy top-down induction. In Step 1, we initialize the set of T trees to T roots. Steps 2.1 and 2.2 choose a tree (Υ_*) and a leaf to split (λ_*) in the tree. In our implementation, we pick the leaf among all trees which is the heaviest with respect to R. Hence, we merge Steps 2.1 and 2.2. Picking the heaviest leaf is standard to grant boosting in decision tree induction [19]; in our case, there is a second benefit: we tend to learn size-balanced models. For example, during the first $J = T$ iterations, each of the T roots gets one split because each root has a larger mass (1) than any leaf in a tree with depth > 0. Step 2.4 splits Υ_* by replacing λ_* by a stump whose corresponding splitting predicate, $\mathtt{p}$, is returned in Step 2.3 using a weak splitter oracle called `splitPred`. "weak" refers to boosting's weak/strong learning setting [18] and means that we shall only require lightweight assumptions about this oracle; in decision tree induction, this oracle is the key to boosting from such weak assumptions [19]. This will also be the case for our generative models. We now investigate Step 2.3 and `splitPred`.

The weak splitter oracle `splitPred` In decision tree induction, a splitting predicate is chosen to reduce an expected Bayes risk (1) (*e.g.* that of the log loss [36], square loss [1], etc.). In our case, `splitPred` does about the same *with a catch in the binary task it adresses*, which is[†] $\mathfrak{B}_{\text{GEN}} \doteq (\pi, \mathrm{R}, \mathrm{U})$ (our mixture measure is thus $\mathrm{M} \doteq \pi \cdot \mathrm{R} + (1-\pi) \cdot \mathrm{U}$). The corresponding expected Bayes risk that `splitPred` seeks to minimize is just:

$$\mathbb{L}(\mathcal{T}) \quad \doteq \quad \sum_{\mathcal{C} \in \mathcal{P}(\mathcal{T})} p_{\mathrm{M}}[\mathcal{C}] \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right). \tag{4}$$

The concavity of $\underline{L}$ implies $\mathbb{L}(\mathcal{T}) \leqslant \underline{L}(\pi)$. Notation $\mathcal{P}(.)$ overloads that in (3) by depending explicitly on the set of trees of G instead of G itself. Regarding the minimization of (4), there are three main differences with classical decision tree induction. The first is computational: in the latter case, (4) is optimized over a single tree: there is thus a single element in $\mathcal{P}(\mathcal{T})$ which is affected by the split, $\mathcal{X}_{\lambda_*}$. In our case, multiple elements in $\mathcal{P}(\mathcal{T})$ can be affected by one split, so the optimisation is more computationally demanding but a simple trick allows to keep it largely tractable: we do not need to care about keeping in $\mathcal{P}(\mathcal{T})$ support elements with zero empirical measure since they will generate no data. Keeping only elements with strictly positive empirical measure guarantees a size $|\mathcal{P}(\mathcal{T})|$ never bigger than the number of observations defining R. The second difference plays to our advantage: compared to classical decision tree induction, a single split generally buys a substantially bigger slack in $\mathbb{L}(\mathcal{T})$ in our case. To see this, remark that for the candidate leaf λ_*,

$$\begin{aligned}\sum_{\substack{\mathcal{C} \in \mathcal{P}(\mathcal{T}) \\ \mathcal{C} \subseteq \mathcal{X}_{\lambda_*}}} p_{\mathrm{M}}[\mathcal{C}] \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right) \quad &= \quad p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}] \cdot \sum_{\substack{\mathcal{C} \in \mathcal{P}(\mathcal{T}) \\ \mathcal{C} \subseteq \mathcal{X}_{\lambda_*}}} \frac{p_{\mathrm{M}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}]} \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right) \\ &\leqslant \quad p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}] \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{X}_{\lambda_*}]}{p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}]}\right)\end{aligned}$$

(because $\underline{L}$ is concave and $\sum_{\mathcal{C} \in \mathcal{P}(\mathcal{T}), \mathcal{C} \subseteq \mathcal{X}_{\lambda_*}} p_{\mathrm{R}}[\mathcal{C}] = p_{\mathrm{R}}[\mathcal{X}_{\lambda_*}]$). The top-left term is the contribution of λ_* to $\mathbb{L}(\mathcal{T})$, the bottom-right one its contribution to $\mathbb{L}(\{\Upsilon_*\})$ (the decision tree case), so the slack gives a proxy to our potential advantage after splitting. The third difference with decision tree induction is the possibility to converge much faster to a good solution in our case. Scrutinizing the two terms, we indeed get that in the decision tree case, the split only gets two new terms. In our case however, there can be up to $2 \cdot \mathrm{Card}(\{\mathcal{C} \in \mathcal{P}(\mathcal{T}) : \mathcal{C} \subseteq \mathcal{X}_{\lambda_*}\})$ new elements in $\mathcal{P}(\mathcal{T})$.

Boosting Two questions remain: can we quantify the slack in decrease and of course what quality guarantee does it bring for the *generative* model G whose set of trees is learned by GF.BOOST ? We answer both questions in a single Theorem, which relies on a weak learning assumption that parallels the classical weak learning assumption of boosting:

Definition 5.1. *(**WLA**(γ, κ)) There exists $\gamma > 0, \kappa > 0$ such that at each iteration of* GF.BOOST, *the couple $(\lambda_*, \mathtt{p})$ chosen in Steps 2.2, 2.3 of* GF.BOOST *satisfies the following properties: **(a)** λ_* is not skinny: $p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}] \geqslant 1/\mathrm{Card}(\Lambda(\Upsilon_*))$, **(b)** truth values of $\mathtt{p}$ moderately correlates with $\mathfrak{B}_{\text{GEN}}$ at λ_*: $\left|p_{\mathrm{R}}[\mathtt{p}|\mathcal{X}_{\lambda_*}] - p_{\mathrm{U}}[\mathtt{p}|\mathcal{X}_{\lambda_*}]\right| \geqslant \gamma$, and finally **(c)** there is a minimal proportion of real data at λ_*: $\pi p_{\mathrm{R}}[\mathcal{X}_{\lambda_*}]/p_{\mathrm{M}}[\mathcal{X}_{\lambda_*}] \geqslant \kappa$.*

The convergence proof of [19] reveals that both **(a)** and **(b)** are jointly made at any split, where our **(b)** is equivalent to their *weak hypothesis assumption* where their γ parameter is twice ours. **(c)**

[†]The prior is chosen by the user: without reasons to do otherwise, a balanced approach suggests $\pi = 0.5$.

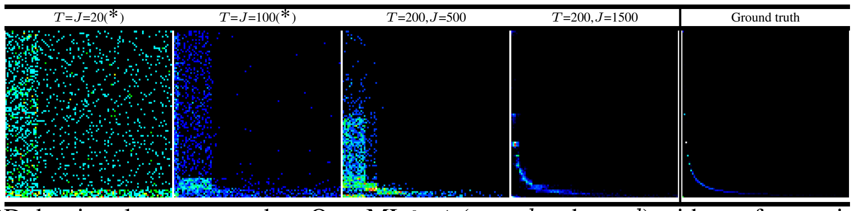


Table 2: 2D density plots generated on OpenML `kc1` ($x = d$ and $y = l$) with GF, for varying number of trees T and number of splits J (columns). "*" = all trees are stumps. The rightmost column recalls the domain ground truth for comparison. Each generated dataset contains $m = 2000$ observations. The two variables (among $d = 22$) were chosen because of their deterministic dependence.

GF $\gg$ KDE	neither	KDE $\gg$ GF
9	5	3

Table 3: DENSITY: comparison between us and KDE (summary), counting the number of domains for which we are statistically better (left), or worse (right). The central column counts the remaining domains, for which no statistical significance ever holds.

postulates that the leaf split has empirical measure at least a fraction of its uniform measure – thus, of its relative volume. **(c)** is important to avoid splitting leaves that would essentially be useless for our tasks: a leaf for which **(c)** is invalid would indeed locally model comparatively tiny density values.

Theorem 5.2. *Suppose the loss ℓ is* ***strictly proper*** *and* ***differentiable****. Let G_0 (= U) denote the initial* GF *with T roots in its trees (Step 1,* GF.BOOST*) and G_J the final* GF*, assuming wlog that the number of boosting iterations J is a multiple of T. Under WLA(γ, κ), we get the following on likelihood ratio risks: $\mathbb{D}_\ell\left(\mathrm{R}, \mathrm{G}_J\right) \leqslant \mathbb{D}_\ell\left(\mathrm{R}, \mathrm{G}_0\right) - \frac{\varkappa\gamma^2\kappa^2}{8} \cdot T \log\left(1 + \frac{J}{T}\right)$, for some constant $\varkappa > 0$.*

It comes from [25, Remark 1] that we can choose $\varkappa = \inf\{\ell'_{-1} - \ell'_1\}$, which is > 0 if ℓ is strictly proper and differentiable. Strict properness is also essential for the loss to guarantee that minimization gets to a good generator (Section 3).

6 Experiments

Our code is provided and commented in Appendix, Section V.2. The main setting of our experiments is realistic data generation ('LIFELIKE'), but we have also tested our method for missing data imputation ('IMPUTE') and density estimation ('DENSITY'): for this reason, we have selected a broad panel of state of the art approaches to test against, relying on models as diverse as (or mixing elements of) trees, neural nets, kernels or graphical models, with MICE [42], adversarial random forests (ARFs [44]), CT-GANs [47], Forest Flows [17], Vine copulas auto-encoders (VCAE, [41]) and Kernel density estimation (KDE, [4, 34]). All algorithms *not* using neural networks were ran on a low-end CPU laptop – this was purposely done for our technique. Neural network-based algorithms are run on a bigger machine (technical specs in Appendix, Section V.2). We carried out experiments on a total of 21 datasets, from UCI [10], Kaggle, OpenML, the Stanford Open Policing Project, or simulated. All are presented in Appendix, Section V.1. We summarize results that are presented *in extenso* in Appendix. Before starting, we complete the 2D heatmap of Table 1 by another one showing our models can also learn deterministic dependences in real-world domains (Table 2). The Appendix also provides an example experiment on interpreting our models for a sensitive domain (in Section V.V.3.1).

Generation capabilities of our models: **LIFELIKE** The objective of the experiment is to evaluate whether a generative model is able to create "realistic" data. The evaluation pipeline is simple: we create for each domain a 5-fold stratified experiment. Evaluation is done via four metrics: a regularized optimal transport ("Sinkhorn", [8]), coverage and density [30] and the F1 score [17]. All metrics are obtained after generating a sample of the same size as the test fold. Sinkhorn evaluates an optimal transport distance between generated and real and F1 estimates the error of a classifier (a 5-nearest neighbor) to distinguish generated vs real (smaller is better for both). Coverage and density are refinements of precision and recall for generative models (the higher, the better). Due to size constraint, we provide results on one set of parameters for "medium-sized" generative forests with $T = 500$ trees, J =2 000 total splits (Table 5), thus learning very small trees with an average 4 splits per tree. In the Appendix, we provide additional results on even smaller models ($T = 200$, $J = 500$)

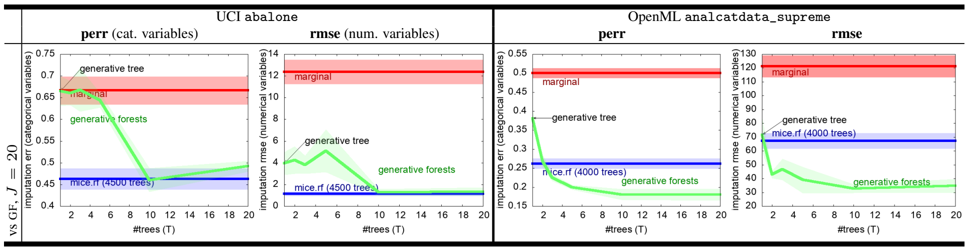


Table 4: IMPUTE: results on two domains (left and right pane). Since both domain contain categorical and numerical variables, we compute for each **perr** (categorical variables) and the **rmse** (numerical variables). In each plot, we display both the corresponding results of GF.BOOST, the results of MICE.RF (indicating the total number of trees used to impute the dataset) and the result of a fast imputation baseline using the `marginals` computed from the training sample. The x axis displays the number of trees T in GF.BOOST and each curve is displayed with its average $\pm$ standard deviation in shaded color. The result for $T = 1$ equivalently indicates the performance of a single generative tree (GT) with J splits [31], shown with an arrow (see text).

and additional results on one metric (Sinkhorn) against contenders with various parameterizations on more domains. In Table 5, contenders are parameterized as follows: ARFs learn sets of 200 trees. Since each tree has to be separately a good generative model, we end up with big models in general. CT-GANs are trained for 1 000 epochs. Forest Flows and VCAEs are run with otherwise default parameters. Table 5 contains just a subset of all our domains, furthermore not the same for each contender. This benchmark was crafted to compare against Forest Flows, our best LIFELIKE contender but a method requiring to re-encode categorical variables. Since our method and ARFs process data natively, we selected only domains with numerical or binary variables (8 domains). On some of these domains, CT-GANs or VCAE crashed on some folds so we do not provide the results for the related domains. Globally, Generative Forests are very competitive against each contender, significantly beating all VCAEs on all metrics and CT-GANs on coverage and density. Tree-based contender methods appear to perform substantially better than neural networks, with FFs performing better than ARFs. Ultimately, our models compare favorably or very favorably against both FFs and ARFs, while on average being much smaller – for example, each FF model can contain up to 6 750 total splits. As our experiments demonstrate (Appendix, Table A6), learning much smaller generative forests with 500 total splits can still buy an advantage in most cases, a notable counterexample being density for FFs. From the memory standpoint, our code (Java) is not optimized yet managed to crunch domains of up to $m \times d \approx$1.5 M while always using less than 1.5Gb memory to train our biggest models. Finally, it clearly emerges from our experiments that there is a domain-dependent "ideal" size (T, J) for the best generative forests. "Guessing" the right size is a prospect for future work.
Missing data imputation with our models: IMPUTE We compared our method against a powerful contender, MICE [42], which relies on round-robin prediction of missing values using supervised models. We optimized MICE by choosing as supervised models trees (CART) and random forests (RFs, we increased the number of trees to 100 for better results). We create a 5-fold experiment; in each fold, we remove a fixed % of observed values (data missing completely at random, MCAR). We then use the data *with missing values* as input to MICE or us to learn models which are then used to predict the missing values. We compute a per-feature discrepancy, the average error probability (**perr**, for categorical variables), and the root mean square error (**rmse**, numerical variables). We also compute one of the simplest baselines, which consists in imputing each variable from its empirical `marginal`. *Results summary* Table 4 puts the spotlight on two domains. The Appendix provides many more plots. From all results, we conclude that generative forests can compete or beat MICE.RF while using hundred times less trees (eventually using just stumps, when $J = T = 20$). From all our results, we also conclude that there is a risk of overfitting if J and / or T are too large. This could motivate further work on pruning generative models. For GF, an unexpected pattern is that the pattern "small number of small trees works well" can work on real-world domains as well (see also Appendix), thereby demonstrating that the nice result displayed in Table 1 generalises to bigger domains.
Density estimation: DENSITY We compared GF vs kernel density estimation (KDE) [4, 24, 34]. The experimental setting is the same as for LIFELIKE: in each of the 5-fold stratified cross validation fold, we use the training sample to learn a model (with KDE or GF.BOOST) which is then used to predict the observation's density in the the test sample. The higher the density, the better the model. The

Domain	Sinkhorn↓			Coverage↑			Density↑			F1 measure↓		
	us (GF)	ARF	pval	us (GF)	ARF	pval	us (GF)	ARF	pval	us (GF)	ARF	pval
ring	⋆0.285±0.008	0.286±0.007	0.62	⋆0.968±0.010	0.960±0.008	0.05	⋆1.031±0.049	0.976±0.030	0.11	⋆0.070±0.030	0.086±0.047	0.54
circ	⋆0.351±0.005	0.355±0.005	0.33	0.945±0.015	⋆0.962±0.021	0.35	⋆0.993±0.045	0.954±0.011	0.05	0.514±0.529	⋆0.507±0.032	0.92
grid	⋆0.390±0.002	0.394±0.002	0.02	⋆0.975±0.009	0.908±0.010	▲	⋆0.995±0.077	0.630±0.050	▲	⋆0.017±0.015	0.043±0.012	0.04
rand	⋆0.286±0.003	0.288±0.002	0.36	⋆0.961±0.012	0.953±0.004	0.29	⋆0.979±0.023	0.940±0.027	▲	⋆0.014±0.013	0.029±0.010	▲
wred	⋆0.980±0.032	1.099±0.031	▲	⋆0.962±0.028	0.929±0.010	0.14	⋆0.961±0.026	0.801±0.028	▲	⋆0.459±0.031	0.531±0.028	▲
wwhi	⋆1.064±0.003	1.150±0.021	▲	⋆0.963±0.002	0.946±0.012	0.10	⋆0.989±0.017	0.941±0.013	▲	⋆0.492±0.037	0.500±0.027	0.04
comp	⋆0.532±0.008	0.535±0.033	0.86	0.556±0.021	⋆0.560±0.037	0.75	0.430±0.008	⋆0.440±0.011	0.11	⋆0.496±0.021	0.520±0.020	0.26
arti	⋆0.821±0.004	0.849±0.010	▲	⋆0.947±0.003	0.892±0.014	▲	⋆0.967±0.013	0.747±0.013	▲	⋆0.429±0.056	0.512±0.037	▲
	us (GF)	CT	pval	us (GF)	CT	pval	us (GF)	CT	pval	us (GF)	CT	pval
ring	⋆0.285±0.008	0.351±0.042	0.01	⋆0.968±0.010	0.798±0.050	▲	⋆1.031±0.049	0.757±0.025	▲	⋆0.070±0.030	0.100±0.073	0.40
circ	⋆0.351±0.005	0.435±0.075	0.07	⋆0.945±0.015	0.734±0.041	▲	⋆0.993±0.045	0.401±0.033	▲	⋆0.514±0.529	0.746±0.033	▲
grid	⋆0.390±0.002	0.408±0.019	0.01	⋆0.975±0.009	0.828±0.053	▲	⋆0.995±0.077	0.649±0.058	▲	⋆0.017±0.015	0.034±0.011	0.21
rand	⋆0.286±0.003	0.327±0.024	0.03	⋆0.961±0.012	0.659±0.049	▲	⋆0.979±0.023	0.582±0.035	▲	⋆0.014±0.013	0.079±0.053	0.06
wred	⋆0.980±0.032	1.384±0.047	▲	⋆0.962±0.028	0.808±0.016	▲	⋆0.961±0.026	0.589±0.129	▲	⋆0.459±0.031	0.654±0.030	▲
wwhi	⋆1.064±0.003	1.158±0.009	▲	⋆0.963±0.002	0.894±0.026	▲	⋆0.989±0.017	0.953±0.035	0.05	⋆0.492±0.037	0.581±0.043	▲
	us (GF)	FF	pval	us (GF)	FF	pval	us (GF)	FF	pval	us (GF)	FF	pval
ring	0.285±0.008	⋆0.276±0.001	0.07	⋆0.968±0.010	0.957±0.020	0.09	1.031±0.049	⋆1.045±0.020	0.28	0.070±0.030	⋆0.051±0.031	0.07
circ	⋆0.351±0.005	0.354±0.003	0.27	0.945±0.020	⋆0.956±0.015	0.25	⋆0.993±0.045	0.989±0.027	0.79	⋆0.514±0.529	0.530±0.028	0.46
grid	⋆0.390±0.002	0.393±0.001	0.04	⋆0.975±0.009	0.954±0.012	0.01	0.995±0.077	⋆1.013±0.050	0.21	⋆0.017±0.015	0.045±0.007	▲
rand	⋆0.286±0.003	0.287±0.001	0.43	⋆0.961±0.012	0.927±0.011	0.02	0.979±0.023	⋆1.000±0.008	0.05	⋆0.014±0.013	0.028±0.008	0.09
wred	⋆0.980±0.032	1.030±0.029	▲	⋆0.962±0.028	0.954±0.021	0.71	0.961±0.026	⋆1.001±0.034	0.04	0.459±0.031	⋆0.458±0.052	0.97
wwhi	⋆1.064±0.003	1.097±0.007	▲	⋆0.963±0.002	0.945±0.007	▲	⋆0.989±0.017	0.970±0.023	0.13	⋆0.492±0.037	0.498±0.041	0.78
comp	⋆0.532±0.007	0.891±0.007	▲	⋆0.556±0.021	0.548±0.027	0.32	⋆0.430±0.008	0.404±0.013	▲	⋆0.496±0.021	0.526±0.018	0.04
arti	⋆0.821±0.004	0.834±0.016	0.12	⋆0.947±0.003	0.879±0.008	▲	⋆0.967±0.013	0.774±0.016	▲	⋆0.429±0.056	0.530±0.032	▲
	us (GF)	VC-G	pval	us (GF)	VC-G	pval	us (GF)	VC-G	pval	us (GF)	VC-G	pval
ring	⋆0.285±0.008	0.392±0.005	▲	⋆0.968±0.010	0.364±0.037	▲	⋆1.031±0.049	0.132±0.013	▲	⋆0.070±0.030	0.291±0.035	▲
circ	⋆0.351±0.005	0.415±0.012	▲	⋆0.945±0.015	0.620±0.020	▲	⋆0.993±0.045	0.265±0.015	▲	⋆0.514±0.529	0.805±0.012	▲
grid	⋆0.390±0.002	0.400±0.003	▲	⋆0.975±0.009	0.165±0.037	▲	⋆0.995±0.077	0.042±0.012	▲	⋆0.017±0.015	0.065±0.002	▲
rand	⋆0.286±0.003	0.414±0.003	▲	⋆0.961±0.012	0.317±0.033	▲	⋆0.979±0.023	0.156±0.018	▲	⋆0.014±0.013	0.224±0.019	▲
wred	⋆0.980±0.032	1.105±0.042	▲	⋆0.962±0.028	0.913±0.013	0.04	⋆0.961±0.026	0.821±0.041	▲	⋆0.459±0.031	0.537±0.007	▲
wwhi	⋆1.064±0.003	1.181±0.019	▲	⋆0.963±0.002	0.938±0.008	▲	⋆0.989±0.017	0.907±0.026	▲	⋆0.492±0.037	0.527±0.028	▲
	us (GF)	VC-D	pval	us (GF)	VC-D	pval	us (GF)	VC-D	pval	us (GF)	VC-D	pval
ring	⋆0.285±0.008	0.390±0.011	▲	⋆0.968±0.010	0.331±0.067	▲	⋆1.031±0.049	0.122±0.028	▲	⋆0.070±0.030	0.319±0.037	▲
circ	⋆0.351±0.005	0.411±0.004	▲	⋆0.945±0.015	0.649±0.055	▲	⋆0.993±0.045	0.269±0.026	▲	⋆0.514±0.529	0.813±0.018	▲
grid	⋆0.390±0.002	0.398±0.002	▲	⋆0.975±0.009	0.162±0.034	▲	⋆0.995±0.077	0.043±0.009	▲	⋆0.017±0.015	0.064±0.000	▲
rand	⋆0.286±0.003	0.414±0.003	▲	⋆0.961±0.012	0.312±0.040	▲	⋆0.979±0.023	0.149±0.018	▲	⋆0.014±0.013	0.225±0.017	▲
wred	⋆0.980±0.032	1.383±0.037	▲	⋆0.962±0.028	0.868±0.042	0.02	⋆0.961±0.026	0.738±0.030	▲	⋆0.459±0.031	0.587±0.019	▲
wwhi	⋆1.064±0.003	1.484±0.056	▲	⋆0.963±0.002	0.876±0.027	▲	⋆0.989±0.017	0.891±0.010	▲	⋆0.492±0.037	0.608±0.037	▲
	us (GF)	VC-R	pval	us (GF)	VC-R	pval	us (GF)	VC-R	pval	us (GF)	VC-R	pval
ring	⋆0.285±0.008	0.388±0.004	▲	⋆0.968±0.010	0.331±0.065	▲	⋆1.031±0.049	0.124±0.022	▲	⋆0.070±0.030	0.322±0.023	▲
circ	⋆0.351±0.005	0.402±0.003	▲	⋆0.945±0.015	0.664±0.017	▲	⋆0.993±0.045	0.274±0.006	▲	⋆0.514±0.529	0.806±0.022	▲
grid	⋆0.390±0.002	0.399±0.003	▲	⋆0.975±0.009	0.153±0.032	▲	⋆0.995±0.077	0.038±0.006	▲	⋆0.017±0.015	0.065±0.001	▲
rand	⋆0.286±0.003	0.417±0.009	▲	⋆0.961±0.012	0.315±0.023	▲	⋆0.979±0.023	0.145±0.021	▲	⋆0.014±0.013	0.229±0.022	▲
wred	⋆0.980±0.032	1.365±0.098	▲	⋆0.962±0.028	0.839±0.027	▲	⋆0.961±0.026	0.716±0.072	▲	⋆0.459±0.031	0.596±0.047	▲
wwhi	⋆1.064±0.003	1.353±0.038	▲	⋆0.963±0.002	0.747±0.014	▲	⋆0.989±0.017	0.942±0.029	0.02	⋆0.492±0.037	0.729±0.024	▲

Table 5: LIFELIKE: comparison of Generative Forests (us, GF), with $T = 500$ trees, $J =$2 000 total splits to contenders: Adversarial Random Forests (ARF, 200 trees [44]), CT-GAN (CT, 1 000 epochs [47]), Forest Flows (FF, 50 trees [17]) and Vine copulas autoencoders (VCAE, VC-* where *=G for Gaussian, D for Direct and R for Regular [41]). Metrics used are regularized OT (Sinkhorn), Coverage, Density and F1 measure. Values shown = average over the 5-folds ± std dev. . The best average for us vs contender is shown with a star "⋆". Convention for p-values: computed using a Student pared t-test; if $p < 0.01$, value is replaced by ▲(we beat the contender) or ▼(the contender beats us). See text for details.

