WildWood: A New Random Forest Algorithm

Stéphane Gaïffas, Ibrahim Merad®, and Yiyang Yu

Abstract—We introduce WildWood (WW), a new ensemble algorithm for supervised learning of Random Forest (RF) type. While standard RF algorithms use bootstrap out-of-bag samples to compute out-of-bag scores, WW uses these samples to produce improved predictions given by an aggregation of the predictions of all possible subtrees of each fully grown tree in the forest. This is achieved by aggregation with exponential weights computed over out-of-bag samples, that are computed exactly and very efficiently thanks to an algorithm called context tree weighting. This improvement, combined with a splitting strategy to accelerate split finding, makes WW fast and competitive compared with other well-established ensemble methods, such as standard RF and extreme gradient boosting algorithms.

Index Terms—Random forest, aggregation methods, ensemble methods, decision tree.

I. INTRODUCTION

This paper introduces WildWood (WW), a new ensemble method of Random Forest (RF) type [1]. The main contributions of the paper and the main advantages of WW are as follows. Firstly, we use out-of-bag samples (trees in a RF use different bootstrapped samples) very differently than what is done in standard RF [2], [3]. Indeed, WW uses these samples to compute an aggregation of the predictions of all possible subtrees of each tree in the forest, using aggregation with exponential weights [4]. This leads to much improved predictions: while only leaves contribute to the predictions of a tree in standard RF, the full tree structure contributes to predictions in WW. An illustration of this effect is given in Figure 1 on a toy binary classification example, where we can observe that subtrees aggregation leads to improved and regularized decision functions for each individual tree and for the forest. We further illustrate in Figure 2 that each tree becomes a stronger learner, and that excellent performance can be achieved even when WW uses few trees, granting better interpretability of the solutions as a side benefit. Indeed, reducing the number of trees is a known way to obtain explainable models [5], [6]. A remarkable aspect of WW is that this improvement comes only at a small computational cost, thanks to a technique called “context tree weighting”, used in lossless compression or online learning to aggregate all subtrees of a given tree [4], [7], [8], [9], [10]. Also, the predictions of WW do not rely on MCMC approximations required with Bayesian variants of RF [11], [12], [13], [14], which is a clear distinction from such methods. Secondly, WW uses feature binning (“histogram” strategy), similarly to extreme gradient boosting (EGB) libraries such as XGBoost [16], LightGBM [17] and CatBoost [18], [19]. This strategy helps accelerate computations in WW compared with standard RF algorithms, that typically require to sort features locally in nodes and try a larger number of splits [2]. This combination of subtrees aggregation and of the histogram strategy makes WW comparable with state-of-the-art implementations of EGB libraries, as illustrated in Figure 3. Moreover, WW supports optimal split finding for categorical features and missing values, with no need for particular pre-processing (such as one-hot encoding, see [16] or target encoding, see [18], [19]). Finally, WW is supported by some theoretical evidence, since we prove that for a general loss function, the subtrees aggregation considered in WW leads indeed to a performance close to that of the best subtree.

A. Related Works

Tree based decision models appeared as a supervised learning tool in the 1960s [20], [21], [22]. Their use was promoted by the in-depth practical and theoretical study of [23]. The key idea of forming tree ensembles led to the well-known Random Forest algorithm [1], [24] which became one of the most popular supervised learning algorithms thanks to its ease of use, robustness to hyperparameters [3], [25] and applicability to a wide range of domains. Recent examples include predictive medicine [26], [27], intrusion detection [28], car safety [29], differential privacy [30] and COVID-19 [31] to cite but a few. The RF algorithm sparked considerable interest and benefited from a rich literature of methodological developments [15], [32], [33]. In particular, RF was adapted for various tasks including the computation of prediction intervals [34], [35], [36], online learning [10], [37] and quantile regression [38]. A theoretical framework was progressively built around random forests to study their statistical properties such as consistency [39], [40], [41], control over bias and variance [42], [43] and feature importance assessment [44], [45]. These works often consider purely random trees (splits are not optimized using training data) which are more amenable...
to theoretical analysis. More recent works introduced “honest” trees which are built using disjoint samples from those used for prediction [46]. Honest forests were considered and proven consistent for causal inference tasks by [47].