GF models we consider are the same as in LIFELIKE ($T = 500$, $J =$2 000).
Results summary Table (3, right) summarizes the results (plots: Appendix, Section V.V.3.7). The leftmost column ("GF $\gg$ KDE") counts domains for which there exists an iteration j for GF after which we tend to be statistically better (and never statistically worse) than KDE. The rightmost column ("KDE $\gg$ GF") counts domains for which KDE is always statistically better than GF. The last type of plots appears to be those for which there is no such statistical difference (central column). The results support the conclusion that GF can also be good models to carry out density estimation.

7 Conclusion

In this paper, we have introduced new generative models based on ensemble of trees and a top-down boosting-compliant training algorithm which casts training in a supervised 2-classes framework. Those generative forests allow not just efficient data generation: they also allow efficient missing data imputation and density estimation. Experiments have been carried out on all three problems against a variety of contenders, demonstrating the ability of small generative forests to perform well against potentially much more complex contenders. A responsible use of such models necessarily includes restriction to tabular data (Section 1). Experiments clearly show that there is a domain-dependent "best fit" size for our models; estimating it can be done via cross-validation but an important question, left for future work, is how to directly "guess" the right model size. A pruning algorithm with good generalization seems like a very promising direction.

Acknowledgments

The authors thank the reviewers for engaging discussions and many suggestions that helped to improve the paper's content.

References

[1] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, 1984.

[2] A. Buja, W. Stuetzle, and Y. Shen. Loss functions for binary class probability estimation ans classification: structure and applications, 2005. Technical Report, University of Pennsylvania.

[3] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *22nd KDD*, pages 785–794, 2016.

[4] Y.-C. Chen. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1:161–187, 2017.

[5] Y. Choi, A. Vergari, and G. Van den Broeck. Probabilistic circuits: a unifying framework for tractable probabilistic models, 2020. `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`.

[6] M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung P. Nel, and S. Malhotra. *Notes from the AI frontier*. McKinsey Global Institute, 2018.

[7] A.-H.-C. Correia, R. Peharz, and C.-P. de Campos. Joints in random forests. In *NeurIPS'20*, 2020.

[8] M. Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. In *NIPS*26*, pages 2292–2300, 2013.

[9] A. Darwiche. A logical approach to factoring belief networks. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *KR'02*, pages 409–420. Morgan Kaufmann, 2002.

[10] D. Dua and C. Graff. UCI machine learning repository, 2021.

[11] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A.-C. Courville. Adversarially learned inference. In *ICLR'17*. OpenReview.net, 2017.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*27*, pages 2672–2680, 2014.

[13] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? In *NeurIPS'22 Datasets and Benchmarks*, 2022.

[14] A. Grover and S. Ermon. Boosted generative models. In *AAAI'18*, pages 3077–3084. AAAI Press, 2018.

[15] G.-E. Hinton. The forward-forward algorithm: Some preliminary investigations. *CoRR*, abs/2212.13345, 2022.

[16] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *MLJ*, 11:63–91, 1993.

[17] A. Jolicoeur-Martineau, K. Fatras, and T. Kachman. Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees. In *AISTATS'24*, volume 238, pages 1288–1296. PMLR, 2024.

[18] M.J. Kearns. Thoughts on hypothesis boosting, 1988. ML class project.

[19] M.J. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proc. of the 28th ACM STOC*, pages 459–468, 1996.

[20] D.-P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR'14*, 2014.

[21] D.-E. Knuth. Two notes on notation. *The American Mathematical Monthly*, 99(5):403–422, 1992.

[22] A. Krizhevsky, I. Sutskever, and G.-E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*25*, pages 1106–1114, 2012.

[23] S. Lang, M. Mundt, F. Ventola, R. Peharz, and K. Kersting. Elevating perceptual sample quality in pcs through differentiable sampling. In *NeurIPS 2021 Workshop on Pre-Registration in Machine Learning, 13 December 2021, Virtual*, volume 181 of *Proceedings of Machine Learning Research*, pages 1–25. PMLR, 2021.

[24] Q. Li and J.-S. Racine. Nonparametric estimation of distributions with categorical and continuous data. *Journal of Multivariate Analysis*, 86:266–292, 2003.

[25] Y. Mansour, R. Nock, and R.-C. Williamson. Random classification noise does not defeat all convex potential boosters irrespective of model choice. In *ICML'23*, 2023.

[26] P. Maréchal. On a functional operation generating convex functions, part 1: duality. *J. of Optimization Theory and Applications*, 126:175–189, 2005.

[27] P. Maréchal. On a functional operation generating convex functions, part 2: algebraic properties. *J. of Optimization Theory and Applications*, 126:375–366, 2005.

[28] D. McElfresh, S. Khandagale, J. Valverde, V. Prasad C, G. Ramakrishnan, M. Goldblum, and C. White. When do neural nets outperform boosted trees on tabular data? In *NeurIPS'23 Datasets and Benchmarks*, 2023.

[29] A. Menon and C.-S. Ong. Linking losses for density ratio and class-probability estimation. In *33rd ICML*, pages 304–313, 2016.

[30] M.-F. Naeem, S.-J. Oh, Y. Uh, Y. Choi, and J. Yoo. Reliable fidelity and diversity metrics for generative models. In *ICML'20*, volume 119 of *Proceedings of Machine Learning Research*, pages 7176–7185. PMLR, 2020.

[31] R. Nock and M. Guillame-Bert. Generative trees: Adversarial and copycat. In *39th ICML*, pages 16906–16951, 2022.

[32] R. Nock, A.-K. Menon, and C.-S. Ong. A scaled Bregman theorem with applications. In *NIPS*29*, pages 19–27, 2016.

[33] S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: training generative neural samplers using variational divergence minimization. In *NIPS*29*, pages 271–279, 2016.

[34] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33:1065–1076, 1962.

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

[36] J. R. Quinlan. *C4.5 : programs for machine learning*. Morgan Kaufmann, 1993.

[37] M.-D. Reid and R.-C. Williamson. Information, divergence and risk for binary experiments. *JMLR*, 12:731–817, 2011.

[38] R. Sánchez-Cauce, I. París, and F.-J. Díez. Sum-product networks: A survey. *IEEE Trans.PAMI*, 44(7):3821–3839, 2022.

[39] L.-J. Savage. Elicitation of personal probabilities and expectations. *J. of the Am. Stat. Assoc.*, pages 783–801, 1971.

[40] M. Sugiyama and M. Kawanabe. *Machine Learning in Non-Stationary Environments - Introduction to Covariate Shift Adaptation*. Adaptive computation and machine learning. MIT Press, 2012.

[41] N. Tagasovska, D. Ackerer, and T. Vatter. Copulas as high-dimensional generative models: Vine copula autoencoders. In *NeurIPS*32*, pages 6525–6537, 2019.

[42] S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3):1–67, 2011.

[43] A. Vergari, Y. Choi, and R. Peharz. Probabilistic circuits: Representations, inference, learning and applications, 2022. NeurIPS'22 tutorials.

[44] D.-S. Watson, K. Blesch, J. Kapar, and M.-N. Wright. Adversarial random forests for density and generative modeling. In *AISTATS'23*, Proceedings of Machine Learning Research. PMLR, 2023.

[45] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[46] C. Xiao, P. Zhong, and C. Zheng. BourGAN: Generative networks with metric embeddings. In *NeurIPS'18*, pages 2275–2286, 2018.

[47] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional GAN. In *NeurIPS*32*, pages 7333–7343, 2019.

Appendix

To differentiate with the numberings in the main file, the numbering of Theorems, etc. is letter-based (A, B, ...).

Table of contents

I Related work

The typical Machine Learning (ML) problem usually contains at least three parts: (i) a training algorithm minimizes (ii) a loss function to output an object whose key part is (iii) a model. The main problem we are interested in is data generation as generally captured by "Generative AI". The type of data we are interested in still represents one of the most prevalent form of data: tabular data [6]. When it comes to tabular data, a singular phenomenon of the data world can be observed: there is a fierce competition on part (iii) above, the *models*. When data has other forms, like images, the ML community has generally converged to a broad idea of what the best models look like at a high-level for many generic tasks: neural networks[‡]. Tabular data offers no such consensus yet, even on well-defined tasks like supervised learning [13]. In fact, even on such "simple tasks" the consensus is rather that there is no such consensus [28]. It is an understatement to state that in the broader context of all tasks of interest to us, a sense of a truly intense competition emerges, whose "gradient" clearly points towards simultaneously the most complex / expressive / tractable models, as shown in [43, Slides 27, 53]. One can thus end up finding models based on trees [7, 31, 44], neural networks [12, 14, 20, 47], probabilistic circuits [5, 43, 38], kernel methods [4, 34], graphical models [41] (among others: note that some are in fact hybrid models).

So the tabular data world is blessed with a variety of possible models to solve problems like the ones we are interested in, *but* – and this is another singular phenomenon of the tabular data world –, getting the best solutions is not necessarily a matter of competing on size or compute. In fact, it can be the opposite: striving for model simplicity or (non exclusive) lightweight tuning can substantially pay off [28]. In relative terms, this phenomenon is not new in the tabular data world: it has been known for more than two decades [16]. That it has endured all major revolutions in ML points to the fact that lasting solutions can be conveniently addressing *all* three parts (i–iii) above at once on models, algorithms and losses.

From this standpoint, the closest approaches to ours are [44] and [31], first and foremost because the models include trees with a stochastic activation of edges to pick leaves, and a leaf-dependent data generation process. While [31] learn a single tree, [44] use a way to generate data from a set of trees – called an adversarial random forest – which is simple: sample a tree, and then sample an observation from the tree. Hence, the distribution is a convex combination of the trees' density. This is simple but it suffers several drawbacks: (i) each tree has to be accurate enough and thus big enough to model "tricky" data for tree-based models (tricky can be low-dimensional, see the 2D data of Table 1, main file); (ii) if leaves' samplers are simple, which is the case for [31] and our approach, it makes it tricky to learn sets of simple models, such as when trees are stumps (we do not have this issue, see Table 1). In our case, while our models include sets of trees, generating one observation makes use of leaves in *all* trees instead of just one as in [44] (Figure 1, main file). We note that the primary goal of that latter work is in fact not to generate data [44, Section 4].

Theoretical results that are relevant to data generation are in general scarce compared to the flurry of existing methods if we omit the independent convergence rates of the toolbox often used, such as for (stochastic) gradient descent. Specific convergence results are given in [44], but they are relevant to statistical consistency (infinite sample) and they also rely on assumptions that are not realistic for real world domains, such as Lipschitz continuity of the target density, with second derivative continuous, square integrable and monotonic. The assumption made on models, that splitting probabilities on trees is lowerbounded by a constant, is also impeding to model real world data.

In the generative trees of [31], leaf generation is the simplest possible: it is uniform. This requires big trees to model real-world or tricky data. On the algorithms side, [31] introduce two training algorithms in the generative adversarial networks (GAN) framework [12], thus involving the generator to train but also a discriminator, which is a decision tree in [31]. The GAN framework is very convenient to tackle (ii) above in the context of our tasks because it allows to tie using first principles the problem of learning a density or a sampler and that of training a model (a "discriminator") to distinguish fakes from real, model that parameterizes the loss optimized. An issue with neural networks is that this parameterization has an uncontrollable slack unless the discriminator is extremely complex [33]. The advantage of using trees as in [31] is that for such classifiers, the slack disappears because the models are calibrated, so there is a tight link between training the generator and the discriminator. [31] go

[‡]Whether such a perception is caused by the models themselves or the dazzling amount of optimisation that has progressively wrapped the models, as advocated for the loss in [23], is not the focus of our paper.

Algorithm 4 ITERATIVESUPPORTUPDATE($\{\Upsilon_t\}_{t=1}^T$)

```
Input: Trees $\{\Upsilon_t\}_{t=1}^T$ of a GF;
Output: sampling support $\mathcal{C}_s$ for one observation;
Step 1 : $\mathcal{C}_s \leftarrow \mathcal{X}$;
Step 2 : INIT($\{\Upsilon_t\}_{t=1}^T$);
Step 3 : for $t \in [T]$
    Step 2.1 : while !$\Upsilon_t$.done
        Step 2.1.1 : STARUPDATE($\Upsilon_t$, $\mathcal{C}_s$, R);
return $\mathcal{C}_s$;
```

Algorithm 5 CONCURRENTSUPPORTUPDATE

```
Step 1 : while !done
    Step 1.1 : P[accessible] // locks $\mathcal{C}_s$
    Step 1.2 : STARUPDATE(this,$\mathcal{C}_s$, R);
    Step 1.3 : V[accessible] // unlocks $\mathcal{C}_s$
```

further, showing that one can replace the adversarial training by a *copycat* training, involving copying parts of the discriminator in the generator to speed-up training (also discussed in [15] for neural nets), with strong rates on training in the original boosting model. There is however a limitation in the convergence analysis of [31] as losses have to be symmetric, a property which is not desirable for data generation since it ties the misclassification costs of real and fakes with no argument to do so in general.

Our paper lifts the whole setting of [31] to models that can fit more complex densities using simpler models (Table 1), using sets of trees that can be much smaller than those of [44] (Figure 1); training such models is achieved by merging the two steps of copycat training into a single one where only the generator is trained, furthermore keeping strong rates on training via the original boosting model, all this while getting rid of the undesirable symmetry assumption of [31] for the loss at hand.

II Additional content

In this additional content, we provide the three ways to pick the trees in a Generative Forest to generate one observation (sequential, concurrent, randomized), and then give a proof that the optimal splitting threshold on a continuous variable when training a generative forest using GF.BOOST is always an observed value if there is one tree, but can be another value if there are more (thus highlighting some technical difficulties of one wants to stick to the optimal choice of splitting).

II.1 Sequentially choosing trees in a GF for the generation of observations

II.2 Concurrent generation of observations

We provide a concurrent generation using Algorithm 5, which differs from Algorithm 4 (main file). In concurrent generation, each tree runs concurrently algorithm UPDATESUPPORT (hence the use of the Java-style `this` handler), with an additional global variables (in addition to $\mathcal{C}_s$, initialized to $\mathcal{X}$): a Boolean semaphore `accessible` implementing a lock, whose value 1 means $\mathcal{C}$ is available for an update (and otherwise it is locked by a tree in Steps 1.2/1.3 in the set of trees of the GF). We assume that INIT has been run beforehand (eventually locally).

II.3 Randomized generation of observations

We provide a general randomized choice of the sequence of trees for generation, in Algorithm 6.

Algorithm 6 RANDOMIZEDSUPPORTUPDATE

```
Step 1 : INIT($\{\Upsilon_t\}_{t=1}^T$); $\mathbb{I} \leftarrow \{1,2,...,T\}$;
Step 2 : while ($\mathbb{I}$ != $\varnothing$)
    Step 2.1 : $i \leftarrow$ RANDOM($\mathbb{I}$);
    Step 2.2 : STARUPDATE($\Upsilon_i$,$\mathcal{C}_s$, R);
    Step 2.3 : if ($\Upsilon_i$.done) then $\mathbb{I} \leftarrow \mathbb{I}\backslash\{i\}$;
```

III Supplementary material on proofs

III.1 Proof of Lemma 4.4

Given sequence $\boldsymbol{v}$ of dimension $\dim(\boldsymbol{v})$, denote $\{\mathcal{C}_j\}_{j\in[1+\dim(\boldsymbol{v})]}$ the sequence of subsets of the domain appearing in the parameters of UPDATESUPPORT through sequence $\boldsymbol{v}$, to which we add a last element, $\mathcal{C}_s$ (and its first element is $\mathcal{X}$). If we let $\mathcal{X}_j$ denote the support of the corresponding star node whose Bernoulli event is triggered at index j in sequence $\boldsymbol{v}$ (for example, $\mathcal{X}_1 = \mathcal{X}$), then we easily get

$$\mathcal{C}_j \quad \subseteq \quad \mathcal{X}_j, \forall j \in [\dim(\boldsymbol{v})], \tag{5}$$

indicating the Bernoulli probability in Step 1.1 of STARUPDATE is always defined. We then compute

$$\begin{aligned}
p_{\mathrm{G}}\left[\mathcal{C}_s|\boldsymbol{v}\right] &= p_{\mathrm{G}}(\cap_j \mathcal{C}_j) && (6)\\
&= \prod_{j=1}^{N} p_{\mathrm{G}}\left[\mathcal{C}_{j+1}|\mathcal{C}_j\right] && (7)\\
&= \prod_{j=1}^{N} \frac{p_{\mathrm{R}}\left[\mathcal{C}_{i+1}\right]}{p_{\mathrm{R}}\left[\mathcal{C}_i\right]} && (8)\\
&= \frac{p_{\mathrm{R}}\left[\mathcal{C}_{N+1}\right]}{p_{\mathrm{R}}\left[\mathcal{C}_0\right]} && (9)\\
&= \frac{p_{\mathrm{R}}\left[\mathcal{C}_s\right]}{p_{\mathrm{R}}\left[\mathcal{X}\right]}\\
&= p_{\mathrm{R}}\left[\mathcal{C}_s\right]. && (10)
\end{aligned}$$

(7) holds because updating the support generation is Markovian in a GF, (8) holds because of Step 3.2 and (9) is a simplification through cancelling terms in products.

III.2 Proof of Theorem 5.2

Notations: we iteratively learn generators $\mathrm{G}_0, \mathrm{G}_1, ..., \mathrm{G}_J$ where G_0 is just a root. We let $\mathcal{P}(\mathrm{G}_j)$ denote the partition induced by the set of trees of G_j, recalling that each element is the (non-empty) intersection of the support of a set of leaves, one for each tree (for example, $\mathcal{P}(\mathrm{G}_0) = \{\mathcal{X}\}$). The criterion minimized to build G_{j+1} from G_j is

$$\mathbb{L}(\mathrm{G}_j) \quad \doteq \quad \sum_{\mathcal{C}\in\mathcal{P}(\mathrm{G}_j)} p_{\mathrm{M}}[\mathcal{C}] \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right). \tag{11}$$

The proof entails fundamental notions on loss functions, models and boosting. We start with loss functions. A key function we use is

$$g^{\pi}(t) \doteq (\pi t + 1 - \pi) \cdot \underline{L}\left(\frac{\pi t}{\pi t + 1 - \pi}\right), \forall t \in \mathbb{R}_+,$$

which is concave [37, Appendix A.3].

Lemma A.

$$\mathbb{D}_\ell\left(\mathrm{R}, \mathrm{G}_{J-1}\right) - \mathbb{D}_\ell\left(\mathrm{R}, \mathrm{G}_J\right) \quad = \quad \mathbb{L}(\mathrm{G}_{J-1}) - \mathbb{L}(\mathrm{G}_J). \tag{12}$$

Proof. We make use of the following important fact about GFs:

(F1) For any $\mathcal{X}' \in \mathcal{P}_J$, $\int_{\mathcal{X}'} \mathrm{d}\mathrm{G}_J = \int_{\mathcal{X}'} \mathrm{d}\mathrm{R}$.

We have from the proof of [31, Lemma 5.6],

$$\mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_J) = \sum_{\mathcal{X}' \in \mathcal{P}_J} \left\{ g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[\frac{\mathrm{d}\mathrm{G}_J}{\mathrm{d}\mathrm{U}}\right]\right) - \mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[g^\pi\left(\frac{\mathrm{d}\mathrm{R}}{\mathrm{d}\mathrm{U}}\right)\right]\right\}. \quad (13)$$

We note that **(F1)** implies (13) is just a sum of slacks of Jensen's inequality. We observe

$$\begin{aligned}
&\mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_{J-1}) - \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_J) \\
&= \sum_{\mathcal{X}' \in \mathcal{P}_{J-1}} \left\{ g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[\frac{\mathrm{d}\mathrm{G}_{J-1}}{\mathrm{d}\mathrm{U}}\right]\right) - \mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[g^\pi\left(\frac{\mathrm{d}\mathrm{R}}{\mathrm{d}\mathrm{U}}\right)\right]\right\} \\
&\quad - \sum_{\mathcal{X}' \in \mathcal{P}_{J}} \left\{ g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[\frac{\mathrm{d}\mathrm{G}_{J}}{\mathrm{d}\mathrm{U}}\right]\right) - \mathbb{E}_{\mathrm{U}|\mathcal{X}'}\left[g^\pi\left(\frac{\mathrm{d}\mathrm{R}}{\mathrm{d}\mathrm{U}}\right)\right]\right\} \\
&= \sum_{(\mathcal{X}_{\mathtt{t}}, \mathcal{X}_{\mathtt{f}}) \in \mathcal{P}_J^{\mathrm{split}}} \left\{ g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{t}} \cup \mathcal{X}_{\mathtt{f}}}\left[\frac{\mathrm{d}\mathrm{G}_{J-1}}{\mathrm{d}\mathrm{U}}\right]\right) - \sum_{\mathtt{v} \in \{\mathtt{t},\mathtt{f}\}} g^\pi\left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{v}}}\left[\frac{\mathrm{d}\mathrm{G}_J}{\mathrm{d}\mathrm{U}}\right]\right)\right\},
\end{aligned}$$

where $\mathcal{P}_J^{\mathrm{split}}$ contains all couples $(\mathcal{X}_{\mathtt{t}}, \mathcal{X}_{\mathtt{f}})$ such that $\mathcal{X}_{\mathtt{t}}, \mathcal{X}_{\mathtt{f}} \in \mathcal{P}_J$ and $\mathcal{X}_{\mathtt{t}} \cup \mathcal{X}_{\mathtt{f}} \in \mathcal{P}_{J-1}$. These unions were subsets of the partition of $\mathcal{X}$ induced by the set of trees of G_{J-1} *and* that were cut by the predicate p put at the leaf λ_* that created G_J from G_{J-1}. To save space, denote $\mathcal{X}_{\mathtt{tf}} \doteq \mathcal{X}_{\mathtt{t}} \cup \mathcal{X}_{\mathtt{f}}$.

$$\begin{aligned}
&g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{tf}}}\left[\frac{\mathrm{d}\mathrm{G}_{J-1}}{\mathrm{d}\mathrm{U}}\right]\right) - \sum_{\mathtt{v} \in \{\mathtt{t},\mathtt{f}\}} g^\pi\left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{v}}}\left[\frac{\mathrm{d}\mathrm{G}_J}{\mathrm{d}\mathrm{U}}\right]\right) \\
&= \left(\pi \int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{G}_{J-1} + (1-\pi)\int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{U}\right) \cdot \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{G}_{J-1}}{\pi \int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{G}_{J-1} + (1-\pi)\int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{U}}\right) \\
&\quad - \left(\pi \int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{G}_{J} + (1-\pi)\int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{U}\right) \cdot \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{G}_{J}}{\pi \int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{G}_{J} + (1-\pi)\int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{U}}\right) \\
&\quad - \left(\pi \int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{G}_{J} + (1-\pi)\int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{U}\right) \cdot \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{G}_{J}}{\pi \int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{G}_{J} + (1-\pi)\int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{U}}\right). \quad (14)
\end{aligned}$$

We now work on (14). Using **(F1)**, we note $\int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{G}_{J-1} = \int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{R}$ since $\mathcal{X}_{\mathtt{tf}} \in \mathcal{P}_{J-1}$. Similarly, $\int_{\mathcal{X}_{\mathtt{v}}} \mathrm{d}\mathrm{G}_J = \int_{\mathcal{X}_{\mathtt{v}}} \mathrm{d}\mathrm{R}, \forall \mathtt{v} \in \{\mathtt{t}, \mathtt{f}\}$, so we make appear R in (14) and get:

$$\begin{aligned}
&g^\pi \left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{tf}}}\left[\frac{\mathrm{d}\mathrm{G}_{J-1}}{\mathrm{d}\mathrm{U}}\right]\right) - \sum_{\mathtt{v} \in \{\mathtt{t},\mathtt{f}\}} g^\pi\left(\mathbb{E}_{\mathrm{U}|\mathcal{X}_{\mathtt{v}}}\left[\frac{\mathrm{d}\mathrm{G}_J}{\mathrm{d}\mathrm{U}}\right]\right) \\
&= p(\mathtt{tf}) \cdot \left\{ \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{tf}}} \mathrm{d}\mathrm{R}}{p(\mathtt{tf})}\right) - \frac{p(\mathtt{t})}{p(\mathtt{tf})} \cdot \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{t}}} \mathrm{d}\mathrm{R}}{p(\mathtt{t})}\right) - \frac{p(\mathtt{f})}{p(\mathtt{tf})} \cdot \underline{L}\left(\frac{\pi \int_{\mathcal{X}_{\mathtt{f}}} \mathrm{d}\mathrm{R}}{p(\mathtt{f})}\right)\right\}, \quad (15)
\end{aligned}$$

where we let for short

$$p(\mathtt{v}) \doteq \pi \int_{\mathcal{X}_{\mathtt{v}}} \mathrm{d}\mathrm{R} + (1-\pi)\int_{\mathcal{X}_{\mathtt{v}}} \mathrm{d}\mathrm{U}, \forall \mathtt{v} \in \{\mathtt{t}, \mathtt{f}, \mathtt{tf}\}. \quad (16)$$

We finally get

$$\begin{aligned}
&\mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_{J-1}) - \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_J) \\
&= \sum_{\mathcal{C} \in \mathcal{P}_{J-1}} p_{\mathrm{M}}[\mathcal{C}] \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right) - \sum_{\mathcal{C} \in \mathcal{P}_{J-1}} p_{\mathrm{M}}[\mathcal{C}] \cdot \sum_{\mathtt{v} \in \{\mathtt{t},\mathtt{f}\}} \frac{p_{\mathrm{M}}[\mathcal{C}_{\mathrm{p}_{J-1},\mathtt{v}}]}{p_{\mathrm{M}}[\mathcal{C}]} \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}_{\mathrm{p}_{J-1},\mathtt{v}}]}{p_{\mathrm{M}}[\mathcal{C}_{\mathrm{p}_{J-1},\mathtt{v}}]}\right) \\
&= \underline{\mathbb{L}}(\mathrm{G}_{J-1}) - \underline{\mathbb{L}}(\mathrm{G}_J),
\end{aligned}$$

as claimed. The last identity comes from the fact that the contribution to $\underline{\mathbb{L}}(.)$ is the same outside $\mathcal{P}_{J-1}$ for both G_{J-1} and G_J. $\square$

Algorithm 7 MODABOOST($\mathrm{R}, \ell, \mathrm{WL}, J, T$)

```
Input: measure $\mathrm{R}$, SPD loss $\ell$, weak learner WL, iteration number $J \geqslant 1$, number of trees $T$;
Output: PLM (partition-linear model) $H_J$ with $T$ trees;
Step 1 : $\Gamma_0 \doteq \{\Upsilon_t\}_{t=1}^T \leftarrow \{\mathrm{root}, \mathrm{root}, ..., \mathrm{root}\}$; $\mathcal{P}(\Gamma_0) \leftarrow \{\mathcal{X}\}$;
Step 2 : for $j = 1, 2, ..., J$
    Step 2.1 : pick $t \in [T]$;
    Step 2.2 : pick $\lambda \in \Lambda(\Upsilon_t)$;
    Step 2.3 : $h_j \leftarrow \mathrm{WL}(\mathcal{X}_\lambda)$;
               // call to the weak learner for some $h_j$ of the type $h_j \doteq 1_{\mathtt{p}(.)} \cdot K_j$,
               // $K_j$ constant, $\mathtt{p}(.)$ a splitting predicate (e.g. $x_i \geqslant a$)
    Step 2.4 : for $\mathcal{C} \in \mathcal{P}_\lambda(\Gamma_{j-1})$,
```

$$\mathcal{C}^{\mathtt{p}} \leftarrow \mathcal{C} \cap \{\boldsymbol{x} : \mathtt{p}(\boldsymbol{x}) \text{ is } \mathtt{true}\}, \quad (17)$$

$$\alpha(\mathcal{C}^{\mathtt{p}}) \leftarrow (-\underline{L}')\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}^{\mathtt{p}}]}{p_{\mathrm{M}}[\mathcal{C}^{\mathtt{p}}]}\right), \quad (18)$$

$$\mathcal{C}^{\neg\mathtt{p}} \leftarrow \mathcal{C} \cap \{\boldsymbol{x} : \mathtt{p}(\boldsymbol{x}) \text{ is } \mathtt{false}\}, \quad (19)$$

$$\alpha(\mathcal{C}^{\neg\mathtt{p}}) \leftarrow -(-\underline{L}')\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}^{\neg\mathtt{p}}]}{p_{\mathrm{M}}[\mathcal{C}^{\neg\mathtt{p}}]}\right). \quad (20)$$

```
    Step 2.5 : // update $\mathcal{P}(\Gamma_.)$:
```

$$\mathcal{P}(\Gamma_j) \leftarrow (\mathcal{P}(\Gamma_{j-1})\backslash\mathcal{P}_\lambda(\Gamma_{j-1})) \cup \left(\cup_{\mathcal{C}\in\mathcal{P}_\lambda(\Gamma_{j-1}):\mathcal{C}^{\mathtt{p}}\neq\emptyset}\mathcal{C}^{\mathtt{p}}\right) \cup \left(\cup_{\mathcal{C}\in\mathcal{P}_\lambda(\Gamma_{j-1}):\mathcal{C}^{\neg\mathtt{p}}\neq\emptyset}\mathcal{C}^{\neg\mathtt{p}}\right) \quad (21)$$

```
    Step 2.6 : split leaf $\lambda$ at $\Upsilon_t$ using predicate $\mathtt{p}$;
return $H_J(\boldsymbol{x}) \doteq \alpha(\mathcal{C}(\boldsymbol{x}))$ with $\mathcal{C}(\boldsymbol{x}) \doteq \mathcal{C} \in \mathcal{P}(\Gamma_J)$ such that $\boldsymbol{x} \in \mathcal{C}$;
```

We now come to models and boosting. Suppose we have t trees $\{\Upsilon_1, \Upsilon_2, ..., \Upsilon_t\}$ in G_j. We want to split leaf $\lambda \in \Upsilon_t$ into two new leaves, $\lambda_{\mathtt{f}}, \lambda_{\mathtt{t}}$. Let $\mathcal{P}_\lambda(\mathrm{G}_j)$ denote the subset of $\mathcal{P}(\mathrm{G}_j)$ containing only the subsets defined by intersection with the support of λ, $\mathcal{X}_\lambda$. The criterion we minimize can be reformulated as

$$\underline{\mathbb{L}}(\mathrm{G}_j) = \sum_{\lambda\in\Lambda(\Upsilon)} p_{\mathrm{M}}[\mathcal{X}_\lambda] \cdot \sum_{\mathcal{C}\in\mathcal{P}_\lambda(\mathrm{G}_j)} \frac{p_{\mathrm{M}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{X}_\lambda]} \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right), \Upsilon \in \{\Upsilon_1, \Upsilon_2, ..., \Upsilon_t\}.$$

Here, Υ is any tree in the set of trees, since $\cup_{\lambda\in\Lambda(\Upsilon)}\mathcal{P}_\lambda(\mathrm{G}_j)$ covers the complete partition of $\mathcal{X}$ induced by $\{\Upsilon_1, \Upsilon_2, ..., \Upsilon_t\}$. If we had a single tree, the inner sum would disappear since we would have $\mathcal{P}_\lambda(\mathrm{G}_j) = \{\mathcal{X}_\lambda\}$ and so one iteration would split one of these subsets. In our case however, with a set of trees, we still split $\mathcal{X}_\lambda$ but the impact on the reduction of $\underline{\mathbb{L}}$ can be substantially better as we may simultaneously split as many subsets as there are in $\mathcal{P}_\lambda(\mathrm{G}_j)$. The reduction in $\underline{\mathbb{L}}$ can be obtained by summing all reductions to which contribute each of the subsets.

To analyze it, we make use of a reduction to the MODABOOST algorithm of [25]. We present the algorithm in Algorithm 7. The original MODABOOST algorithm trains *partition-linear models*, *i.e.* models whose output is defined by a sum of reals over a set of partitions to which the observation to be classified belongs to. Training means then both learning the organisation of partitions and the reals – which are just the output of a weak classifier given by a weak learner, times a leveraging constant computed by MODABOOST. As in the classical boosting literature, the original MODABOOST includes the computation and updates of *weights*.

In our case, the structure of partition learned is that of a set of trees, each weak classifier h is of the form

$$h(\boldsymbol{x}) \doteq K \cdot \mathtt{p}(\boldsymbol{x}),$$

where K is a real and $\mathtt{p}$ is a Boolean predicate, usually doing an axis-parallel split of $\mathcal{X}$ on an observed feature, such as $\mathtt{p}(\boldsymbol{x}) \equiv 1_{x_i \geqslant a}$.

From now on, it will be interesting to dissociate our generator G_j – which includes a model structure and an algorithm to generate data from this structure– from its set of trees – *i.e.* its model structure – which we denote $\Gamma_j \doteq \{\Upsilon_t\}_{t=1}^T$. MODABOOST learns both Γ_j but also predictions for each possible

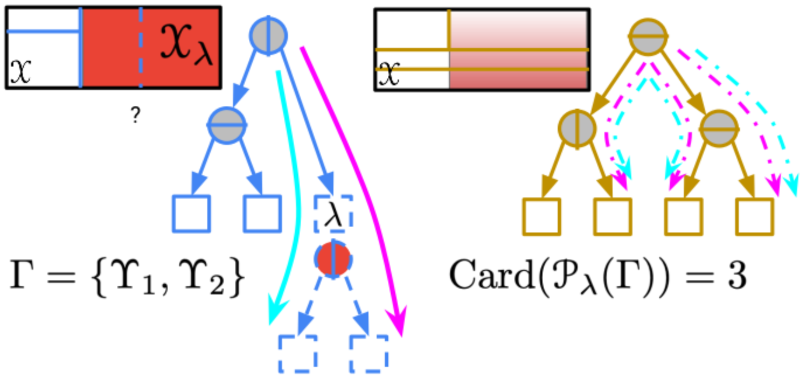

$\mathcal{X}_\lambda$
$\mathcal{X}$
?
λ
$\Gamma = \{\Upsilon_1, \Upsilon_2\}$
$\mathcal{X}$
$\mathrm{Card}(\mathcal{P}_\lambda(\Gamma)) = 3$

Figure III.1: Learning a set of two trees Γ with MODABOOST. If we want a split at leaf λ indicated, then it would takes two steps (but a *single* application of the weak learning assumption, [25, Lemma 5]) of the original MODABOOST to learn Υ_1 alone [25, Section 4]. Each step, figured with a plain arrow, ends with adding a new leaf. In our case however, we have to take into account the partition induced by the leaves of Υ_2, which results in considering not one but three subsets in $\mathcal{P}_\lambda$, thus adding not one but three weak hypotheses simultaneously (one for each of the dashed arrows on Υ_2). Thanks to Jensen's inequality, a guaranteed decrease of the loss at hand can be obtained by a single application of the weak learning assumption at λ, just like in the original MODABOOST.

outcome, predictions that we shall not use since the loss we are minimizing can be made to depend only on the model structure Γ_j. Given the simplicity of the weak classifiers h and the particular nature of the partitions learned, the original MODABOOST of [25] can be simplified to our presentation in Algorithm 7. Among others, the simplification discards the weights from the presentation of the algorithm. What we show entangles two objectives on models and loss functions as we show that

MODABOOST *learns the same structure* Γ *as our* GF.BOOST*, and yields a guaranteed decrease of* $\mathbb{L}(\mathrm{G})$,

and it is achieved via the following Theorem.

Theorem B. MODABOOST *greedily minimizes the following loss function:*

$$\mathbb{L}(\Gamma_j) \quad = \quad \sum_{\lambda \in \Lambda(\Upsilon)} p_{\mathrm{M}}[\mathcal{X}_\lambda] \cdot \sum_{\mathcal{C} \in \mathcal{P}_\lambda(\Gamma_j)} \frac{p_{\mathrm{M}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{X}_\lambda]} \cdot \underline{L}\left(\frac{\pi p_{\mathrm{R}}[\mathcal{C}]}{p_{\mathrm{M}}[\mathcal{C}]}\right), \Upsilon \in \Gamma_j.$$

Furthermore, suppose there exists $\gamma > 0, \kappa > 0$ *such that at each iteration* j*, the predicate* $\mathfrak{p}$ *splits* $\mathcal{X}_\lambda$ *of leaf* $\lambda \in \Lambda(\Upsilon)$ *into* $\mathcal{X}_\lambda^{\mathfrak{p}}$ *and* $\mathcal{X}_\lambda^{\neg\mathfrak{p}}$ *(same nomenclature as in* (17), (19)*) such that:*

$$p_{\mathrm{M}}[\mathcal{X}_\lambda] \quad \geqslant \quad \frac{1}{\mathrm{Card}(\Lambda(\Upsilon))}, \tag{22}$$

$$\left|\frac{\int_{\mathcal{X}_\lambda^{\mathfrak{p}}} \mathrm{dR}}{\int_{\mathcal{X}_\lambda} \mathrm{dR}} - \frac{\int_{\mathcal{X}_\lambda^{\mathfrak{p}}} \mathrm{dU}}{\int_{\mathcal{X}_\lambda} \mathrm{dU}}\right| \quad \geqslant \quad \gamma, \tag{23}$$

$$\min\left\{\frac{\pi p_{\mathrm{R}}[\mathcal{X}_\lambda]}{p_{\mathrm{M}}[\mathcal{X}_\lambda]}, 1 - \frac{\pi p_{\mathrm{R}}[\mathcal{X}_\lambda]}{p_{\mathrm{M}}[\mathcal{X}_\lambda]}\right\} \quad \geqslant \quad \kappa. \tag{24}$$

Then we have the following guaranteed slack between two successive models:

$$\mathbb{L}(\Gamma_{j+1}) - \mathbb{L}(\Gamma_j) \quad \leqslant \quad -\frac{\kappa \gamma^2 \kappa^2}{8\mathrm{Card}(\Lambda(\Upsilon))}, \tag{25}$$

where κ *is such that* $0 < \kappa \leqslant \inf\{\ell'_{-1} - \ell'_1\}$.

Proof sketch The loss function comes directly from [25, Lemma 7]. At each iteration, MODABOOST makes a number of updates that guarantee, once Step 2.4 is completed (because the elements in $\mathcal{P}_\lambda(\Gamma_j)$ are disjoint)

$$\begin{aligned}\underline{\mathbb{L}}(\Gamma_{j+1}) - \underline{\mathbb{L}}(\Gamma_j) &\leqslant -\frac{\kappa}{2} \cdot \sum_{\mathcal{C} \in \mathcal{P}_\lambda(\Gamma_j)} p_{\mathrm{M}}[\mathcal{C}] \cdot \mathbb{E}_{\mathrm{M}|\mathcal{C}} \left[(w_{j+1} - w_j)^2\right] &(26)\\ &= -\frac{\kappa}{2} \cdot p_{\mathrm{M}}[\mathcal{X}_\lambda] \mathbb{E}_{\mathrm{M}|\mathcal{X}_\lambda} \left[(w_{j+1} - w_j)^2\right], &(27)\end{aligned}$$

where the weights are given in [25]. Each expression in the summand of (26) is exactly the guarantee of [25, ineq. before (69)]; all such expressions are not important; what is more important is (26): all steps occurring in Step 2.4 are equivalent to a single step of MODABOOST carried out over the whole $\mathcal{X}_\lambda$. Overall, the number of "aggregated" steps match the counter j in MODABOOST. We then just have to reuse the proof of [25, Theorem B], which implies, in lieu of their [25, eq. (74)]

$$\underline{\mathbb{L}}(\Gamma_{j+1}) - \underline{\mathbb{L}}(\Gamma_j) \leqslant -2\kappa \cdot p_{\mathrm{M}}[\mathcal{X}_\lambda] \cdot \left(\frac{1}{2} \cdot \left(\frac{\int_{\mathcal{X}^{\mathrm{P}}_\lambda} \mathrm{dR}}{\int_{\mathcal{X}_\lambda} \mathrm{dR}} - \frac{\int_{\mathcal{X}^{\mathrm{P}}_\lambda} \mathrm{dU}}{\int_{\mathcal{X}_\lambda} \mathrm{dU}}\right)\right)^2 \cdot \underline{L}^{\mathrm{SQ}} \left(\frac{\pi p_{\mathrm{R}}[\mathcal{X}_\lambda]}{p_{\mathrm{M}}[\mathcal{X}_\lambda]}\right)^2 \quad (28)$$

with $\underline{L}^{\mathrm{SQ}}(u) \doteq u(1-u)$. Noting $2\underline{L}^{\mathrm{SQ}}(u) \geqslant \min\{u, 1-u\}$, we then use (22) – (24), which yields the statement of the Theorem. ■

As a consequence, if we assume that the total number of boosting iterations J is a multiple of the number of trees T, it comes from [25, eq. (29)] that after J iterations, we have

$$\underline{\mathbb{L}}(\Gamma_J) - \underline{\mathbb{L}}(\Gamma_0) \leqslant -\frac{\kappa\gamma^2\kappa^2}{8} \cdot T \log\left(1 + \frac{J}{T}\right). \quad (29)$$

Using Lemma A and the fact that the induction of the sets of trees in MODABOOST is done in the same way as the induction of the set of trees of our generator G in GF.BOOST, we get:

$$\begin{aligned}\pi \cdot \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_J) &= \pi \cdot \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_0) + (\underline{\mathbb{L}}(\mathrm{G}_J) - \underline{\mathbb{L}}(\mathrm{G}_0))\\ &= \pi \cdot \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_0) + (\underline{\mathbb{L}}(\Gamma_J) - \underline{\mathbb{L}}(\Gamma_0))\\ &\leqslant \pi \cdot \mathbb{D}_\ell(\mathrm{R}, \mathrm{G}_0) - \frac{\kappa\gamma^2\kappa^2}{8} \cdot T \log\left(1 + \frac{J}{T}\right),\end{aligned}$$

as claimed.

Remark C. *One may wonder what is the relationship between the loss that we minimize and "conventional" losses used to train generative models. The advantage of our Theorem 5.2 is that by choosing different proper losses, one can get convergence guarantees for different "conventional" losses. Let us illustrate this with the* KL *divergence between measures* A *and* B*, noted* KL(A‖B).

Lemma D. *Suppose the prior satisfies $\pi > 0$ and* R *absolutely continuous with respect to* G*. Then for the choice $\ell \doteq \ell^{\mathrm{LOG}}$ = log-loss, we get*

$$\mathbb{D}_{\ell^{\mathrm{LOG}}}(\mathrm{R}, \mathrm{G}) = \pi \cdot \mathrm{KL}(\mathrm{R}\|\mathrm{G}) - \mathrm{KL}(\pi\mathrm{dR} + (1-\pi)\mathrm{dU}\|\pi\mathrm{dG} + (1-\pi)\mathrm{dU}). \quad (30)$$

Proof. We first recall that for any convex F, we have with our choice of g (assuming $\pi > 0$),

$$\begin{aligned}\check{F}(z) &= \frac{\pi z + 1 - \pi}{\pi} \cdot F\left(\frac{\pi z}{\pi z + 1 - \pi}\right), &(31)\\ \check{F}'(z) &= F\left(\frac{\pi z}{\pi z + 1 - \pi}\right) + \frac{1-\pi}{\pi z + 1 - \pi} \cdot F'\left(\frac{\pi z}{\pi z + 1 - \pi}\right). &(32)\end{aligned}$$

The log-loss has $\ell_1^{\mathrm{LOG}}(u) = -\log u$, $\ell_{-1}^{\mathrm{LOG}}(u) = -\log(1-u)$. It is strictly proper and so it comes

$$\left(\widetilde{-\underline{L}}\right)(z) = z \cdot \log\left(\frac{\pi z}{\pi z + 1 - \pi}\right) + \frac{1-\pi}{\pi} \cdot \log\left(\frac{1-\pi}{\pi z + 1 - \pi}\right).$$

We thus get, in our case

$$\left(\widetilde{-\underline{L}}\right)\left(\frac{\mathrm{dR}}{\mathrm{dU}}\right) = \frac{\mathrm{dR}}{\mathrm{dU}} \cdot \log\left(\frac{\pi\mathrm{dR}}{\pi\mathrm{dR} + (1-\pi)\mathrm{dU}}\right) + \frac{1-\pi}{\pi} \cdot \log\left(\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dR} + (1-\pi)\mathrm{dU}}\right) \quad (33)$$

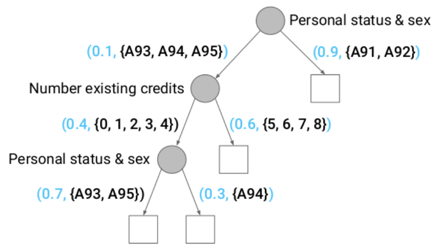


Figure IV.1: A generative tree (GT) associated to UCI German Credit.

and using (32) and the fact that for the log-loss $(-\underline{L})'(u) = \log(u/(1-u))$,

$$\begin{aligned}\left(\widetilde{-\underline{L}}\right)'\left(\frac{\mathrm{dG}}{\mathrm{dU}}\right) &= \frac{\pi\mathrm{dG}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\cdot\log\left(\frac{\pi\mathrm{dG}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)\\ &\quad+\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\cdot\log\left(\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)\\ &\quad+\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\cdot\log\left(\frac{\pi\mathrm{dG}}{(1-\pi)\mathrm{dU}}\right) = \log\left(\frac{\pi\mathrm{dG}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)\end{aligned} \tag{34}$$

Finally, we get, assuming R absolutely continuous with respect to G,

$$\begin{aligned}&\mathbb{D}_{\ell\textsc{log}}(\mathrm{R},\mathrm{G})\\ &\doteq \pi\cdot\int_{\mathcal{X}}\mathrm{dU}\cdot\left(\left(\widetilde{-\underline{L}}\right)\left(\frac{\mathrm{dR}}{\mathrm{dU}}\right)-\left(\widetilde{-\underline{L}}\right)\left(\frac{\mathrm{dG}}{\mathrm{dU}}\right)-\left(\frac{\mathrm{dR}-\mathrm{dG}}{\mathrm{dU}}\right)\cdot\left(\widetilde{-\underline{L}}\right)'\left(\frac{\mathrm{dG}}{\mathrm{dU}}\right)\right)\\ &= \int_{\mathcal{X}}\left\{\begin{array}{l}\pi\mathrm{dR}\cdot\log\left(\frac{\pi\mathrm{dR}}{\pi\mathrm{dR}+(1-\pi)\mathrm{dU}}\right)+(1-\pi)\mathrm{dU}\cdot\log\left(\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dR}+(1-\pi)\mathrm{dU}}\right)\\ -\pi\mathrm{dG}\cdot\log\left(\frac{\pi\mathrm{dG}}{\pi\mathrm{dG}-(1-\pi)\mathrm{dU}}\right)-(1-\pi)\mathrm{dU}\cdot\log\left(\frac{(1-\pi)\mathrm{dU}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)\\ -\pi\mathrm{dR}\cdot\log\left(\frac{\pi\mathrm{dG}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)+\pi\mathrm{dG}\cdot\log\left(\frac{\pi\mathrm{dG}}{\pi\mathrm{dG}+(1-\pi)\mathrm{dU}}\right)\end{array}\right. &(35)\\ &= \int_{\mathcal{X}}\left\{\begin{array}{l}\pi\mathrm{dR}\cdot\log\left(\frac{\mathrm{dR}}{\mathrm{dG}}\right)\\ -(\pi\mathrm{dR}+(1-\pi)\mathrm{dU})\cdot\log(\pi\mathrm{dR}+(1-\pi)\mathrm{dU})\\ +(\pi\mathrm{dR}+(1-\pi)\mathrm{dU})\cdot\log(\pi\mathrm{dG}+(1-\pi)\mathrm{dU})\end{array}\right. &(36)\\ &= \pi\cdot\mathrm{KL}(\mathrm{R}\|\mathrm{G})-\mathrm{KL}(\pi\mathrm{dR}+(1-\pi)\mathrm{dU}\|\pi\mathrm{dG}+(1-\pi)\mathrm{dU}), &(37)\end{aligned}$$

as claimed (end of the proof of Lemma D). □

Because of the joint convexity of KL divergence, we always have

$$\begin{aligned}\mathrm{KL}(\pi\mathrm{dR}+(1-\pi)\mathrm{dU}\|\pi\mathrm{dG}+(1-\pi)\mathrm{dU}) &\leqslant \pi\mathrm{KL}(\mathrm{R}\|\mathrm{G})+(1-\pi)\mathrm{KL}(\mathrm{dU}\|\mathrm{dU})\\ &= \pi\mathrm{KL}(\mathrm{R}\|\mathrm{G}),\end{aligned}$$

which yields (another proof) that our loss, $\mathbb{D}_{\ell\textsc{log}}(\mathrm{R},\mathrm{G})$, is lowerbounded by 0. In general, if we let $\gamma > 0$ such that $\mathrm{KL}(\pi\mathrm{dR}+(1-\pi)\mathrm{dU}\|\pi\mathrm{dG}+(1-\pi)\mathrm{dU}) \leqslant (1-\gamma)\cdot\pi\mathrm{KL}(\mathrm{R}\|\mathrm{G})$ (The strict positivity of γ is also a weak assumption as long as R, G sufficiently differ from the uniform distribution), then

$$\mathbb{D}_{\ell\textsc{log}}(\mathrm{R},\mathrm{G}) \geqslant \gamma\pi\cdot\mathrm{KL}(\mathrm{R}\|\mathrm{G}), \tag{38}$$

so any upperbound on $\mathbb{D}_{\ell\textsc{log}}(\mathrm{R},\mathrm{G})$ (such as obtained from Theorem 5.2) translates to an upperbound on the KL divergence. Note that the condition for absolute continuity in Lemma D is always met with the models we learn (both generative forests and ensembles of generative trees).