By essence, decision trees are universal approximators which lends them strong expressive power but also makes them prone to overfitting, hence the need for regularization. This was addressed in early works using pruning methods such as CCP, REP or MEP [22], [48]. Although they are fairly effective at reducing tree overfitting, these methods are mostly based on heuristics so that little is known about their theoretical properties. A form of soft-pruning was proposed by [49] and referred to as tree smoothing. The latter efficiently computes predictions as approximate Bayesian posteriors over the set of possible prunings, however, the associated complexity is of the order of the tree-size, which makes the computation of predictions slow. An alternative form of regularization called “shrinkage” was proposed in [50] and [51] and consists in averaging predictions between leaf nodes and their ancestors. Although a parallel can be drawn between such methods and kernel Ridge regression, their theoretical support remains limited and does not allow to claim improved performance.
Recent studies suggest that tree depth limitation and randomness of the tree building process bring an implicit regularization effect which improves performance on low signal-to-noise ratio datasets [52], [53].

In [10], an improvement of Mondrian Forests [37] is introduced for online learning, using subtrees aggregation with exponential weights, which is particularly convenient in the online learning setting. However, [10] considers only the online setting, with purely random trees, leading to poor performances compared with realistic decision trees.

Extreme boosting algorithms are another family of tree ensemble methods. XGBoost [16] provides an extremely popular scalable tree boosting system which has been widely adopted in industry. LightGBM [17] employed the “histogram strategy” for faster split finding, together with clever down-sampling and features grouping algorithms in order to achieve high performance in reduced computation times. CatBoost [25] is another boosting library which pays particular attention to categorical features using target encoding, while addressing the potential bias issues associated to such an encoding.

In WW, we combine several ideas from the literature into a novel high performing RF algorithm. We use a subtrees aggregation mechanism similar to [10] for batch learning in a different way: we exploit the bootstrap, one of the key ingredients of RF, which provides in-the-bag and out-of-bag samples, to perform aggregation with exponential weights leading to considerably improved predictions. In addition, we employ the histogram strategy [17] in order to grow decision trees much more efficiently. Finally, we implement an improved split finding method along categorical features which makes for better predictions.

B. Limitations

Our implementation of WW is still evolving and is not yet at the level of maturity of state-of-the-art EGB libraries such as [16], [17], and [25]. It does not outperform such strong baselines, but proposes an improvement of RF algorithms, and gives an interesting balance between performance and computational efficiency.

II. WILDWOOD: A NEW RANDOM FOREST ALGORITHM

We consider batch supervised learning, where data comes as a set of i.i.d training samples $(x_i, y_i)$ for $i = 1, \ldots, n$ with vectors of numerical or categorical features $x_i \in \mathcal{X} \subset \mathbb{R}^d$ and $y_i \in \mathcal{Y}$. Our aim is to design a RF predictor $\hat{g}(\cdot; \Pi) = \frac{1}{M} \sum_{m=1}^M \hat{f}(\cdot; \Pi_m) : \mathcal{X} \to \mathcal{Y}$ computed from training samples, where $\mathcal{Y}$ is the prediction space. Such a RF computes the average of $M$ randomized trees predictions $\hat{f}(\cdot; \Pi_m)$ following the principle of bagging [54], [55], with $\Pi = (\Pi_1, \ldots, \Pi_M)$ where $\Pi_1, \ldots, \Pi_M$ are i.i.d realizations of a random variable corresponding to bootstrap and feature subsampling (see Section II-A below). Each tree is trained independently of each other, in parallel. In what follows we describe only the construction of a single tree and omit from now on the dependence on $m = 1, \ldots, M$.

Feature Binning: The split finding strategy described in Section II-B below works on binned features. While this technique is of common practice in EGB libraries [16], [17], [18], we are not aware of an implementation of it for RF. Note that binning may affect predictive performance by making split thresholds less accurate. However, the experimental performance from EGB libraries (see also Section IV) suggests that this effect is minor. The input $n \times d$ matrix $X$ of features is transformed into another same-size matrix of “binned” features denoted $X^{\text{bin}}$. To each input feature $j = 1, \ldots, d$ is associated a set $B_j = \{1, \ldots, b_j\}$ of bins, where $b_j \leq b_{\max}$ with $b_{\max}$ a hyperparameter corresponding to the maximum number of bins a feature can use (default is $b_{\max} = 256$ similarly to [17], so that a single byte can be used for entries of $X^{\text{bin}}$). When a feature is continuous, it is binned into $b_{\max}$ bins using interquartile intervals. If it is categorical, each modality is mapped to a bin whenever $b_{\max}$ is larger than its number of modalities, otherwise sparsest modalities end up binned together. If a feature $j$ contains missing values, its rightmost bin in $B_j$ is used to encode them (in such case, later in split, we do not loop only left to right (along bin order), but right to left as well, in order to compare splits that put missing values on the left or on the right). After binning, each column satisfies $X^{\text{bin}}_{*,j} \in B_j$.