IV Simpler models: ensembles of generative trees

Storing a GF requires keeping information about the empirical measure R to compute the branching probability in Step 1 of STARUPDATE. This does not require to store the full training sample, but requires at least an index table recording the $\{\text{leaves}\}\times\{\text{observations}\}$ association, for a storage cost in between $\Omega(m)$ and $O(mT)$ where m is the size of the training sample. There is a simple way to get rid of this constraint and approximate the GF by a set of *generative trees* (GTs) of [31].

Models Simply put, a GT is a simplified equivalent representation of a generative forest with 1 tree only. In this case, the branching probabilities in Step 1 of STARUPDATE depend only on the tree's star node position. Thus, instead of recomputing them from scratch each time an observation needs to be generated, we compute them beforehand and use them to label the arcs in addition to the features' domains, as shown in Figure IV.1. This representation is equivalent to that of generative trees [31]. If we have several trees, we do this process independently for each tree and end up with an *ensemble of generative trees* (EOGT). The additional memory footprint for storing probabilities ($O(\sum_i |\Lambda(\Upsilon_i)|)$) is offset by the fact that we do not have anymore to store associations between leaves and observations for generative forests, for a potentially significant reduction is storing size. However, we cannot use anymore STARUPDATE as is for data generation since we cannot compute exactly the probabilities in Step 1. Two questions need to be addressed: is it possible to use an ensemble of generative trees to generate data with good approximation guarantees (and if so, how) and of course how do we train such models.

Data generation We propose a simple solution based on how well an EOGT can approximate a generative forest. It relies on a simple assumption about R and $\mathcal{X}$. Taking as references the parameters in STARUPDATE, we now make two simplifications on notations, first replacing $\Upsilon.\nu^\star_{\mathtt{v}}$ by $\nu^\star_{\mathtt{v}}$ (tree implicit), and then using notation $\mathcal{C}_{\mathtt{v}} \doteq \mathcal{X}_{\nu^\star_{\mathtt{v}}} \cap \mathcal{C}$ for any $\mathtt{v} \in \{\mathtt{t}, \mathtt{f}\}$, the reference to the tree / star node being implicit. The assumption is as follows.

Assumption A. *There exists $\varkappa \in (0, \infty)$ such that at any Step 1 of* STARUPDATE, *for any $\mathtt{v} \in \{\mathtt{t}, \mathtt{f}\}$, we have*

$$\frac{p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{v}} | \mathcal{X}_{\nu^\star_{\mathtt{v}}}\right]}{p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{v}} | \mathcal{X}_{\nu^\star_{\mathtt{v}}}\right]} \quad \in \quad [\exp(-\varkappa), \exp(\varkappa)] \,. \tag{39}$$

A simple condition to ensure the existence of $\varkappa$ is a requirement weaker than **(c)** in Definition 5.1: at all Steps 1 of STARUPDATE, $p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{C}\right] \in (0,1)$, which also guarantees $p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{f}}|\mathcal{C}\right] \in (0,1)$ and thus postulates that the branching in Step 1 of STARUPDATE never reduces to one choice only. The next Lemma shows that it is indeed possible to combine the generation of an ensemble of generative trees with good approximation properties, and provides a simple algorithm to do so, which reduces to running STARUPDATE with a specific approximation to branching probabilities in Step 1. For any GF G and $\mathcal{C} \in \mathcal{P}(\mathrm{G})$, $\mathrm{depth}_{\mathrm{G}}(\mathcal{C})$ is the sum of depths of the leaves in each tree whose support intersection is $\mathcal{C}$. We need the definition of the *expected* depth of G.

Definition B. *The expected depth of* GF G *is* $\overline{\mathrm{depth}}(\mathrm{G}) \doteq \sum_{\mathcal{C}\in\mathcal{P}(\mathrm{G})} p_{\mathrm{G}}[\mathcal{C}] \cdot \mathrm{depth}_{\mathrm{G}}(\mathcal{C})$.

$\overline{\mathrm{depth}}(\mathrm{G})$ represents the expected complexity to sample an observation (see also Lemma C).

Lemma C. *In Step 1 of* STARUPDATE, *suppose $p_{\mathrm{R}}\left[\mathcal{C}_t \cap \mathcal{C}|\mathcal{C}\right]$ is replaced by*

$$\hat{p}_{\nu^\star} \doteq \frac{p_{\mathrm{U}}\left[\mathcal{C}_t|\mathcal{X}_{\nu^\star_t}\right] \cdot p_{\nu^\star}}{p_{\mathrm{U}}\left[\mathcal{C}_t|\mathcal{X}_{\nu^\star_t}\right] \cdot p_{\nu^\star} + p_{\mathrm{U}}\left[\mathcal{C}_f|\mathcal{X}_{\nu^\star_f}\right] \cdot (1 - p_{\nu^\star})}, \tag{40}$$

with $p_{\nu^\star} \doteq p_{\mathrm{R}}\left[\mathcal{X}_{\nu^\star_t}|\mathcal{X}_{\nu^\star}\right]$ the probability labeling arc $(\nu^\star, \nu_t)$ in its generative tree. Under Assumption A, if we denote G the initial GF *and $\hat{\mathrm{G}}$ the* EOGT.P *using* (40)*, the following bound holds on the* KL *divergence between G and $\hat{\mathrm{G}}$:*

$$\mathrm{KL}(\mathrm{G}\|\hat{\mathrm{G}}) \leqslant 2\varkappa \cdot \overline{\mathrm{depth}}(\mathrm{G}), \quad \textit{where } \overline{\mathrm{depth}}(\mathrm{G}) \textit{ is given in Definition B.} \tag{41}$$

Proof. Because $\mathcal{C} \subseteq \mathcal{X}_{\Upsilon.\nu^\star}$ at any call of STARUPDATE, we have

$$\begin{aligned}
&p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}} \cap \mathcal{C}|\mathcal{C}\right] \\
&\quad = p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{C}\right] \\
&\quad = \frac{p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{t}}\right]}{p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{t}}\right] + p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{f}}\right]} && (42)\\
&\quad \leqslant \frac{\exp(\varkappa) p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right]}{\exp(-\varkappa) p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] + \exp(-\varkappa) p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{f}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{f}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{f}}}\right]} && (43)\\
&\quad = \exp(2\varkappa) \cdot \frac{p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}|\mathcal{X}_{\Upsilon.\nu^\star}\right]}{p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{t}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}|\mathcal{X}_{\Upsilon.\nu^\star}\right] + p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{f}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{f}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{f}}}|\mathcal{X}_{\Upsilon.\nu^\star}\right]}. && (44)
\end{aligned}$$

(43) is due to the fact that, since $\mathcal{C} \subseteq \mathcal{X}_{\Upsilon.\nu^\star}$, $p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{v}}\right] = p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{v}} \cap \mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{v}}}\right] = p_{\mathrm{R}}\left[\mathcal{C}_{\mathtt{v}}|\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{v}}}\right] p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{v}}}\right]$, and then using (39). (44) is obtained by dividing numerator and denominator by $p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{t}}}\right] + p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star_{\mathtt{f}}}\right] = p_{\mathrm{R}}\left[\mathcal{X}_{\Upsilon.\nu^\star}\right]$.

G and $\hat{\mathrm{G}}$ satisfy $\mathcal{P}(\mathrm{G}) = \mathcal{P}(\hat{\mathrm{G}})$ so

$$\begin{aligned}
\mathrm{KL}(\mathrm{G}\|\hat{\mathrm{G}}) &= \int \mathrm{d}\mathrm{G} \log \frac{\mathrm{d}\mathrm{G}}{\mathrm{d}\hat{\mathrm{G}}} && (45)\\
&= \sum_{\mathcal{C}\in\mathcal{P}(\mathrm{G})} p_{\mathrm{G}}[\mathcal{C}] \log \frac{p_{\mathrm{G}}[\mathcal{C}]}{p_{\hat{\mathrm{G}}}[\mathcal{C}]}. && (46)
\end{aligned}$$

Finally, $p_{\mathrm{G}}[\mathcal{C}]$ and $p_{\hat{\mathrm{G}}}[\mathcal{C}]$ are just the product of the branching probabilities in any admissible sequence. If we use the same admissible sequence in both generators, we can write $p_{\mathrm{G}}[\mathcal{C}] = \prod_{j=1}^{n(\mathcal{C})} p_j$ and $p_{\hat{\mathrm{G}}}[\mathcal{C}] = \prod_{j=1}^{n(\mathcal{C})} \hat{p}_j$, and (44) directly yields $p_j \leqslant \exp(2\varkappa)\cdot\hat{p}_j, \forall j \in [n(\mathcal{C})]$, $n(\mathcal{C})$ being a shorthand for $\mathrm{depth}_{\mathrm{G}}(\mathcal{C}) = \mathrm{depth}_{\hat{\mathrm{G}}}(\mathcal{C})$ (see main file). So, for any $\mathcal{C} \in \mathcal{P}(\mathrm{G})$,

$$\frac{p_{\mathrm{G}}[\mathcal{C}]}{p_{\hat{\mathrm{G}}}[\mathcal{C}]} \leqslant \exp(2\varkappa \cdot \mathrm{depth}_{\mathrm{G}}(\mathcal{C})), \quad (47)$$

and finally, replacing back $n(\mathcal{C})$ by notation $\mathrm{depth}(\mathcal{C})$,

$$\begin{aligned}
\mathrm{KL}(\mathrm{G}\|\hat{\mathrm{G}}) &\leqslant 2\varkappa \cdot \sum_{\mathcal{C}\in\mathcal{P}(\mathrm{G})} p_{\mathrm{G}}[\mathcal{C}] \cdot \mathrm{depth}(\mathcal{C})\\
&= 2\varkappa \cdot \overline{\mathrm{depth}}(\mathrm{G}), && (48)
\end{aligned}$$

as claimed. ☐

Note the additional leverage for training and generation that stems from Assumption A, not just in terms of space: computing $p_{\mathrm{U}}\left[\mathcal{C}_{\mathtt{v}}|\mathcal{X}_{\nu^\star_{\mathtt{v}}}\right]$ is $O(d)$ and does not necessitate data so the computation of key conditional probabilities (40) drops from $\Omega(m)$ to $O(d)$ for training and generation in an EOGT. Lemma C provides the change in STARUPDATE to generate data.

Training To train an EOGT, we cannot rely on the idea that we can just train a GF and then replace each of its trees by generative trees. To take a concrete example of how this can be a bad idea in the context of missing data imputation, we have observed empirically that a generative forest (GF) can have many trees whose node's observation variables are the *same* within the tree. Taken independently of the forest, such trees would only model marginals, but in a GF, sampling is dependent on the other trees and Lemma 4.4 guarantees the global accuracy of the forest. *However*, if we then replace the GF by an EOGT with the same trees and use them for missing data imputation, this results in imputation at the mode of the marginal(s) (exclusively), which is clearly suboptimal. To avoid this, we have to use a specific training for an EOGT, and to do this, it is enough to change a key part of training in `splitPred`, the computation of probabilities $p_{\mathrm{R}}[.]$ used in (4). Suppose $\varkappa$ very small in Assumption A. To evaluate a split at some leaf, we observe (suppose we have a single tree)

$$p_{\mathrm{R}}[\mathcal{X}_{\lambda_{\mathtt{f}}}] = p_{\mathrm{R}}[\mathcal{X}_{\lambda}] \cdot p_{\mathrm{R}}[\mathcal{X}_{\lambda_{\mathtt{f}}}|\mathcal{X}_{\lambda}] \approx p_{\mathrm{R}}[\mathcal{X}_{\lambda}] \cdot p_{\mathrm{U}}[\mathcal{X}_{\lambda_{\mathtt{f}}}|\mathcal{X}_{\lambda}]$$

(and the same holds for $\lambda_{\mathtt{t}}$). Extending this to multiple trees and any $\mathcal{C} \in \mathcal{P}(\mathcal{T})$, we get the computation of any $p_{\mathrm{R}}[\mathcal{C}_{\mathtt{f}}]$ and $p_{\mathrm{R}}[\mathcal{C}_{\mathtt{t}}]$ needed to measure the new (4) for a potential split. Crucially, it does not necessitate to split the empirical measure at $\mathcal{C}$ but just relies on computing $p_{\mathrm{U}}[\mathcal{C}_{\mathtt{v}}|\mathcal{C}], \mathtt{v} \in \{\mathtt{f}, \mathtt{t}\}$: with the product measure and since we make axis-parallel splits, it can be done in $O(1)$, thus substantially reducing training time.

More with ensembles of generative trees we can follow the exact same algorithmic steps as for generative trees to perform missing data imputation and density estimation, but use branching probabilities in the trees to decide branching, instead of relying on the empirical measure at tuples of nodes, which we do not have access to anymore. For this, we rely on the approximation (40) in Lemma C. A key difference with a GF is that no tuple can have zero density because branching probabilities are in $(0, 1)$ in each generative tree.

Domain	Tag	Source	Missing data ?	m	d	# Cat.	# Num.
`iris`	–	UCI	No	150	5	1	4
*`ringGauss`	`ring`	–	No	1 600	2	–	2
*`circGauss`	`circ`	–	No	2 200	2	–	2
*`gridGauss`	`grid`	–	No	2 500	2	–	2
`forestfires`	`for`	OpenML	No	517	13	2	11
*`randGauss`	`rand`	–	No	3 800	2	–	2
`tictactoe`	`tic`	UCI	No	958	9	9	–
`ionosphere`	`iono`	UCI	No	351	34	2	32
`student_performance_mat`	`stm`	UCI	No	396	33	17	16
`winered`	`wred`	UCI	No	1 599	12	–	12
`student_performance_por`	`stp`	UCI	No	650	33	17	16
`analcatdata_supreme`	`ana`	OpenML	No	4 053	8	1	7
`abalone`	`aba`	UCI	No	4 177	9	1	8
`kc1`	–	OpenML	No	2 110	22	1	21
`winewhite`	`wwhi`	UCI	No	4 898	12	–	12
`sigma-cabs`	–	Kaggle	Yes	5 000	13	5	8
`compas`	`comp`	OpenML	No	5 278	14	9	5
`artificial_characters`	`arti`	OpenML	No	10 218	8	–	8
`jungle_chess`	`jung`	OpenML	No	44 819	8	1	6
`open-policing-hartford`	–	SOP	Yes	18 419	20	16	4
`electricity`	`elec`	OpenML	No	45 312	9	2	7

Table A1: Public domains considered in our experiments (m = total number of examples, d = number of features), ordered in increasing $m \times d$. "Cat." is a shorthand for categorical (nominal / ordinal / binary); "Num." stands for numerical (integers / reals). (*) = simulated, URL for OpenML: `https://www.openml.org/search?type=data&sort=runs&id=504&status=active`; SOP = Stanford Open Policing project, `https://openpolicing.stanford.edu/` (see text). "Tag" refers to tag names used in Tables A7 and A8.

V Appendix on experiments

V.1 Domains

`ringGauss` is the seminal 2D ring Gaussians appearing in numerous GAN papers [46]; those are eight (8) spherical Gaussians with equal covariance, sampling size and centers located on regularly spaced (2-2 angular distance) and at equal distance from the origin. `gridGauss` was generated as a decently hard task from [11]: it consists of 25 2D mixture spherical Gaussians with equal variance and sampled sizes, put on a regular grid. `circGauss` is a Gaussian mode surrounded by a circle, from [46]. `randGauss` is a substantially harder version of `ringGauss` with 16 mixture components, in which covariance, sampling sizes and distances on sightlines from the origin are all random, which creates very substantial discrepancies between modes.

V.2 Algorithms configuration and choice of parameters

GF.BOOST We have implemented GF.BOOST in Java, following Algorithm 3's blueprint. Our implementation of `tree` and `leaf` in Steps 2.1, 2.2 is simple: we pick the heaviest leaf among all trees (with respect to R). The search for the best split is exhaustive unless the variable is categorical with more than a fixed number of distinct modalities (22 in our experiments), above that threshold, we pick the best split among a random subset. We follow [31]'s experimental setting: in particular, the input of our algorithm to train a generator is a .csv file containing the training data *without any further information*. Each feature's domain is learned from the training data only; while this could surely and trivially be replaced by a user-informed domain for improved results (*e.g.* indicating a proportion's domain as $[0\%, 100\%]$, informing the complete list of socio-professional categories, etc.) — and is in fact standard in some ML packages like `weka`'s ARFF files, we did not pick this option to alleviate all side information available to the GT learner. Our software automatically recognizes three types of variables: nominal, integer and floating point represented.

Comments on implementation For the interested reader, we give here some specific implementation details regarding choices mades for two classical bottlenecks on tree-based methods (this also applies to the supervised case of learning decision trees): handling features with large number of modalities and handling continuous features.

We first comment the case where the number of modalities of a feature is large. The details can be found in the code in `File Algorithm.java` (class `Algorithm`), as follows:

- method `public GenerativeModelBasedOnEnsembleOfTrees learn_geot()` implements both GF.Boost (for generative forests) and its extension to training ensembles of generative trees (Appendix) as a single algorithm.
- method `Node choose_leaf(String how_to_choose_leaf)` is the method to choose a leaf (Step 2.2). Since it picks the heaviest leaf among all trees, it also implements Step 2.1 (we initialize all trees to their roots; having a tree = root for generation does not affect generation).
- method `public boolean split_generative_forest(Node leaf, HashSet <MeasuredSupportAtTupleOfNodes> tol)` finds the split for a Generative Forest, given a node chosen for the split. Here is how it works:
  1. It first computes the complete description of possible splits (not their quality yet), using method `public static Vector<FeatureTest> ALL_FEATURE_TESTS(Feature f, Dataset ds)` in `Feature.java`. The method is documented: for continuous features, the list of splits is just a list of evenly spaced splits.
  2. then it calls `public SplitDetails split_top_down_boosting_generative_forest_fast(Node leaf, boolean [] splittable_feature, FeatureTest [][] all_feature_tests, HashSet <MeasuredSupportAtTupleOfNodes> tol)`, which returns the best split as follows: (i) it shuffles the potential list of splits and then (ii) picks the best one in the sublist containing the first `Algorithm.MAXIMAL_NUMBER_OF_SPLIT_TESTS_TRIES_PER_BOOSTING_ITERATION` elements (if the list is smaller, the search for the best is exhaustive); this class variable is currently = 1000.

Last, we comment how we handle continuous variables: the boosting theory tells us that finding a moderately good split (condition (b) Definition 5.1, main file) is enough to get boosting-compliant convergence (Theorem 5.2, main file), so we have settled for a trivial and efficient mechanism: cut-points are evenly spaced and the number is fixed beforehand. See method `public static Vector<FeatureTest> ALL_FEATURE_TESTS(Feature f, Dataset ds) in Feature.java` for the details. It turns out that this works very well and shows our theory was indeed concretely used in the design/implementation of the training algorithm.

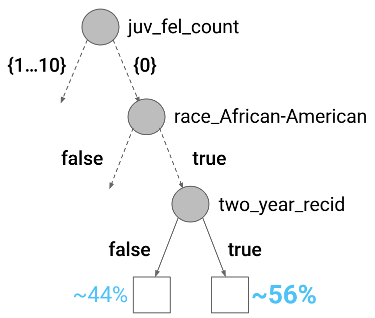
MICE We have used the R MICE package V 3.13.0 with two choices of methods for the round robin (column-wise) prediction of missing values: CART [1] and random forests (RF) [42]. In that last case, we have replaced the default number of trees (10) by a larger number (100) to get better results. We use the default number of round-robin iterations (5). We observed that random forests got the best results so, in order not to multiply the experiments reported and perhaps blur the comparisons with our method, we report only MICE's results for random forests.

TENSORFLOW To learn the additional Random Forests involved in experiments GEN-DISCRIM, we used Tensorflow Decision Forests library[§]. We use 300 trees with max depth 16. Attribute sampling: sqrt(number attributes) for classification problems, number attributes / 3 for regression problems (Breiman rule of thumb); the min #examples per leaf is 5.

CT-GAN We used the Python implementation[¶] with default values [47].

[§]`https://github.com/google/yggdrasil-decision-forests/blob/main/documentation/learners.md`

[¶]`https://github.com/sdv-dev/CTGAN`



```
----------------------------------------------------------------------
(name = #0.0 | depth = 8 | #nodes = 17)
[#0:root] internal {5278} (juv_fel_count <= 0 ? #1 : #2)
|-[#1] internal {5087} (age_cat_Greaterthan45 in {0} ? #3 : #4)
| |-[#3] internal {4004} (age <= 41 ? #5 : #6)
| | |-[#5] internal {3760} (race_African-American in {0} ? #7 : #8)
| | | |-[#7] leaf {1321}
| | | \-[#8] internal {2439} (age <= 19 ? #9 : #10)
| | |   |-[#9] leaf {13}
| | |   \-[#10] internal {2426} (age <= 39 ? #11 : #12)
| | |     |-[#11] internal {2347} (age <= 34 ? #13 : #14)
| | |     | |-[#13] internal {2024} (two_year_recid in {0} ? #15 : #16)
| | |     | | |-[#15] leaf {886}
| | |     | | \-[#16] leaf {1138}
| | |     | \-[#14] leaf {323}
| | |     \-[#12] leaf {79}
| | \-[#6] leaf {244}
| \-[#4] leaf {1083}
\-[#2] leaf {191}
----------------------------------------------------------------------
```

Table A2: SCRUTINIZE: Tree in an example of Generative Forest on sensitive domain `compas`, with 17 nodes and part of its graph sketched following Figure 2's convention, modeling a bias learned from data. Ensemble of such small trees can be *very* accurate: on missing data imputation, they compete with or beat MICE comparatively using 7 000 trees for imputation (see text for details).

Adversarial Random Forests We used the R code of the generator FORGE made available from the paper [44][‖], learning forests containing a variable number of trees in $\{10, 50, 100, 200\}$. We noted that the code does not run when the dataset has missing values and we also got an error when trying to run the code on `kc1`.

Vine Copulas AutoEncoders We used the Python code available from the paper [41][**], which processes only fully numerical datasets. We got a few errors when trying to run the code on `compas`.

Forest Flows We used the R code available from the paper [17][††]. Hyperparameters used were default parameters[‡‡].

Kernel Density Estimation We used the R code available in the package `npudens` with default values following the approach of [24]. We tried several kernels but ended up with sticking to the default choices, that seemed to provide overall some of the best results. Compared to us with GF, KDE's code is extremely compute-intensive as on domains below `tictactoe` in Table A1: it took orders of magnitude more than ours to get the density values on all folds, so we did not run it on the biggest domains. Notice that in our case, computation time includes not just training the generative model, but also, *at each applicable iteration J* in GF.BOOST, the computation of the same number of density values as for KDE.

Computers used We ran part of the experiments on a Mac Book Pro 16 Gb RAM w/ 2 GHz Quad-Core Intel Core i5 processor, and part on a desktop Intel(R) Xeon(R) 3.70GHz with 12 cores and 64 Gb RAM. CT-GANs and VCAEs were run on the desktop, the other algorithms on the laptop.

V.3 Supplementary results

Due to the sheer number of tables to follow, we shall group them according to topics

V.V.3.1 Interpreting our models: 'SCRUTINIZE'

Table A2 provides experimental results on sensitive real world domain `compas`. It shows that it is easy to flag potential bias / fairness issues in data by directly estimating conditional probabilities: considering the GF of this example and ignoring variable `age` for simplicity, conditionally to having 0 felony count and race being African-American, the probability of generating an observation with `two_year_recid = true` is .562 (=1138/2024).

[‖]`https://github.com/bips-hb/arf_paper`

[**]`https://github.com/sdv-dev/Copulas`.

[††]`https://github.com/SamsungSAILMontreal/ForestDiffusion`.

[‡‡]`https://htmlpreview.github.io/?https://github.com/SamsungSAILMontreal/ForestDiffusion/master/R-Package/Vignette.html`.

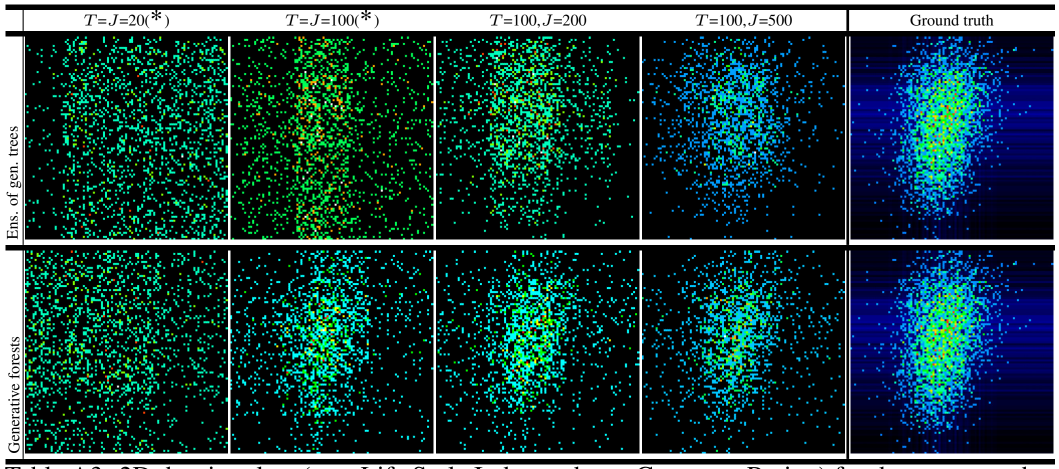


Table A3: 2D density plots (x = Life Style Index and y = Customer Rating) for data generated on `sigma_cabs`, from models learned with 5% missing features (MCAR), using using ensembles of generative trees (top) and generative forests (bottom), for varying total number of trees T and total number of splits J (columns). "*" = all trees are stumps. The rightmost column recalls the domain ground truth for comparison. Each generated dataset contains $m = 2000$ observations.

V.V.3.2 More examples of Table 1 (MF)

Tables A3 completes Tables 2 and A2 (main file), displaying that even a stochastic dependence can be accurately modeled.

We provide in Table A4 the density learned on all our four 2D (for easy plotting) simulated domains and not just `circgauss` as in Table 1 (MF). We see that the observations made in Table 1 (MF) can be generalized to all domains. Even when the results are less different between 50 stumps in a generative forest and 1 generative tree with 50 splits for `ringgauss` and `randgauss`, the difference on `gridgauss` is stark, 1 generative tree merging many of the Gaussians.

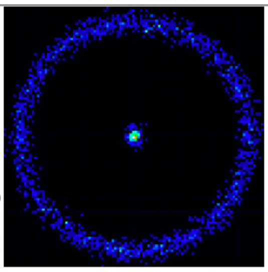
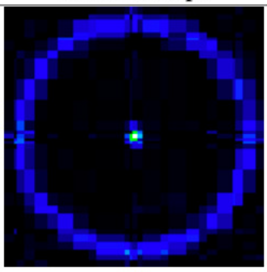
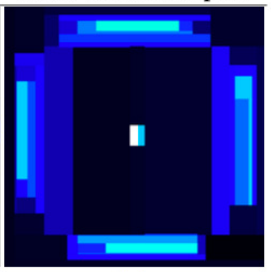
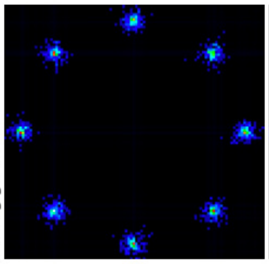
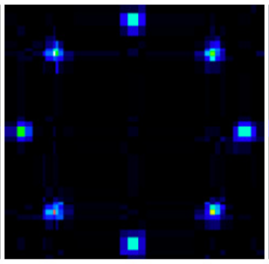
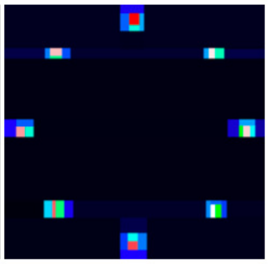
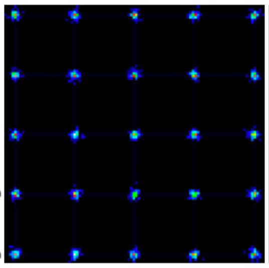
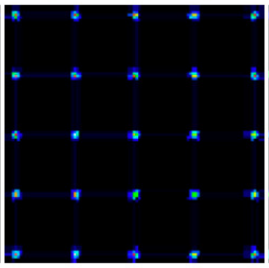
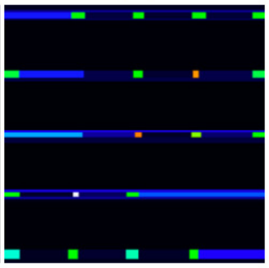
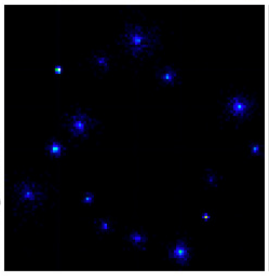
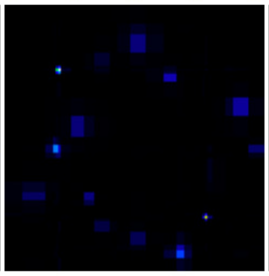
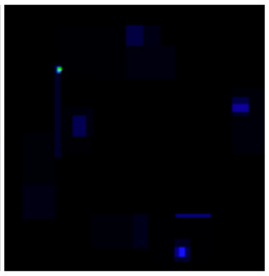
	Ground truth	GF: 50 stumps	GF: 1 tree w/ 50 splits
`circgauss`			
`ringgauss`			
`gridgauss`			
`randgauss`			

Table A4: Comparison of the density learned by a Generative Forest (GF) consisting of a single tree learned by GF.BOOST with n total splits (*right*) and a set of n GF stumps (1 split per tree) learned by GF.BOOST (*center*). The left picture is the domain, represented using the same heatmap. On each domain, 5% of the data is missing (MCAR = missing completely at random).

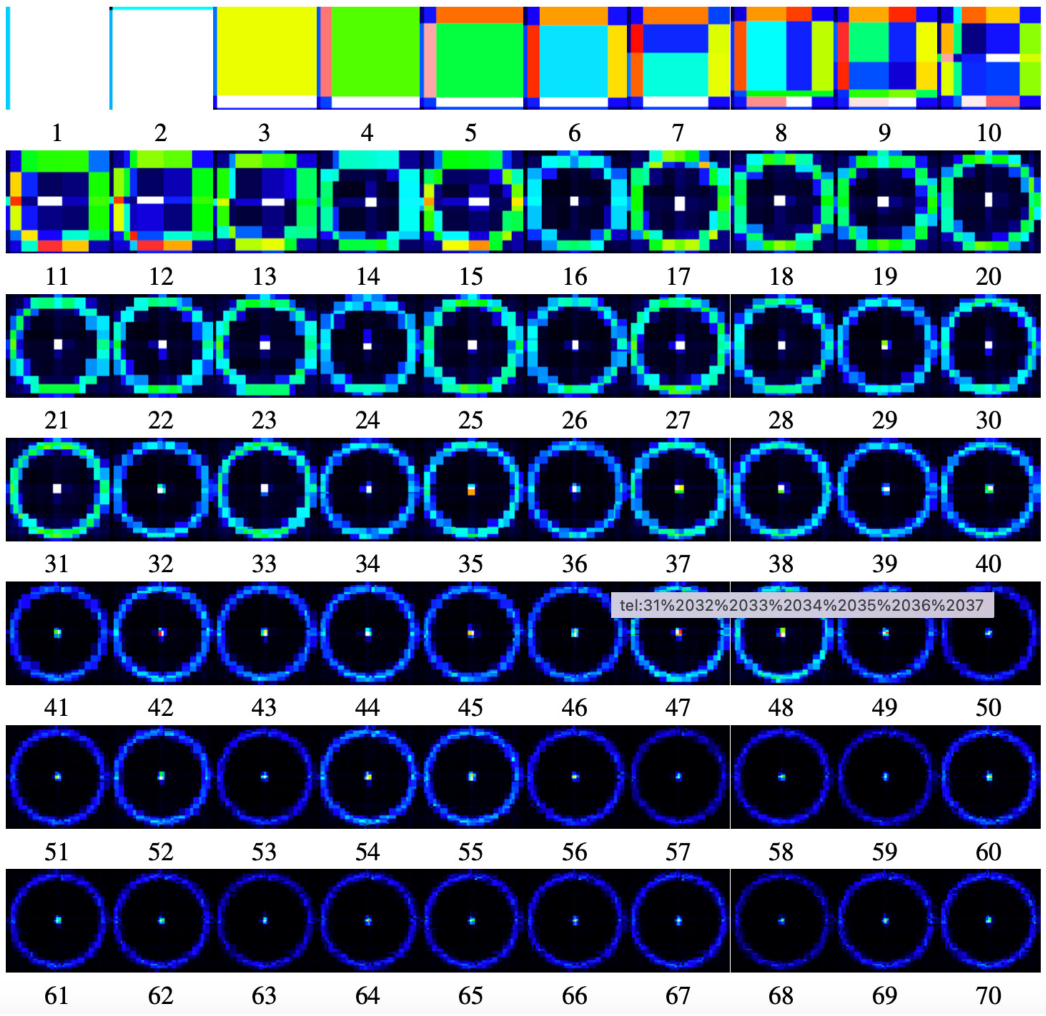


Table A5: Densities learned on `circgauss` by GF.BOOST with generative forests consisting of stumps, for values of $T = J$ in $\{1, 2, ...70\}$. The models quickly capture the ring and the middle dense Gaussian. The boxy-shape of the densities, due to the axis-parallel splits, get less noticeable as $T = J$ exceeds a few dozen.

V.V.3.3 The generative forest of Table 1 (MF) developed further

One might wonder how a set of stumps gets to accurately fit the domain as the number of stumps increases. Table A5 provides an answer. From this experiment, we can see that 16 stumps are enough to get the center mode. The ring shape takes obviously more iterations to represent but still, is takes a mere few dozen stumps to clearly get an accurate shape, the last iterations just finessing the fitting.

V.V.3.4 Experiment LIFELIKE *in extenso*

Domain	Sinkhorn↓			Coverage↑			Density↑			F1 measure↓		
	us (GF)	ARF	pval	us (GF)	ARF	pval	us (GF)	ARF	pval	us (GF)	ARF	pval
`ring`	*0.283±0.005	0.286±0.007	0.74	*0.964±0.022	0.9600±0.008	0.53	*0.999±0.055	0.976±0.030	0.42	*0.060±0.047	0.086±0.047	0.23
`circ`	0.356±0.002	*0.355±0.005	0.74	0.954±0.014	*0.962±0.021	0.27	*0.968±0.027	0.954±0.011	0.04	0.520±0.021	*0.507±0.032	0.67
`grid`	*0.392±0.002	0.394±0.002	0.34	*0.964±0.006	0.908±0.010	▲	*0.953±0.062	0.630±0.050	▲	*0.034±0.010	0.043±0.012	0.26
`rand`	0.290±0.007	*0.288±0.002	0.65	0.954±0.010	*0.953±0.004	0.94	*0.936±0.029	0.940±0.027	0.74	*0.020±0.005	0.029±0.010	0.14
`wred`	*1.027±0.037	1.099±0.031	▲	*0.954±0.013	0.929±0.010	0.04	*0.874±0.042	0.801±0.028	▲	*0.503±0.023	0.531±0.028	0.01
`wwhi`	*1.120±0.013	1.150±0.021	0.03	*0.952±0.008	0.946±0.012	0.22	0.940±0.019	*0.941±0.013	0.88	0.510±0.027	*0.500±0.027	0.09
`comp`	0.538±0.015	*0.535±0.033	0.90	*0.546±0.026	0.560±0.037	0.10	*0.424±0.014	0.440±0.011	0.03	*0.500±0.025	0.520±0.020	0.33
`arti`	*0.830±0.007	0.849±0.010	▲	*0.932±0.018	0.892±0.014	▲	*0.897±0.004	0.747±0.013	▲	*0.457±0.057	0.512±0.037	0.02
	us (GF)	CT	pval	us (GF)	CT	pval	us (GF)	CT	pval	us (GF)	CT	pval
`ring`	*0.283±0.005	0.351±0.042	0.02	*0.964±0.022	0.798±0.05	▲	*0.999±0.055	0.757±0.025	▲	*0.060±0.047	0.100±0.073	0.75
`circ`	*0.356±0.002	0.435±0.075	0.08	*0.954±0.014	0.734±0.041	▲	*0.968±0.027	0.401±0.033	▲	*0.520±0.021	0.746±0.033	▲
`grid`	*0.392±0.002	0.408±0.019	0.12	*0.964±0.006	0.828±0.053	▲	*0.953±0.062	0.649±0.058	▲	0.034±0.010	0.034±0.011	0.98
`rand`	*0.290±0.007	0.327±0.024	0.01	*0.954±0.010	0.659±0.049	▲	*0.936±0.029	0.582±0.035	▲	*0.020±0.005	0.079±0.053	0.08
`wred`	*1.027±0.037	1.384±0.047	▲	*0.954±0.013	0.808±0.016	▲	*0.874±0.042	0.589±0.129	▲	*0.503±0.023	0.654±0.030	▲
`wwhi`	*1.120±0.013	1.158±0.009	0.01	*0.952±0.008	0.894±0.026	▲	0.940±0.019	*0.953±0.035	0.50	*0.510±0.027	0.581±0.043	▲
	us (GF)	FF	pval	us (GF)	FF	pval	us (GF)	FF	pval	us (GF)	FF	pval
`ring`	0.283±0.005	*0.276±0.001	0.09	*0.964±0.022	0.957±0.020	0.31	0.999±0.055	*1.045±0.020	0.07	0.060±0.047	*0.051±0.031	0.75
`circ`	0.356±0.003	*0.354±0.003	0.30	0.954±0.014	*0.956±0.015	0.74	0.968±0.027	*0.989±0.027	0.06	*0.520±0.021	0.530±0.028	0.55
`grid`	*0.392±0.002	0.393±0.001	0.53	*0.964±0.006	0.954±0.013	0.07	0.953±0.062	*1.013±0.050	0.01	*0.034±0.010	0.045±0.007	0.02
`rand`	0.290±0.007	*0.288±0.001	0.55	*0.954±0.010	0.927±0.011	0.04	0.936±0.029	*1.000±0.008	0.01	*0.020±0.005	0.028±0.008	0.11
`wred`	*1.027±0.037	1.030±0.029	0.84	0.954±0.013	*0.955±0.018	0.95	0.874±0.042	*1.009±0.034	▼	0.503±0.023	*0.458±0.052	0.13
`wwhi`	1.120±0.013	*1.097±0.007	0.05	*0.952±0.008	0.945±0.007	0.18	0.940±0.019	*0.970±0.023	0.03	0.510±0.027	*0.498±0.041	0.48
`comp`	*0.537±0.015	0.891±0.007	▲	0.546±0.026	*0.548±0.027	0.74	*0.424±0.014	0.404±0.013	0.03	*0.500±0.025	0.510±0.032	0.36
`arti`	*0.829±0.006	0.834±0.016	0.51	*0.932±0.018	0.879±0.008	▲	*0.897±0.004	0.774±0.016	▲	*0.457±0.057	0.530±0.032	▲
	us (GF)	VC-G	pval	us (GF)	VC-G	pval	us (GF)	VC-G	pval	us (GF)	VC-G	pval
`ring`	*0.283±0.005	0.392±0.005	▲	*0.964±0.022	0.364±0.037	▲	*0.999±0.055	0.132±0.013	▲	*0.060±0.047	0.291±0.035	▲
`circ`	*0.356±0.002	0.415±0.012	▲	*0.954±0.014	0.620±0.020	▲	*0.968±0.027	0.265±0.015	▲	*0.520±0.021	0.805±0.012	▲
`grid`	*0.392±0.002	0.400±0.003	▲	*0.964±0.006	0.165±0.037	▲	*0.953±0.062	0.042±0.012	▲	*0.034±0.010	0.065±0.002	▲
`rand`	*0.290±0.007	0.414±0.003	▲	*0.954±0.010	0.317±0.033	▲	*0.936±0.029	0.156±0.018	▲	*0.020±0.005	0.224±0.019	▲
`wred`	*1.027±0.037	1.105±0.042	0.01	*0.954±0.013	0.913±0.013	0.02	*0.874±0.042	0.821±0.041	0.04	*0.503±0.023	0.537±0.007	0.03
`wwhi`	*1.120±0.013	1.181±0.019	▲	*0.952±0.008	0.938±0.008	0.06	*0.940±0.019	0.907±0.026	0.01	*0.510±0.027	0.527±0.028	0.03
	us (GF)	VC-C	pval	us (GF)	VC-C	pval	us (GF)	VC-C	pval	us (GF)	VC-C	pval
`ring`	*0.283±0.005	0.394±0.016	▲	*0.964±0.022	0.346±0.042	▲	*0.999±0.055	0.126±0.020	▲	*0.060±0.047	0.326±0.015	▲
`circ`	*0.356±0.002	0.409±0.005	▲	*0.954±0.014	0.638±0.033	▲	*0.968±0.027	0.275±0.018	▲	*0.520±0.021	0.800±0.018	▲
`grid`	*0.392±0.002	0.399±0.004	0.03	*0.964±0.006	0.169±0.013	▲	*0.953±0.062	0.045±0.004	▲	*0.034±0.010	0.065±0.002	▲
`rand`	*0.290±0.007	0.415±0.005	▲	*0.954±0.010	0.321±0.034	▲	*0.936±0.029	0.152±0.014	▲	*0.020±0.005	0.226±0.015	▲
`wred`	*1.027±0.037	1.544±0.024	▲	*0.954±0.013	0.600±0.052	▲	*0.874±0.042	0.909±0.053	0.18	*0.503±0.023	0.793±0.040	▲
`wwhi`	*1.120±0.013	1.437±0.048	▲	*0.952±0.008	0.572±0.017	▲	*0.940±0.019	0.920±0.045	0.41	*0.510±0.027	0.837±0.010	▲
	us (GF)	VC-D	pval	us (GF)	VC-D	pval	us (GF)	VC-D	pval	us (GF)	VC-D	pval
`ring`	*0.283±0.005	0.390±0.011	▲	*0.964±0.022	0.331±0.067	▲	*0.999±0.055	0.122±0.028	▲	*0.060±0.047	0.319±0.037	▲
`circ`	*0.356±0.002	0.411±0.004	▲	*0.954±0.014	0.649±0.055	▲	*0.968±0.027	0.269±0.026	▲	*0.520±0.021	0.813±0.018	▲
`grid`	*0.392±0.002	0.398±0.002	▲	*0.964±0.006	0.162±0.034	▲	*0.953±0.062	0.043±0.009	▲	*0.034±0.010	0.064±0.001	▲
`rand`	*0.290±0.007	0.414±0.003	▲	*0.954±0.010	0.312±0.040	▲	*0.936±0.029	0.149±0.018	▲	*0.020±0.005	0.225±0.017	▲
`wred`	*1.027±0.037	1.383±0.037	▲	*0.954±0.013	0.868±0.042	0.02	*0.874±0.042	0.738±0.030	▲	*0.503±0.023	0.587±0.019	▲
`wwhi`	*1.120±0.013	1.484±0.056	▲	*0.952±0.008	0.876±0.027	▲	*0.940±0.019	0.891±0.010	▲	*0.510±0.027	0.608±0.037	▲
	us (GF)	VC-R	pval	us (GF)	VC-R	pval	us (GF)	VC-R	pval	us (GF)	VC-R	pval
`ring`	*0.283±0.005	0.388±0.004	▲	*0.964±0.022	0.331±0.065	▲	*0.999±0.055	0.124±0.022	▲	*0.060±0.047	0.322±0.023	▲
`circ`	*0.356±0.002	0.402±0.003	▲	*0.954±0.014	0.664±0.017	▲	*0.968±0.027	0.274±0.006	▲	*0.520±0.021	0.806±0.022	▲
`grid`	*0.392±0.002	0.399±0.003	0.02	*0.964±0.006	0.153±0.032	▲	*0.953±0.062	0.038±0.006	▲	*0.034±0.010	0.065±0.001	▲
`rand`	*0.290±0.007	0.417±0.009	▲	*0.954±0.010	0.315±0.023	▲	*0.936±0.029	0.145±0.021	▲	*0.020±0.005	0.229±0.022	▲
`wred`	*1.027±0.037	1.365±0.098	▲	*0.954±0.013	0.839±0.027	▲	*0.874±0.042	0.716±0.072	0.01	*0.503±0.023	0.596±0.047	0.02
`wwhi`	*1.120±0.013	1.353±0.038	▲	*0.952±0.008	0.747±0.014	▲	*0.940±0.019	0.942±0.029	0.82	*0.510±0.027	0.729±0.024	▲

Table A6: LIFELIKE: comparison of Generative Forests (us, GF), using smaller generative forest models (T = 200 trees, J =500 total splits) to the same contenders as in Table 5 (main file, we also add the option Center for VCAE; we did not push it in the main file for space constraints but its results are on par with the other VCAE results). Conventions are the same as in Table 5. See text for details.

The objective of the experiment is to evaluate whether a generative model is able to create "realistic" data. Tabular data is hard to evaluate visually, unlike *e,g.* images or text, so we have considered a simple evaluation pipeline: we create for each domain a 5-fold stratified experiment. After a generative model has been trained, we generate the same number of observations as in the test fold and compute the optimal transport (OT) distance between the generated sample and the fold's test sample. To fasten the computation of OT costs, we use Sinkhorn's algorithm [8] with an ε-entropic regularizer, for some $\varepsilon = 0.5$ which we observed was the smallest in our experiments to run with all domains without leading to numerical instabilities. To balance the importance of categorical features (for which the cost is binary, depending on whether the guess is right or not) and numerical features, we normalize numerical features with the domain's mean and standard deviation prior to computing OT costs. We then compare our method to three possible contenders: Adversarial Random Forests (ARF) [44], CT-GANs [47], Forest Flows [17] and Vine copula autoencoders (VCAE) [41]. All these approaches rely on models that are very different from each other. Adversarial Random Forests (ARF) and Forest Flows (FFs) are tree-based generators. ARFs work as follows to generate one observation: one first sample uniformly at random a tree, then samples a leaf in the tree. The leaf is attached to a distribution which is then used to sample the observation. As we already pointed out in Section 2, there are two main differences with our models. From the standpoint of the model's distribution, if one takes the (non-empty) support from a tuple of leaves, its probability is obtained from a weighted average of the tree's distributions in [44] while it is obtained from a product of theirs in our case. Hence, at similar tree sizes, the modelling capability of the set of trees tips in our favor, but it is counterbalanced by the fact that their approach equip leaves with potentially complex distributions (e.g. truncated Gaussians) whereas we stick to uniform distributions at the leaves. A consequence of the modeling of ARFs is that each tree has to separately code for a good generator: hence, it has to be big enough or the distributions used at the leaves have to be complex enough. In our case, as we already pointed out, a small number of trees (sometimes even *stumps*) can be enough to get a fairly good generator, even on real-world domains. Forest Flows use tree-based models to estimate parameters of diffusion models. Their tree-based models are not necessarily big but they need a lot of them to carry out training and generation, which is a big difference with us. Our next contender, CT-GAN [47], fully relies on neural networks so it is *de facto* substantially different from ours. The last, VCAE [41] relies on a subtle combination of deep nets and graphical models learned in the latent space.

Table 5 (main file) with smaller generative forests Table A6 provides the equivalent of Table 5 (with one additional contender, which did not fit in the space allotted for the main file), but in which our generative forests are much smaller: T =200 trees, j =500 total splits (so each tree is on average barely bigger than a decision stump). One can check that our models are still competitive with respect to all contenders. The best contender, Forest Flows, beat our models on density, but it must be remembered that Forest Flows uses many tree-based models to carry out both training and generation.

Focus on Sinkhorn for different parameterizations of contenders We now investigate how changing the parameterization of various contenders affect their performance against our approach. Here, ARFs are trained with a number of trees $T \in \{10, 50, 100, 200\}$ (ARFs include an algorithm to select the tree size so we do not have to select it). CT-GANs are trained with a number of epochs $E \in \{10, 100, 300, 1000\}$.

We compare those models with generative forests with $T = 500$ trees and trained for a total of $J = 2000$ iterations, for all domains considered. Compared to the trees learned by ARFs, the total number of splits we use can still be small compared to theirs, in particular when they learn $T \geqslant 100$ trees. Each table against ARFs and CT-GANs has a different size and contains only a subset of the whole 21 domains: ARFs do not process domains with missing values and we also got issues running contenders on several domains: those are discussed in Appendix, Section V.2.

domain	us (GF)	Adversarial Random Forests (ARFs)							
tag	T=500, J=2 000	$T = 10$	p-val	$T = 50$	p-val	$T = 100$	p-val	$T = 200$	p-val
iris	0.423±0.033	0.530±0.064	0.0039	0.517±0.081	0.0866	0.466±0.050	0.1452	0.502±0.043	0.0044
ring	0.285±0.008	0.289±0.006	0.1918	0.286±0.007	0.8542	0.288±0.010	0.7205	0.286±0.007	0.6213
circ	0.351±0.005	0.354±0.002	0.2942	0.356±0.008	0.2063	0.350±0.002	0.9050	0.355±0.005	0.3304
grid	0.390±0.002	0.394±0.003	0.1376	0.391±0.002	0.3210	0.392±0.001	0.2118	0.394±0.002	0.0252
for	1.108±0.105	1.272±0.281	0.2807	1.431±0.337	0.1215	1.311±0.255	0.0972	1.268±0.209	0.0616
rand	0.286±0.003	0.286±0.003	0.9468	0.290±0.005	0.2359	0.289±0.005	0.3990	0.288±0.002	0.3685
tic	0.575±0.002	0.577±0.001	0.2133	0.578±0.001	0.1104	0.577±0.001	0.2739	0.577±0.001	0.2773
iono	1.167±0.074	1.431±0.043	0.0005	1.469±0.079	0.0001	1.431±0.058	0.0015	1.420±0.056	0.0035
stm	0.975±0.026	1.010±0.015	0.0304	1.015±0.018	0.0231	1.001±0.010	0.0956	1.014±0.021	0.0261
wred	0.980±0.032	1.086±0.036	0.0003	1.072±0.055	0.0133	1.086±0.030	0.0003	1.099±0.031	0.0001
stp	0.976±0.018	0.999±0.011	0.0382	1.012±0.010	0.0250	1.006±0.011	0.0280	1.003±0.003	0.0277
ana	0.368±0.014	0.370±0.012	0.7951	0.382±0.015	0.1822	0.368±0.006	0.9161	0.369±0.007	0.8316
aba	0.490±0.028	0.505±0.046	0.5251	0.484±0.024	0.7011	0.503±0.035	0.6757	0.481±0.031	0.4632
wwhi	1.064±0.003	1.159±0.011	ε	1.159±0.006	ε	1.170±0.014	ε	1.150±0.021	0.0005
comp	0.532±0.008	0.531±0.012	0.8278	0.534±0.009	0.8438	0.551±0.023	0.1371	0.535±0.033	0.8642
arti	0.821±0.004	0.849±0.004	0.0007	0.847±0.010	0.0005	0.843±0.009	0.0006	0.849±0.010	0.0005
jung	0.929±0.002	0.929±0.002	0.6044	0.930±0.002	0.6536	0.929±0.002	0.5937	0.930±0.001	0.2447
elec	1.483±0.033	1.577±0.088	0.1052	1.543±0.017	0.0081	1.552±0.039	0.0522	1.581±0.025	0.0095
	wins / lose for us →	15 / 1		17 / 1		16 / 1		17 / 1	

Table A7: LIFELIKE: comparison of ARFs [44] with different number T of trees, and meduim-sized Generative Forests (us, GF) where the number of trees and number of iterations are **fixed** ($T = 500$ trees, total of $J = 2000$ splits in trees). Values shown = average over the 5-folds $\pm$ std dev. . The p-values for the comparison of our results ("us") and ARFs are shown. Values appearing in green in p-vals mean (i) the average regularized OT cost for us (GF) is smaller than that of ARFs and (ii) $p < 0.1$, *i.e.* our method outperforms ARFs (there is no instance of us being statistically significantly beaten by ARFs). ε means p-val $< 10^{-4}$. Domain details in Table A1.

Results vs Adversarial Random Forests Table A7 digs into results obtained against adversarial random forests on the Sinkhorn metric. A first observation, not visible in the table, is that ARFs indeed tend to learn big models, typically with dozens of nodes in each tree. For the largest ARFs with 100 or 200 trees, this means in general a total of thousands of nodes in models. A consequence, also observed experimentally, is that there is sometimes little difference in performance in general between models with a different number of trees in ARFs as each tree is in fact an already good generator. In our case, this is obviously not the case. Noticeably, the performance of ARFs is not monotonic in the number of trees, so there could be a way to find the appropriate number of trees in ARFs to buy the slight (but sometimes significant) increase in performance in a domain-dependent way. In our case, there is obviously a dependency in the number of trees chosen as well. From the standpoint of the optimal transport metric, even when ARFs make use of distributions at their tree leaves that are much more powerful than ours and in fact fit to some of our simulated domains (Gaussians used in `circgauss`, `randgauss` and `gridgauss`), we still manage to compete or beat ARFs on those simulated domains.

Globally, we manage to beat ARFs on almost all runs, and very significantly on many of them. Since training generative forests does not include a mechanism to select the size of models, we have completed this experiment by another one on which we learn much smaller generative forests. The corresponding Table is in Appendix, Section V.V.3.4. Because we clearly beat ARFs on `student_performance_mat` and `student_performance_por` in Table A7 but are beaten by ARFs for much smaller generative forests, we conclude that there could exist mechanisms to compute the "right size" of our models, or to prune big models to get models with the right size. Globally however, even with such smaller models, we still manage to beat ARFs on a large majority of cases, which is a good figure given the difference in model sizes. The ratio "results quality over model size" tips in our favor and demonstrates the potential in using all trees in a forest to generate an observation (us) vs using a single of its trees (ARFs), see Figure 1.

Results vs CT-GANs Table A8 summarizes our results against CT-GAN using various numbers of training epochs (E). In our case, our setting is the same as versus ARFs: we stick to $T = 500$ trees trained for a total number of $J = 2000$ iterations in the generative forest, for all domains. We see that generative forests consistently and significantly outperform CT-GAN on nearly all cases. Furthermore, while CT-GANs performances tend to improve with the number of epochs, we observe that on a majority of datasets, performances at 1 000 epochs are still far from those of generative forests and already induce training times that are far bigger than for generative forests: for example, more than 6 hours per fold for `stm` while it takes just a few seconds to train a generative forest. Just like we did for ARFs, we repeated the experiment vs CT-GAN using much smaller generative forests ($T = 200, J = 500$). The results are presented in Appendix, Section V.V.3.4. There is no change in the conclusion: even with such small models, we still beat CT-GANs, regardless of the number of training epochs, on all cases (and very significantly on almost all of them).

More results with small generative forests Tables A9 and A10 present the results of generative forests vs adversarial random forests and CT-GANs, when the size of our models is substantially smaller than in the main file ($T = 200, J = 500$). We still manage to compete or beat ARFs on simulated domains for which modelling the leaf distributions in ARFs should represent an advantage (Gaussians used in `circgauss`, `randgauss` and `gridgauss`), even more using comparatively very small models compared to ARFs. We believe this could be due to two factors: (i) the fact that in our models all trees are used to generate each observation (which surely facilitates learning small and accurate models) and (ii) the fact that we do not generate data during training but base our splitting criterion on an *exact* computation of the reduction of Bayes risk in growing trees. If we compute aggregated statistics comparing, for each number of trees in ARFs, the number of times we win / lose versus ARFs, either considering only significant p-values or disregarding them, our generative forests always beat ARFs on a majority of domains. Globally, this means we win on 34 out of 52 cases. The results vs CT-GANs are of the same look and feel as in the main file with larger generative forests: generative forests beat them in all cases, and very significantly in most of them.

V.V.3.5 Comparison with "the optimal generator": GEN-DISCRIM

Rather than compare our method with another one, the question we ask here is "can we generate data that looks like domain's data" ? We replicate the experimental pipeline of [31]. In short, we shuffle a 3-partition (say $\mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3$) of the training data in a $3! = 6$-fold CV, then train a generator G_1

domain	us (GF)	CT-GANs							
tag	T=500, J=2 000	$E = 10$	p-val	$E = 100$	p-val	$E = 300$	p-val	$E = 1000$	p-val
`ring`	0.285±0.008	0.457±0.058	0.0032	0.546±0.079	0.0018	0.405±0.044	0.0049	0.351±0.042	0.0183
`circ`	0.351±0.005	0.848±0.300	0.0213	0.480±0.040	0.0019	0.443±0.014	ε	0.435±0.075	0.0711
`grid`	0.390±0.002	0.749±0.417	0.1268	0.459±0.031	0.0085	0.426±0.022	0.0271	0.408±0.019	0.1000
`for`	1.108±0.105	9.796±5.454	0.0049	1.899±0.289	0.0045	1.532±0.178	0.0244	1.520±0.311	0.0410
`rand`	0.286±0.003	0.746±0.236	0.0119	0.491±0.063	0.0021	0.368±0.015	0.0002	0.327±0.024	0.0288
`tic`	0.575±0.002	0.601±0.003	0.0002	0.581±0.001	0.0082	0.586±0.006	0.0294	0.584±0.003	0.0198
`iono`	1.167±0.074	2.263±0.035	ε	2.084±0.052	ε	1.984±0.136	ε	1.758±0.137	0.0002
`stm`	0.980±0.032	1.511±0.074	ε	1.189±0.045	0.0002	1.168±0.054	0.0056	1.167±0.041	0.0013
`wred`	0.980±0.032	2.836±0.703	0.0044	2.004±0.090	ε	1.825±0.127	0.0001	1.384±0.047	ε
`stp`	0.976±0.018	1.660±0.161	0.0006	1.141±0.018	0.0001	1.206±0.046	0.0007	1.186±0.052	0.0019
`ana`	0.368±0.014	0.542±0.135	0.0050	0.439±0.056	0.0202	0.404±0.012	0.0397	0.436±0.036	0.0051
`aba`	0.490±0.028	1.689±0.053	ε	1.463±0.094	ε	0.754±0.040	0.0009	0.657±0.026	0.0006
`wwhi`	1.064±0.003	1.770±0.160	0.0005	1.849±0.064	ε	1.284±0.029	ε	1.158±0.009	ε
`arti`*	0.821±0.004	1.388±0.109$_4$	0.0020	1.209±0.018$_4$	0.0005	1.026±0.013$_4$	0.0001	0.959±0.013$_3$	0.0038
wins / lose for us →		14 / 0		14 / 0		14 / 0		14 / 0	

Table A8: Comparison of results with respect to CT-GANs [47] trained with different number of epochs E. On some domains, indicated with a "*", CT-GANs crashed on some folds. For such domains, we indicate in index to CT-GANs results the number of folds (out of 5) for which this did *not* happen. In such cases, we restricted the statistical comparisons with us to the folds for which CT-GANs did not crash: the statistical tests take this into account; instead of providing all corresponding average performances for us, we keep giving the average performance for us on all five folds (leftmost column), which is thus indicative. Other conventions follow Table A7.

from R_1 (we use generative forests with GF.BOOST), then train a random forest (RF) to distinguish between G_1 and R_2, and finally estimate its test accuracy on G_1 vs R_3. The *lower* this accuracy, the less distinguishable are fakes from real and thus the *better* the generator. We consider 3 competing baselines to our models: (i) the "optimal" one, COPY, which consists in replacing G_1 by a set of real data, (ii) the "worst" one, UNIF(orm), which uniformly samples data, and (iii) GF.BOOST for $T = 1$, which learns a generative tree (GT). We note that this pipeline is radically different from the one used in [44]. In this pipeline, domains used are supervised (the data includes a variable to predict). A generator is trained from some training data, then generates an amount of data equal to the size of the training data. Then, two classifiers are trained, one from the original training data, one from the generated data, to predict for the variable to predict and their performances are compared on this basis (the higher the accuracy, the better). There is a risk in this pipeline that we have tried to mitigate with our sophisticated pipeline: if the generator consists just in copying its training data, it is guaranteed to perform well according to [44]'s metric. Obviously, this would not be a good generator however. Our pipeline would typically prevent this, R_2 being different from R_3.

Results Tables A11 and A12 provide the results we got, including statistical p-values for the test of comparing our method to COPY. They display that it is possible to get GFs generating realistically looking data: for `iris`, the Student t-tests show that we can keep H_0 = "GFs perform identically to COPY" for $J = T = 40$ ($p > .2$). For `mat`, the result is quite remarkable not because of the highest p-val, which is not negligible (> 0.001), but because to get it, it suffices to build a GF in which the total number of feature occurrences ($J = 80$) is barely three times as big as the domain's dimension ($d = 33$). The full tables display a clear incentive to grow further the GFs as domains increase in size. However, our results show, pretty much like the experiment against Adversarial Random Forests, that there could be substantial value of learning the right model size: `iris` and `student_performance_por` clearly show nontrivial peaks of p-values, for which we would get the best models compared to COPY.

domain	us (GF)	Adversarial Random Forests (ARFs)							
tag	T=200, J=500	$T = 10$	p-val	$T = 50$	p-val	$T = 100$	p-val	$T = 200$	p-val
`iris`	0.529±0.099	0.530±0.064	0.9890	0.517±0.081	0.8645	0.466±0.050	0.1099	0.501±0.043	0.6548
`ring`	0.283±0.005	0.289±0.006	0.0651	0.286±0.007	0.4903	0.288±0.010	0.4292	0.286±0.007	0.7442
`circ`	0.356±0.002	0.354±0.002	0.4561	0.356±0.008	0.7882	0.350±0.002	0.0591	0.355±0.005	0.7446
`grid`	0.392±0.002	0.394±0.003	0.4108	0.391±0.002	0.3403	0.392±0.001	0.8370	0.394±0.002	0.3423
`for`	1.086±0.124	1.272±0.281	0.1420	1.431±0.337	0.1142	1.311±0.255	0.1215	1.268±0.209	0.0370
`rand`	0.290±0.007	0.286±0.003	0.4488	0.290±0.005	0.9553	0.289±0.005	0.9004	0.288±0.002	0.6542
`tic`	0.574±0.001	0.577±0.001	0.0394	0.578±0.001	0.0325	0.577±0.001	0.0657	0.577±0.001	0.0914
`iono`	1.203±0.068	1.431±0.043	0.7307	1.469±0.079	0.3402	1.431±0.058	0.7046	1.420±0.056	0.8737
`stm`	1.044±0.013	1.010±0.015	0.0080	1.015±0.018	0.0094	1.001±0.010	0.0037	1.014±0.021	0.0329
`wred`	1.027±0.037	1.086±0.036	0.0327	1.072±0.055	0.1432	1.086±0.030	0.0095	1.099±0.031	0.0044
`stp`	1.029±0.027	0.999±0.011	0.0826	1.012±0.010	0.3421	1.006±0.011	0.0542	1.003±0.003	0.0764
`ana`	0.367±0.018	0.370±0.012	0.8229	0.382±0.015	0.2318	0.368±0.006	0.9264	0.369±0.007	0.8722
`aba`	0.476±0.017	0.505±0.046	0.2264	0.484±0.024	0.2162	0.503±0.035	0.1847	0.481±0.031	0.7468
`wwhi`	1.120±0.013	1.159±0.011	0.0013	1.159±0.006	0.0014	1.170±0.014	0.0026	1.150±0.021	0.0327
`comp`	0.538±0.015	0.531±0.012	0.5639	0.534±0.009	0.5706	0.551±0.023	0.4057	0.535±0.033	0.9017
`arti`	0.830±0.007	0.849±0.004	0.0078	0.847±0.010	0.0012	0.843±0.009	0.0003	0.849±0.010	0.0010
`jung`	0.928±0.001	0.929±0.002	0.1720	0.930±0.002	0.0646	0.929±0.002	0.2675	0.930±0.001	0.0081
`elec`	1.503±0.035	1.577±0.088	0.1222	1.543±0.017	0.0880	1.552±0.039	0.1228	1.581±0.025	0.0027
	wins / lose for us →	13 / 5		11 / 5		12 / 5		12 / 6	

Table A9: Comparison of results for ARFs [44] with different number J of trees, and small Generative Forests (us, GF) where the number of trees and number of iterations are **fixed** ($T = 200$ trees, total of $J = 500$ splits in trees). Values are shown on the form average over the 5-folds ± standard deviation. The p-values for the comparison of our results ("us") and ARFs are shown. Values appearing in green in p-vals mean (i) the average regularised OT cost for us (GF) is smaller than that of ARFs and (ii) $p < 0.1$, meaning we can conclude that our method outperforms ARFs. Values appearing in magenta in p-vals mean (i) the average regularised OT cost for us (GF) is larger than that of ARFs and (ii) $p < 0.1$, meaning we can conclude that our method is outperformed by ARFs. The last row summarizes the number of times we win / lose against ARFs according to the average metric used. To reduce size, domains are named by a tag: the correspondence with domains and their characteristics is in Table A1.

V.V.3.6 Full comparisons with MICE on missing data imputation

The main file provides just two examples of domains for the comparison, `abalone` and `analcatdata_supreme`. We provide here more complete results on the domain used in the main file and results on more domains in the form of one table for each additional domain:

- Table A13: experiments on `analcatdata_supreme` completing the results shown in Table 4 (MF);
- Table A14: experiments on `abalone` completing the results shown in Table 4 (MF);

  (tables below are ordered in increasing domain size, see Table A1)
- Table A15: experiments on `iris`;
- Table A16: experiments on `ringgauss`;
- Table A17: experiments on `circgauss`;
- Table A18: experiments on `gridgauss`;
- Table A19: experiments on `randgauss`;
- Table A20: experiments on `student_performance_mat`;
- Table A21: experiments on `student_performance_por`;
- Table A22: experiments on `kc1`;
- Table A23: experiments on `sigma_cabs`;
- Table A24: experiments on `compas`;
- Table A25: experiments on `open_policing_hartford`;

The large amount of pictures to process may make it difficult to understand at first glance how our approaches behave against MICE, so we summarize here the key points:

domain	us (GF)	CT-GANs							
tag	T=200, J=500	$E = 10$	p-val	$E = 100$	p-val	$E = 300$	p-val	$E = 1000$	p-val
`ring`	0.283±0.005	0.457±0.058	0.0029	0.546±0.079	0.0020	0.405±0.044	0.0048	0.351±0.042	0.0299
`circ`	0.356±0.002	0.848±0.300	0.0217	0.480±0.040	0.0024	0.443±0.014	0.0001	0.435±0.075	0.0816
`grid`	0.392±0.002	0.749±0.417	0.1279	0.459±0.031	0.0096	0.426±0.022	0.0233	0.408±0.019	0.1239
`for`	1.086±0.124	9.796±5.454	0.0241	1.899±0.289	0.0028	1.532±0.178	0.0020	1.520±0.311	0.0313
`rand`	0.290±0.007	0.746±0.236	0.0129	0.491±0.063	0.0017	0.368±0.015	0.0003	0.327±0.024	0.0124
`tic`	0.574±0.001	0.601±0.003	ε	0.581±0.001	0.0024	0.586±0.006	0.0299	0.584±0.003	0.0228
`iono`	1.203±0.068	2.263±0.035	ε	2.084±0.052	ε	1.98±0.136	0.0001	1.758±0.137	0.0005
`stm`	1.044±0.013	1.511±0.074	0.0001	1.189±0.045	0.0106	1.168±0.054	0.0346	1.167±0.041	0.0036
`wred`	1.027±0.037	2.836±0.703	0.0049	2.004±0.090	ε	1.825±0.127	ε	1.384±0.047	0.0002
`stp`	1.029±0.027	1.660±0.161	0.0010	1.141±0.018	0.0004	1.206±0.046	0.0015	1.186±0.052	0.0057
`ana`	0.367±0.018	0.542±0.135	0.0399	0.439±0.056	0.0196	0.404±0.012	0.0231	0.436±0.036	0.0018
`aba`	0.476±0.017	1.689±0.053	ε	1.463±0.094	ε	0.754±0.040	ε	0.657±0.026	ε
`wwhi`	1.120±0.013	1.770±0.160	0.0009	1.849±0.064	ε	1.284±0.029	0.0006	1.158±0.009	0.0117
`arti*`	0.830±0.007	1.388±0.109$_4$	0.0021	1.209±0.018$_4$	ε	1.026±0.013$_4$	0.0002	0.959±0.013$_3$	0.0051
wins / lose for us →		14 / 0		14 / 0		14 / 0		14 / 0	

Table A10: Comparison of results with respect to CT-GANs [47] trained with different number of epochs E. ε means p-val $< 10^{-4}$. As already mentioned in Table A8, in some domains, indicated with a "*", CT-GANs crashed on some folds. For such domains, we indicate in index to CT-GANs results the number of folds (out of 5) for which this did *not* happen. In such cases, we restricted the statistical comparisons with us to the folds for which CT-GANs did not crash: the statistical tests take this into account; instead of providing all corresponding average performances for us, we keep giving the average performance for us on all five folds (leftmost column), which is thus indicative. Other conventions follow Table A9.

- first and most importantly, there is not one single type of our models that perform better than the other. Both Generative Forests and Ensembles of Generative Trees can be useful for the task of missing data imputation. For example, `analcatdata_supreme` gives a clear advantage to Generative Forests: they even beat MICE with random forests having 4 000 trees (that is hundreds of times more than our models) on both categorical variables (perr) and numerical variables (rmse). However, on `circgauss`, `student_performance_por` and `sigma_cabs`, Ensembles of Generative Trees obtain the best results. On `sigma_cabs`, they can perform on par with MICE if the number of tree is limited (which represents 100+ times less trees than MICE's models);

- as is already noted in the main file (MF) and visible on several plots (see *e,g,* `sigma_cabs`), overfitting can happen with our models (both Generative Forests and Ensembles of Generative Trees), which supports the idea that a pruning mechanism or a mechanism to stop training would be a strong plus to training such models. Interestingly, in several cases where overfitting seems to happen, increasing the number of splits can reduce the phenomenon, at fixed number of trees: see for example `iris` (50 → 100 iterations for EOGTs), `gridgauss`, `randgauss` and `kc1` (GFs);

- some domains show that our models seem to be better at estimating continuous (regression) rather than categorical (prediction) variables: see for example `abalone`, `student_performance_mat`, `student_performance_por` and our biggest domain, `open_policing_hartford`. Note from that last domain that a larger number of iterations or models with a larger number of trees may be required for the best results, in particular for Ensembles of Generative Trees;

- small models may be enough to get excellent results on real world domains whose size would, at first glance, seem to require much bigger ones: on `compas`, we can compete (rmse) or beat (perr) MICE whose models contain 7 000 trees with just 20 stumps;

- it is important to remember that our technique does not just offer the capability to do missing data imputation: our models can also generate data and compute the full density conditional to observed values, which is not the case of MICE's models, crafted with the sole purpose of solving the task of missing data imputation. Our objective was not to beat MICE, but rather show that Generative Forests and Ensembles of Generative Trees can also be useful for this task.

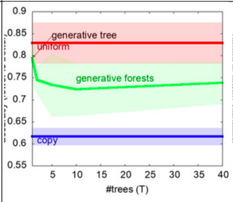
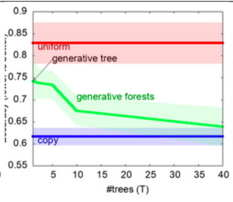
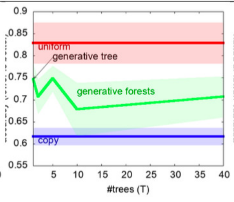
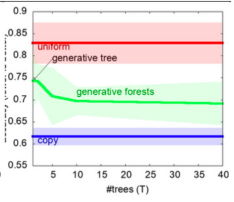
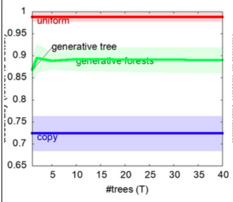
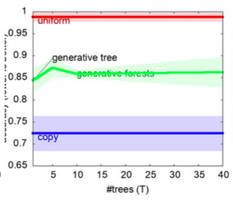
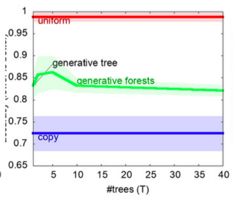
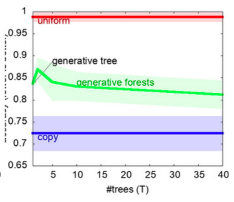
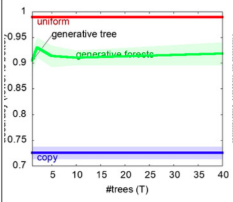
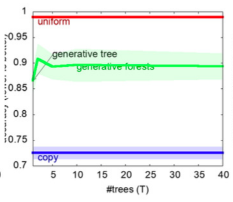
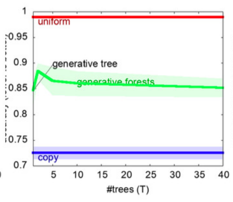
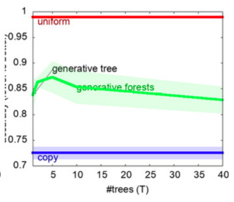
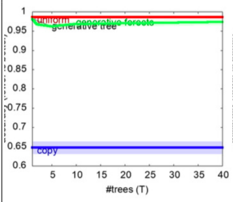
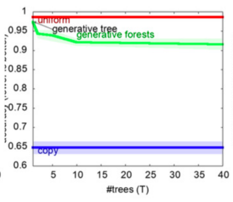
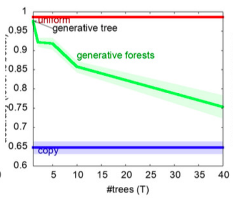
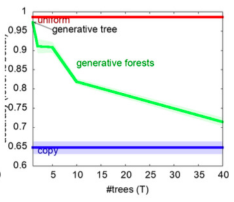
	$J = 20$	$J = 40$	$J = 80$	$J = 100$
`iris`				
`student_performance_mat`				
`student_performance_por`				
`abalone`				

Table A11: Results of generative forests on GEN-DISCRIM for four UCI domains, ordered, **from top to bottom, in increasing domain size**. In each row, the accuracy of distinguishing fakes from real is plotted (the *lower*, the *better* the technique) for four contenders: UNIFORM (generates observation uniformly at random), COPY ("optimal" contender using real data as generated), GF.BOOST inducing generative forests and generative trees (indicated for $T = 1$). A clear pattern emerges, that as the domain size grows, increasing J buys improvements and, if J is large enough, increasing T also improves results. See Table A12 for comparisons (us vs COPY) in terms of p-values.

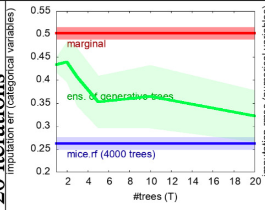
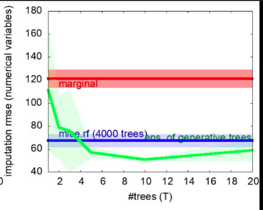
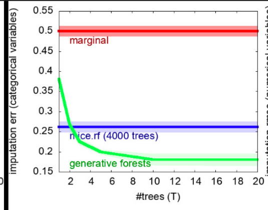
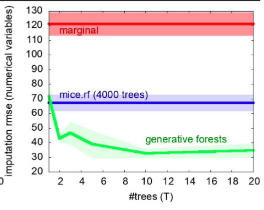
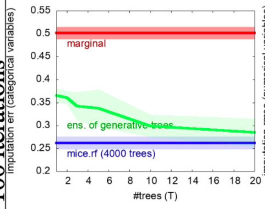
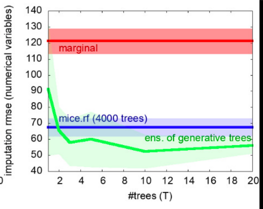
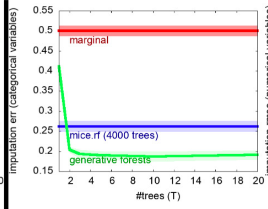
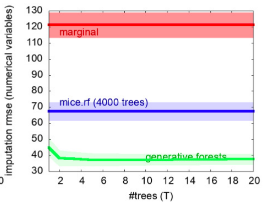
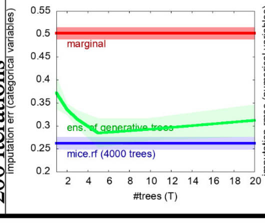
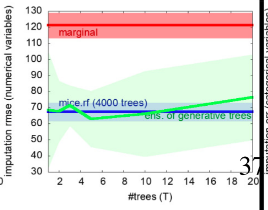
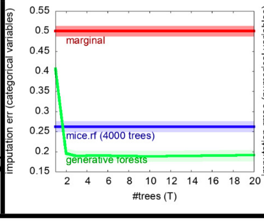
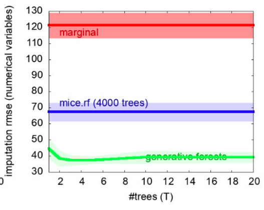
	Ensembles of Generative Trees (EOGT)		Generative Forests (GF)	
J	perr	rmse	perr	rmse
20 iterations				
100 iterations				
200 iterations				

Table A13: Missing data imputation: results on `analcatdata_supreme` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests) with 100

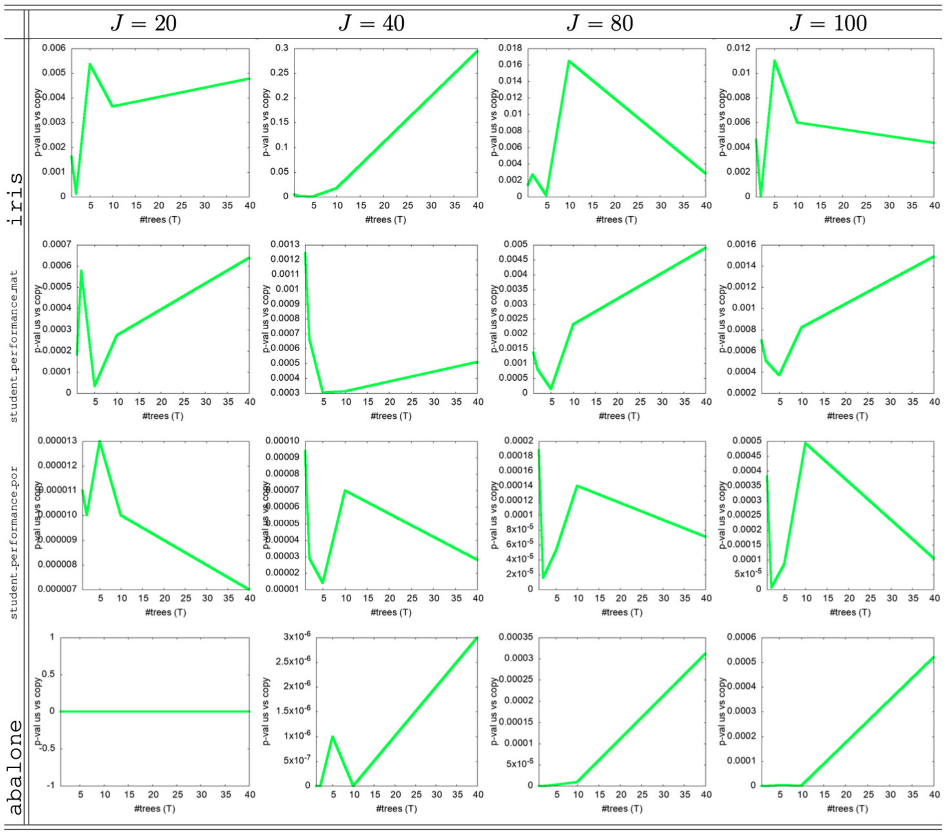


Table A12: p-values for Student tests with H_0 being "GF and COPY perform identically on GEN-DISCRIM" (for the domains in Table A11). Large values indicate we can keep H_0 and thus consider that GF is successful at generating "realistic" data. Clearly, GF are successful for `iris` (p increases with T up to $p \sim 0.3$ for $J = 40$). For the other domains, Table A11 suggests increasing J or T. The curves for `abalone` reveal a dramatic relative increase of p as (J, T) increases, starting from $p \sim 0, \forall T$ for $J = 20$.

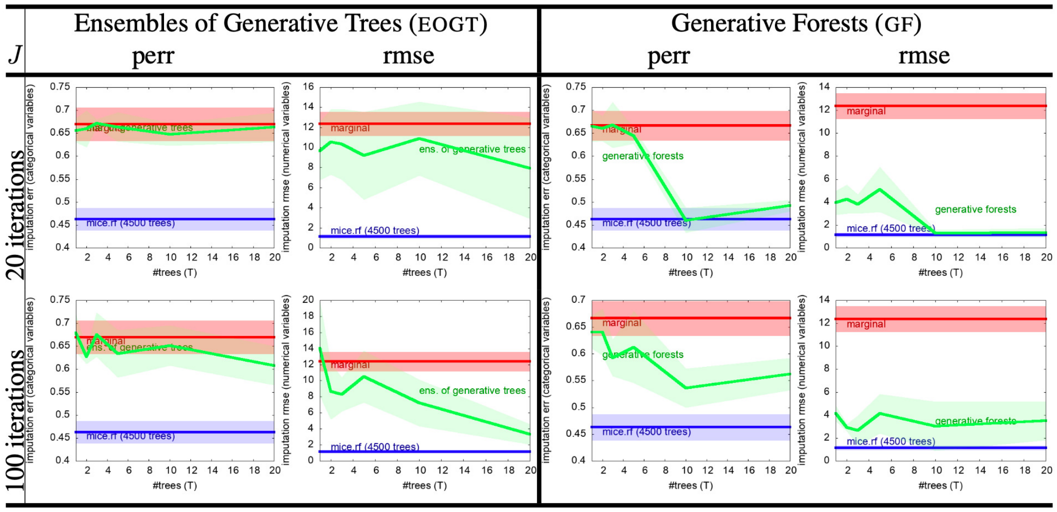


Table A14: Missing data imputation: results on `abalone` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests). Conventions follow Table A13.

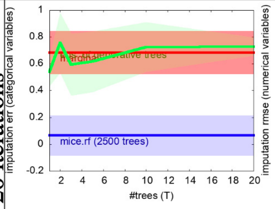
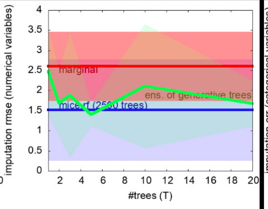
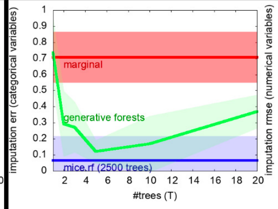
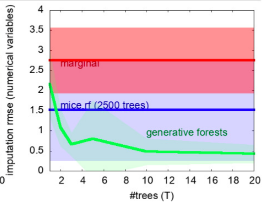
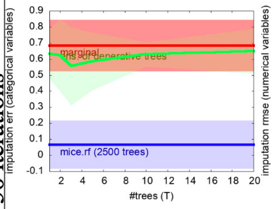
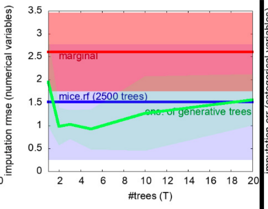
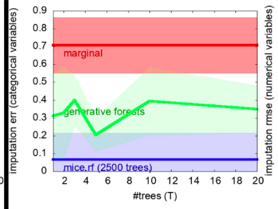
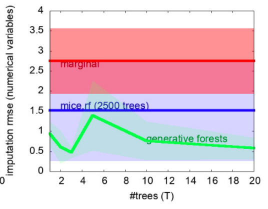
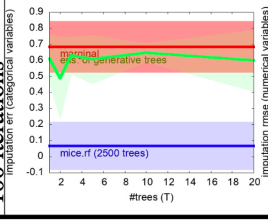
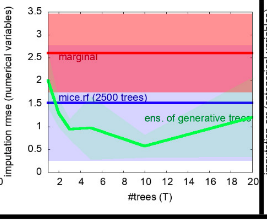
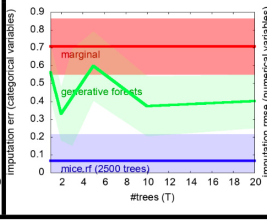
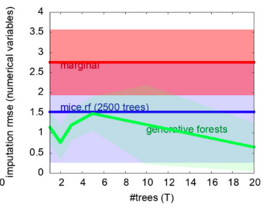
	Ensembles of Generative Trees (EOGT)		Generative Forests (GF)	
J	perr	rmse	perr	rmse
20 iterations				
50 iterations				
100 iterations				

Table A15: Missing data imputation: results on `iris` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

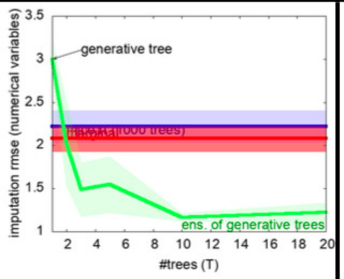
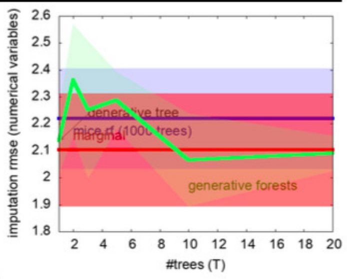
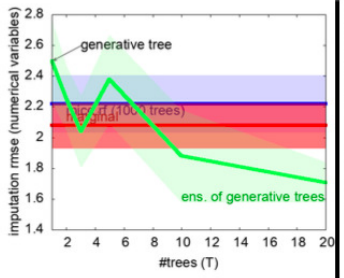
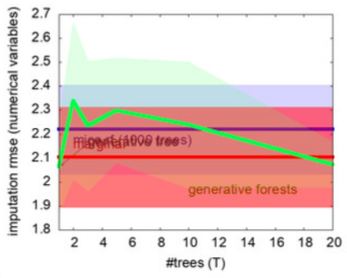
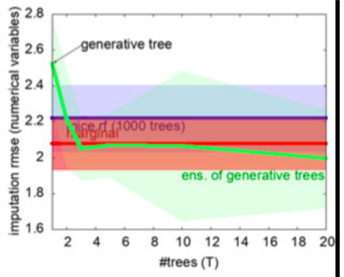
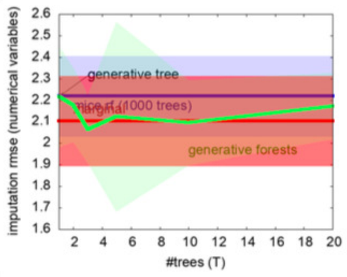
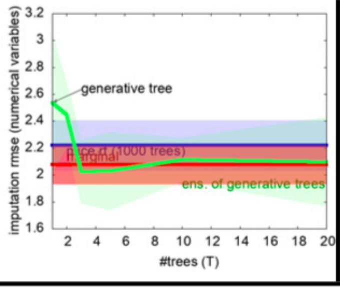
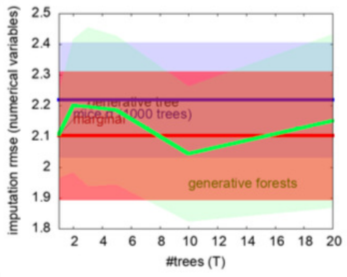
J	Ens. of Generative Trees (EOGT) rmse	Generative Forests (GF) rmse
20 iterations		
100 iterations		
200 iterations		
400 iterations		

Table A16: Missing data imputation: results on `ringgauss` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests). Since there are no nominal attributes in the domain, we have removed column perr. Conventions otherwise follow Table A13.

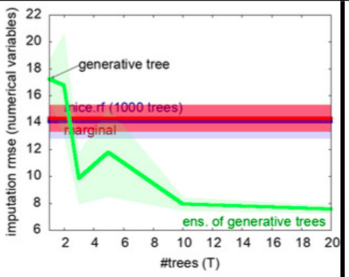
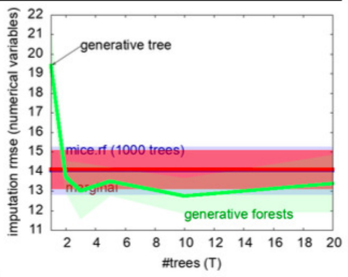
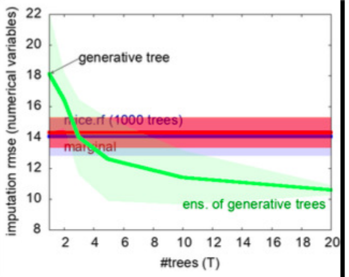
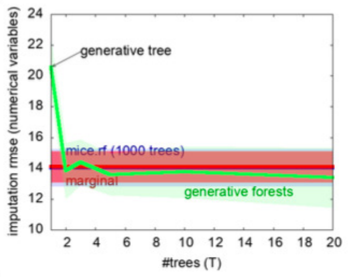
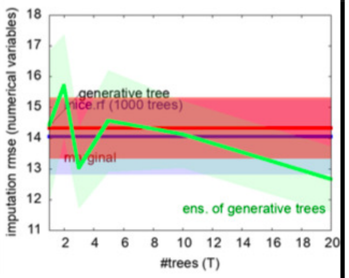
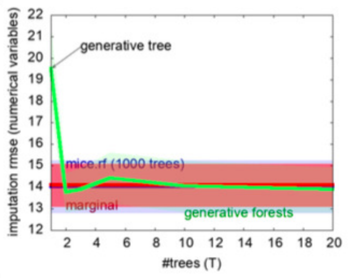
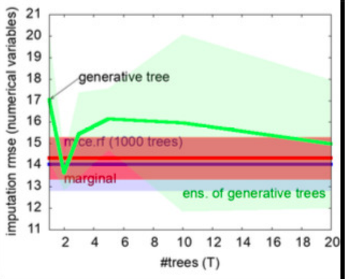
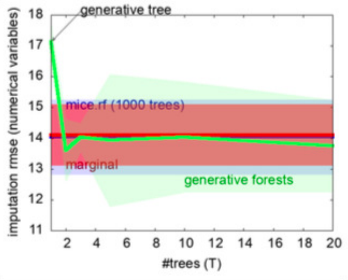
J	Ens. of Generative Trees (EOGT) rmse	Generative Forests (GF) rmse
20 iterations		
100 iterations		
200 iterations		
400 iterations		

Table A17: Missing data imputation: results on `circgauss` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests). Since there are no nominal attributes in the domain, we have removed column perr. Conventions otherwise follow Table A13.

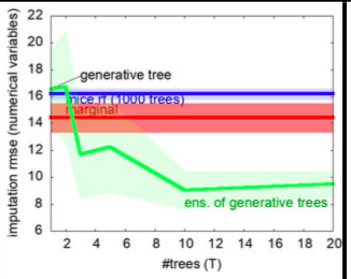
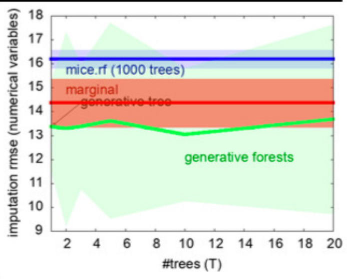
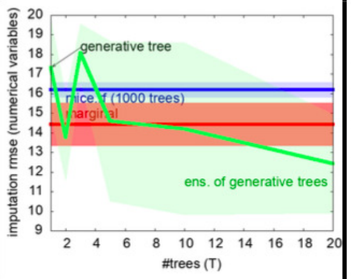
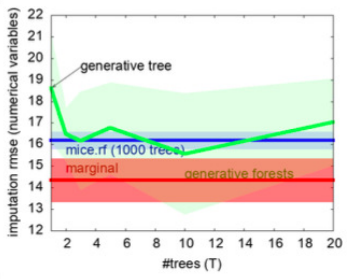
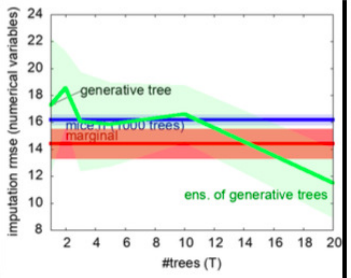
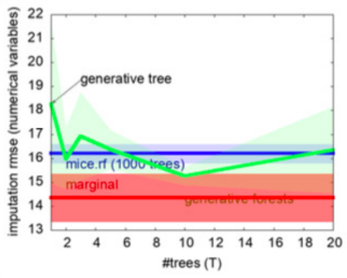
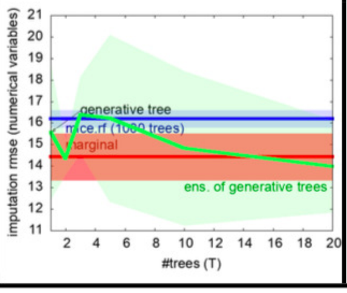
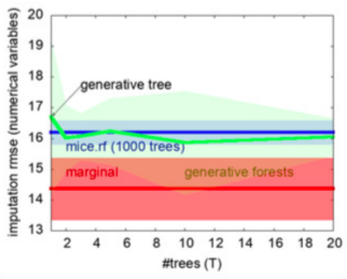
J	Ens. of Generative Trees (EOGT) rmse	Generative Forests (GF) rmse
20 iterations		
100 iterations		
200 iterations		
400 iterations		

Table A18: Missing data imputation: results on `gridgauss` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests). Since there are no nominal attributes in the domain, we have removed column perr. Conventions otherwise follow Table A13.

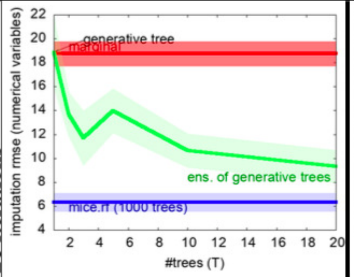
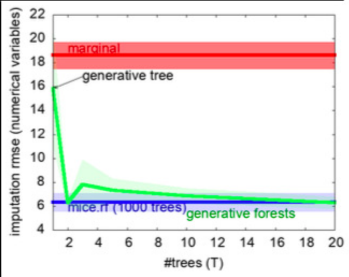
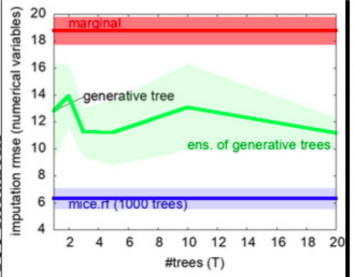
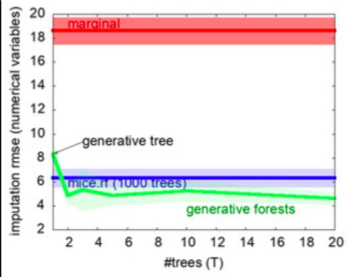
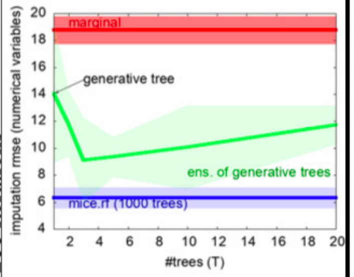
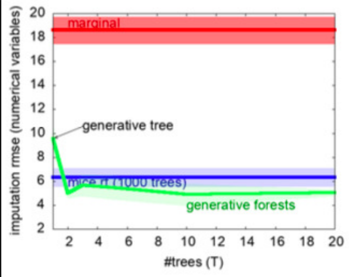
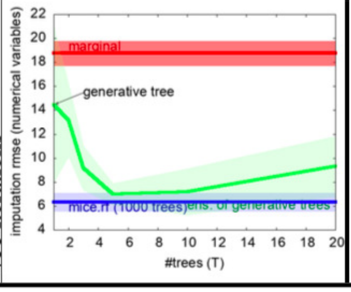
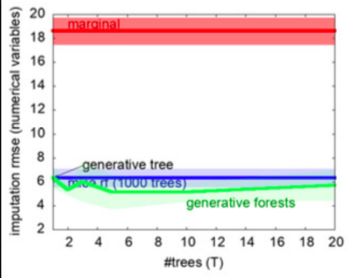
J	Ens. of Generative Trees (EOGT) rmse	Generative Forests (GF) rmse
20 iterations		
100 iterations		
200 iterations		
400 iterations		

Table A19: Missing data imputation: results on `randgauss` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using RF (random forests). Since there are no nominal attributes in the domain, we have removed column perr. Conventions otherwise follow Table A13.

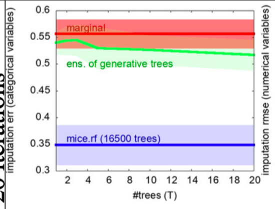
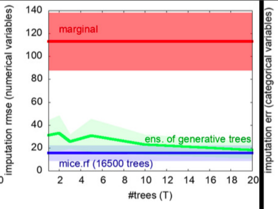
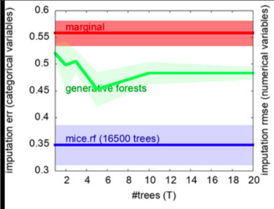
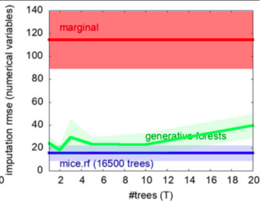
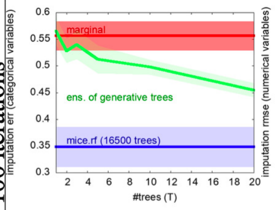
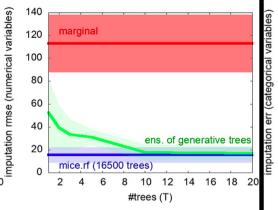
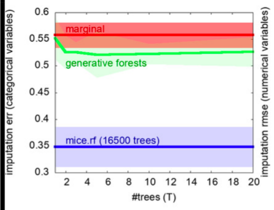
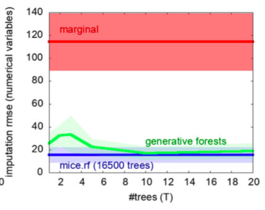
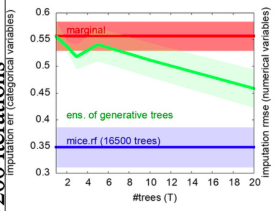
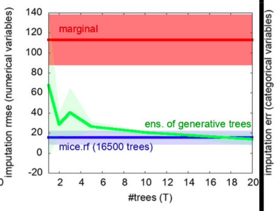
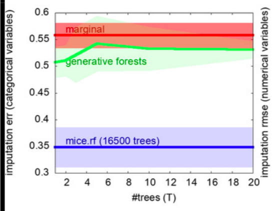
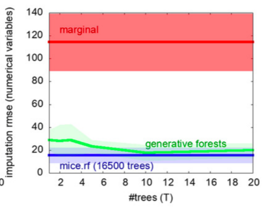
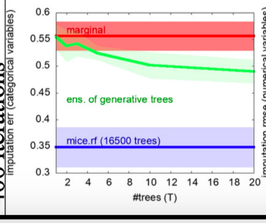
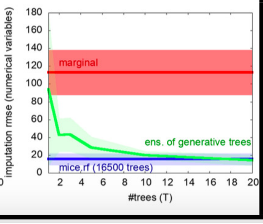
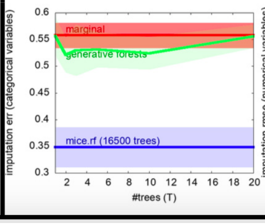
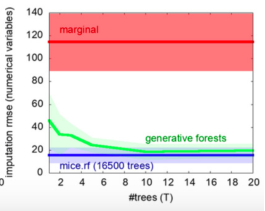
J	Ensembles of Generative Trees (EOGT) perr	Ensembles of Generative Trees (EOGT) rmse	Generative Forests (GF) perr	Generative Forests (GF) rmse
20 iterations				
100 iterations				
200 iterations				
400 iterations				

Table A20: Missing data imputation: results on `student_performance_mat` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

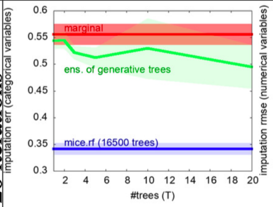
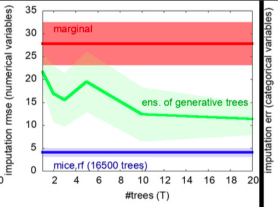
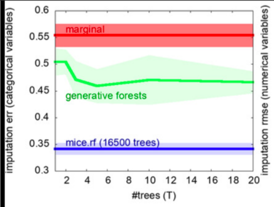
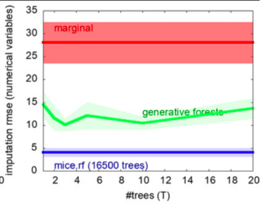
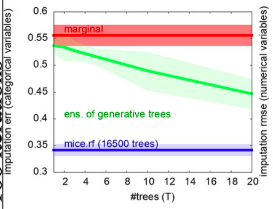
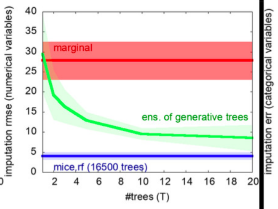
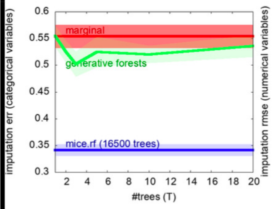
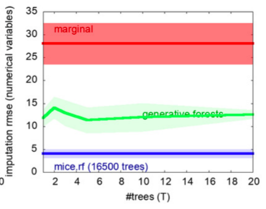
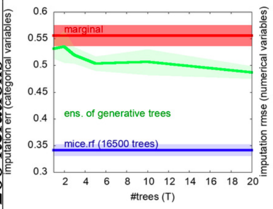
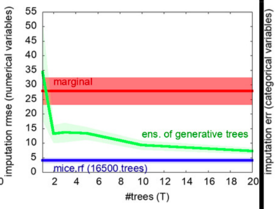
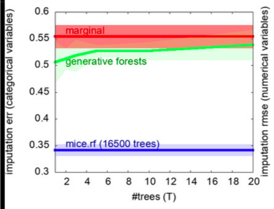
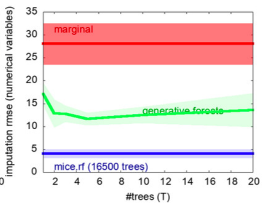
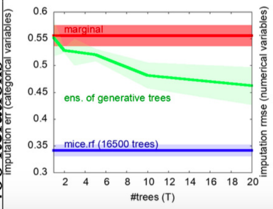
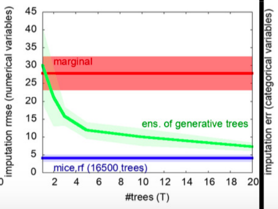
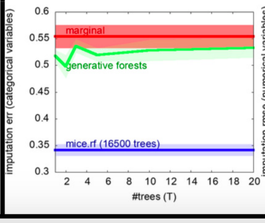
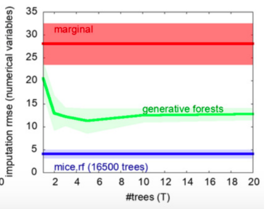
	Ensembles of Generative Trees (EOGT)		Generative Forests (GF)	
J	perr	rmse	perr	rmse
20 iterations				
100 iterations				
200 iterations				
400 iterations				

Table A21: Missing data imputation: results on `student_performance_por` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

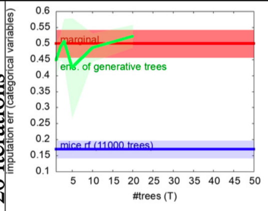
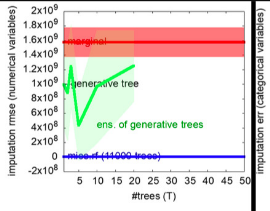
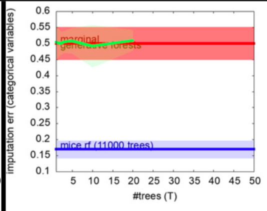
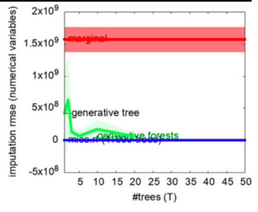
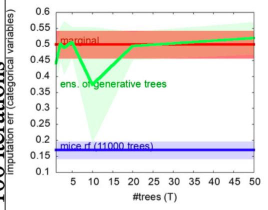
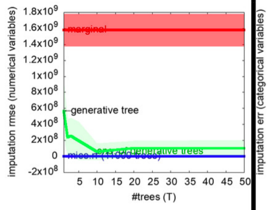
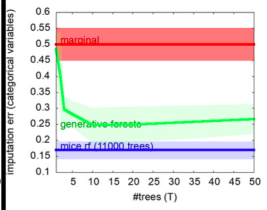
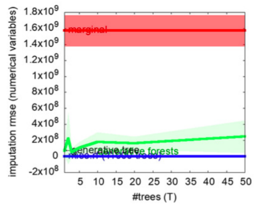
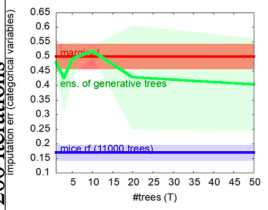
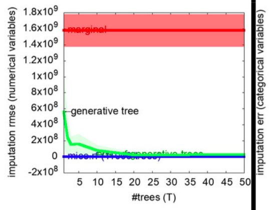
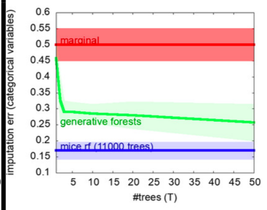
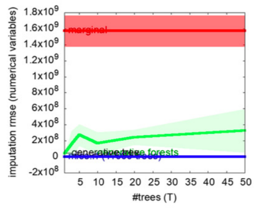
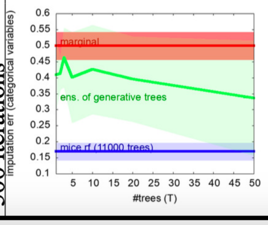
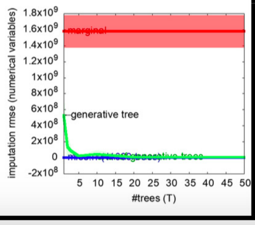
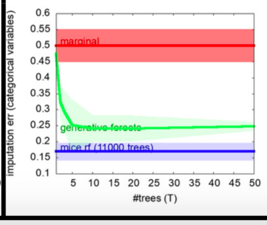
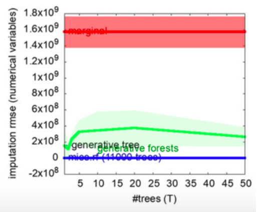
J	Ensembles of Generative Trees (EOGT) perr	Ensembles of Generative Trees (EOGT) rmse	Generative Forests (GF) perr	Generative Forests (GF) rmse
20 iterations				
100 iterations				
200 iterations				
500 iterations				

Table A22: Missing data imputation: results on `kc1` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

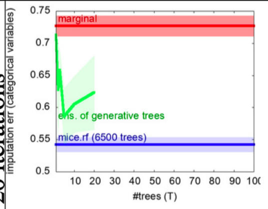
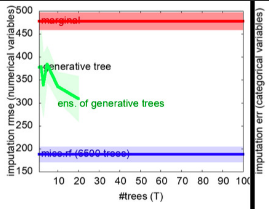
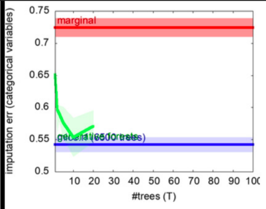
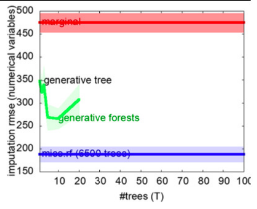
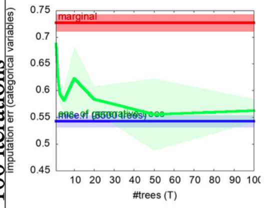
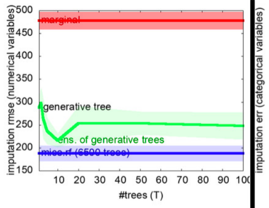
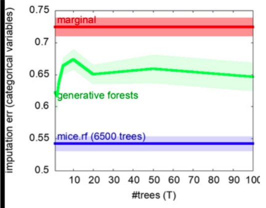
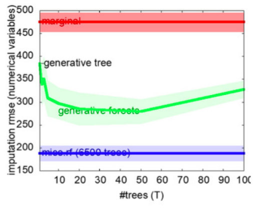
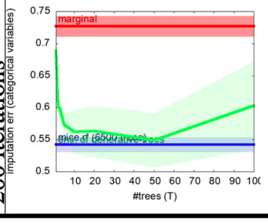
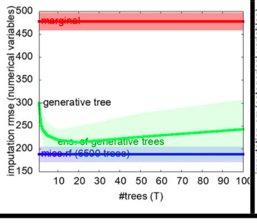
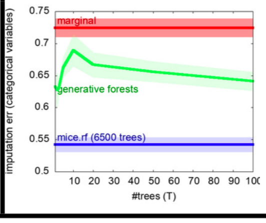
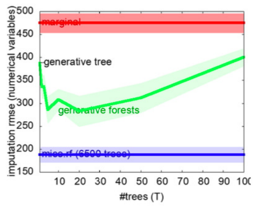
	Ensembles of Generative Trees (EOGT)		Generative Forests (GF)	
J	perr	rmse	perr	rmse
20 iterations				
100 iterations				
200 iterations				

Table A23: Missing data imputation: results on `sigma_cabs` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

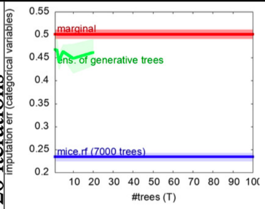
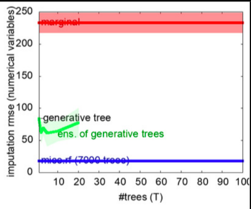
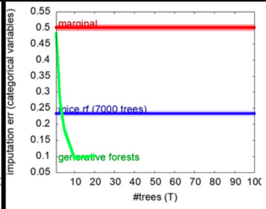
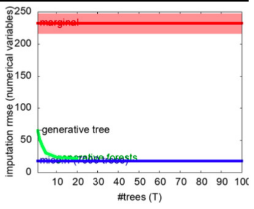
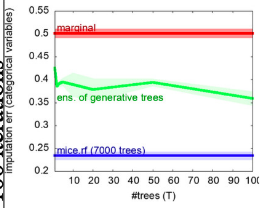
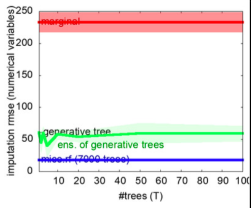
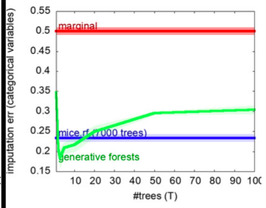
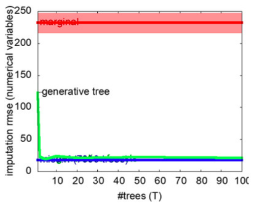
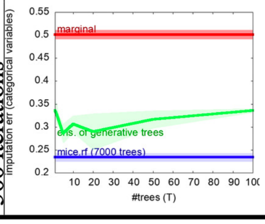
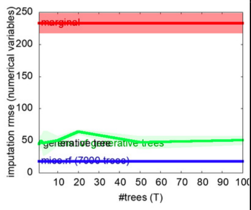
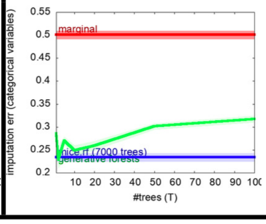
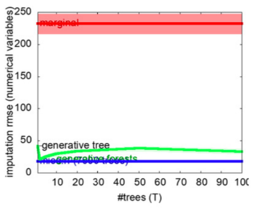
J	Ensembles of Generative Trees (EOGT) perr	Ensembles of Generative Trees (EOGT) rmse	Generative Forests (GF) perr	Generative Forests (GF) rmse
20 iterations				
100 iterations				
500 iterations				

Table A24: Missing data imputation: results on `compas` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

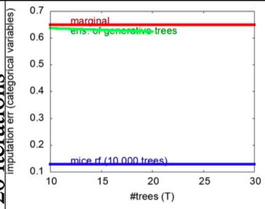
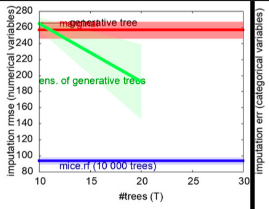
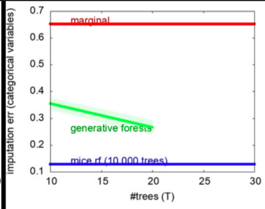
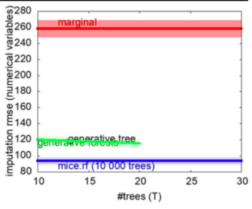
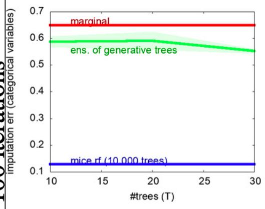
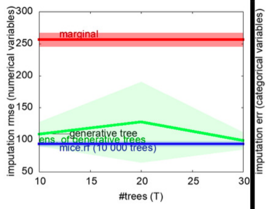
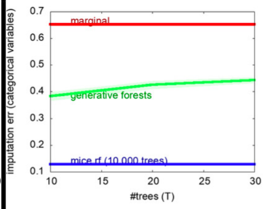
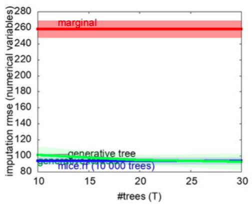
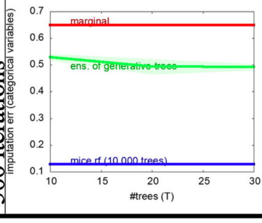
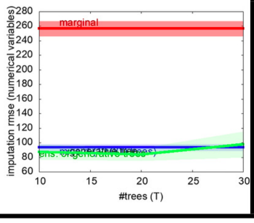
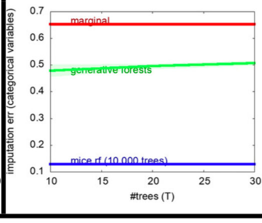
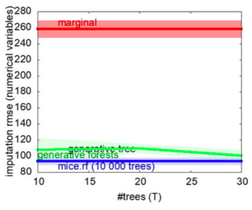
	Ensembles of Generative Trees (EOGT)		Generative Forests (GF)	
J	perr	rmse	perr	rmse
20 iterations				
100 iterations				
500 iterations				

Table A25: Missing data imputation: results on `open_policing_hartford` with 5% missing features (MCAR), for GF.BOOST learning GFs and EOGTs, vs `mice` using `RF` (random forests). Conventions follow Table A13.

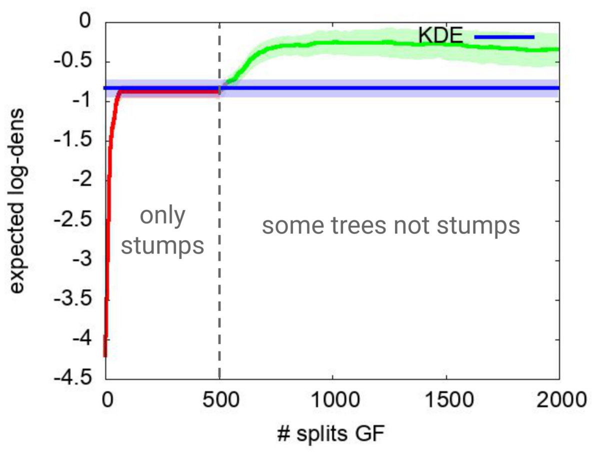


Table A26: DENSITY: zoom on the `gridGauss` domain: the GF is grown by GF.BOOST in a way that first learns a complete set of stumps (there are a total of $T = 500$ trees in the final model) and then grows some of the trees. The plateau for $j \leqslant 500$ leaves displays that we quickly reach a limit in terms of maximal performances of stumps, this being probably dues to the fact that the dimension of the domain is small (2). We can however remark that with just stumps, we still attain close performances to KDE (note that we plot the expected log-densities; see text for details).

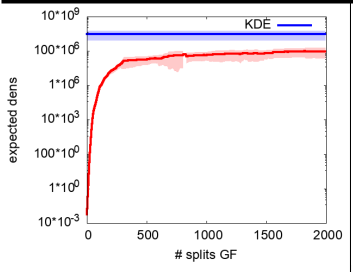
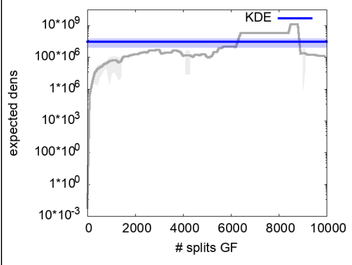
	
$T = 500, J = 2000$	$T = 15, J = 10000$

Table A27: DENSITY: zoom on the `abalone` domain: with $T = 500$ trees, total of $J = 2000$ as in Tables A28 and A29, we are clearly beaten by KDE (left). A much smaller of bigger trees ($T = 15, J = 10000$) is enough to become competitive and beat KDE, albeit not statistically significantly (right). The right picture also displays the interest into getting pruning or early stopping algorithms to prevent eventual overfitting.

V.V.3.7 Full comparisons with KDE on density estimation

For each domain we either use the expected density, or the expected negative log-density, as the metric to be maximized. We consider the expected density to cope with the eventuality of zero (pointwise) density predictions. Tables A28 and A29 present the results against KDE (details in Table A28). Table A26 singles out a result on the domain `gridGauss`, which displays an interesting pattern. The leaf chosen for a split in GF.BOOST is always the heaviest leaf; hence, as long as the iteration number $j \leqslant T$ (the maximum number of trees, which is 500 in this experiment), GF.BOOST grows stumps. In the case of `gridGauss`, like in a few other domains, there is a clear plateau in performances for $j \leqslant 500$, which does not hold anymore as $j > 500$. This, we believe, characterizes the fact that on these domains, which all share the property that their dimension is small, training stumps attains a limit in terms of performances. Table A27 shows that by changing basic parameters – here, decreasing the number of trees, increasing the total number of iterations –, one can obtain substantially better results.

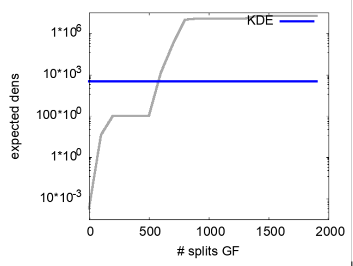
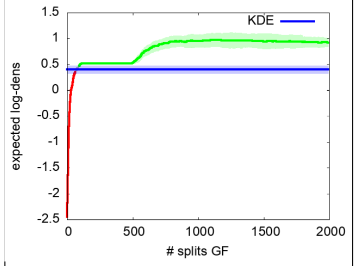
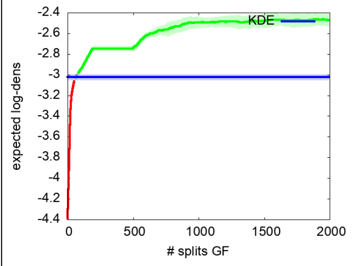
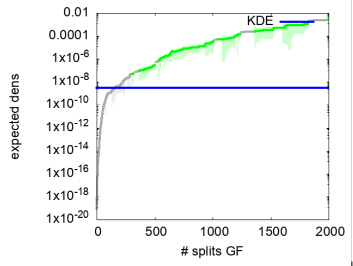
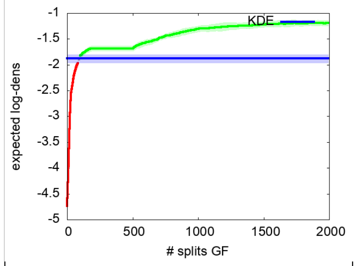
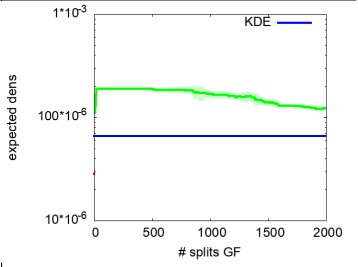
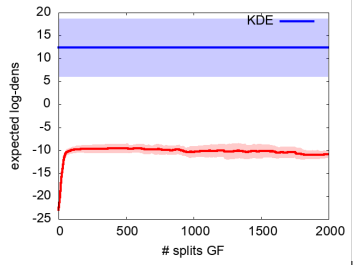
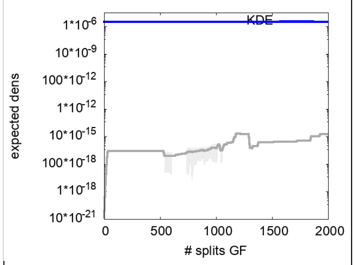
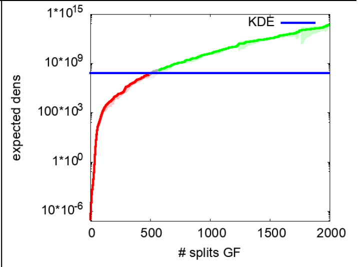
		
`iris`	`ringGauss`	`circGauss`
		
`forestfires`	`randGauss`	`tictactoe`
		
`ionosphere`	`student_performance_mat`	`winered`

Table A28: Comparison of results for Kernel Density Estimation (KDE) and Generative Forests (us, GF) with parameters $T = 500$ trees, total of $J = 2000$ splits in trees, on a first batch of domains, put in increasing size according to Table A1. For each domain, we compute for GF at regular intervals the average $\pm$ standard deviation of either (i) density values or (ii) negative log density values for the test fold (sometimes, KDE gives 0 estimated density which prevents the computation of (ii)). Warning: some y scales are logscale. We then compute the baseline of KDE and display its average $\pm$ standard deviation in blue on each picture. We perform, at each applicable iteration of GF.BOOST (*i.e.* each iteration for many domains, which can be seen from the plots), a Student paired t-test over the five folds to compare the results with KDE. If we are statistically better ($p = 0.1$), our plot is displayed in green; if KDE is better, our plot is displayed in red; otherwise it is displayed in gray. When standard deviations are not displayed, it means average $\pm$ standard deviation exceeds the plot's y scale.

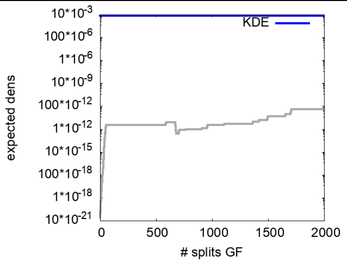
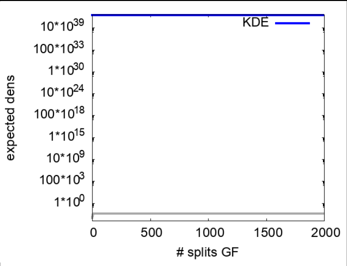
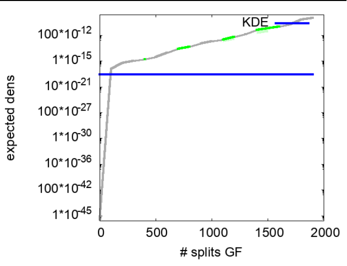
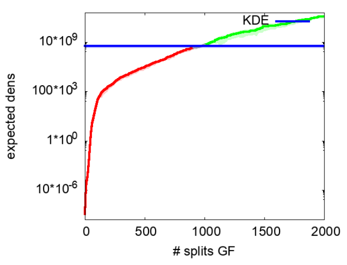
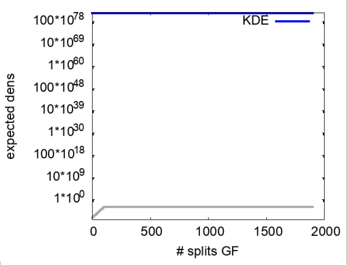
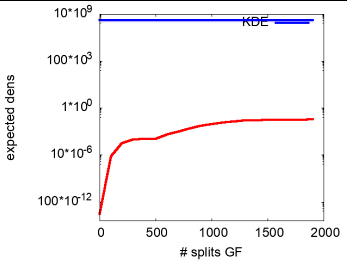
		
`student_performance_por`	`analcatdata_supreme`	`kc1`
		
`winewhite`	`compas`	`artificial_characters`

Table A29: Comparison of results for Kernel Density Estimation and Generative Forests (us, GF) where the number of trees and number of iterations are ($T = 500$ trees, total of $J = 2000$ splits in trees), on a second batch of domains. Conventions follow Table A28.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Claims are supported by both the theoretical results presented in the paper and extensive experimental results, summarized in the main paper but otherwise presented in extenso in the Supplement.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed, in particular in the experimental section vs the SOTA (see e.g. experiment Lifelike).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Assumptions are detailed, formal results given and full proofs are provided in the Appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: **(i)** full code provided, **(ii)** extensive README.txt, **(iii)** coding choices discussed in Appendix, **(iv)** all scripts provided for easy running of each experiments, **(v)** special visualization routines (e.g. heatmaps) also included in the code, **(vi)** example domains (public and simulated) provided for quick assessment.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: **(i)** full code provided, **(ii)** extensive README.txt, **(iii)** coding choices discussed in Appendix, **(iv)** all scripts provided for easy running of each experiments, **(v)** special visualization routines (e.g. heatmaps) also included in the code, **(vi)** example domains (public and simulated) provided for quick assessment.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details are provided in main text and Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: standard devs and inferential statistics test done (details in main file + appendix).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [NA] .

   Justification: Our code runs on any standard computer. Details of codes, computer and SOTA used (version, etc.) in appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research of the paper follows the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See conclusion.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No release of data or models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Appropriate credit / referencesse are provided. Licenses listed in Appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new assets provided.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.