A. Random Decision Trees

Let $C = \prod_{i=1}^n B_j$ be the binned feature space. A random decision tree is a pair $(T, \Sigma)$, where $T$ is a finite ordered binary tree and $\Sigma$ contains information about each node in $T$, such as split information. The tree is random and its source of randomness II comes from the bootstrap and feature subsampling as explained below.

1) Finite Ordered Binary Trees: A finite ordered binary tree $T$ is represented as a finite subset of the set $\{0, 1\}^* = \bigcup_{n \geq 0} \{0, 1\}^n$ of all finite words on $\{0, 1\}$. The set $\{0, 1\}^*$ is endowed with a tree structure (and called the complete binary tree); the empty word $\emptyset$ is the root, and for any $v \in \{0, 1\}^*$, the left (resp. right) child of $v$ is $v0$ (resp. $v1$).

We denote by $\text{innodes}(T) = \{v \in T : v0, v1 \in T\}$ the set of its interior nodes and by $\text{leaves}(T) = \{v \in T : v0, v1 \notin T\}$ the set of its leaves, both sets are disjoint and the set of all nodes is $\text{nodes}(T) = \text{innodes}(T) \cup \text{leaves}(T)$.

2) Splits and Cells: The split $\sigma_j = (j_v, t_v) \in \Sigma$ of each $v \in \text{innodes}(T)$ is characterized by its dimension $j_v \in \{1, \ldots, d\}$ and a non-empty subset of bins $t_v \subseteq \{1, \ldots, b_{j_v}\}$. We associate to each $v \in T$ a cell $C_v \subseteq C$ which is defined recursively: $C_{\emptyset} = C$ and for each $v \in \text{innodes}(T)$ we define $C_{v0} := \{x \in C_v : x_{j_v} \in t_v\}$ and $C_{v1} := C_v \setminus C_{v0}$.

When $j_v$ corresponds to a continuous feature, bins have a natural order so that $t_v$ is encoded by a bin threshold $s_v \in B_{j_v}$ as $t_v = \{1, 2, \ldots, s_v\}$; while for a categorical split, the set $t_v$ may be any non-trivial subset of $B_{j_v}$. By construction, $(C_v)_{v \in \text{leaves}(T)}$ is a partition of $C$.

3) Bootstrap and Feature Subsampling: Let $I = \{1, \ldots, n\}$ be the training samples indices. The randomization II of the tree uses bootstrap: it samples uniformly at random, with replacement, elements of $I$ corresponding to in-the-bag (i.t.b)
samples. If we denote as $I_{itb}$ the indices of unique itb samples, we can define the indices of out-of-bag (oob) samples as $I_{oob} = I \setminus I_{itb}$. A standard argument shows that $P[i \in I_{itb}] = 1 - (1-1/n)^n \rightarrow 1-e^{-1} \approx 0.632$ as $n \rightarrow +\infty$, known as the 0.632 rule [56]. The randomization II uses also feature subsampling: each time we need to find a split, we do not try all the features $\{1, \ldots, d\}$ but only a subset of them of size $d_{\text{max}}$, chosen uniformly at random. This follows what standard RF algorithms do [1], [2], [3], with the default $d_{\text{max}} = \sqrt{d}$.

B. Split Finding on Histograms

For $K$-class classification, when looking for a split for some node $v$, we compute the node’s “histogram” $\text{hist}_v[j, b, k] = \sum_{i \in I_{\text{hist}}: x_i \in C_v} 1_{x_i \in b, v_i \equiv k}$ for each sampled feature $j$, each bin $b$ and label class $k$ seen in the node’s samples (actually weighted counts to handle bootstrapping and sample weights).

Of course, one has $\text{hist}_v = \text{hist}_v0 + \text{hist}_v1$, so that we don’t need to compute two histograms for siblings $v0$ and $v1$, but only a single one. Then, we loop over the set of non-constant (in the node) sampled features $\{j : \#(b : \sum_k \text{hist}_v[j, b, k] \geq 1) \geq 2\}$ and over the set of non-empty bins $\{b : \sum_k \text{hist}_v[j, b, k] \geq 1\}$ to find a split, by comparing standard impurity criteria computed on the histogram’s statistics, such as gini or entropy for classification and variance for regression.

1) Bin Order and Categorical Features: The order of the bins used in the loop depends on the type of the feature. If it is continuous, we use the natural order of bins. If it is categorical and the task is binary classification (labels in $\{0, 1\}$) we use the bin order that sorts $\text{hist}_v[j, b, 1]/\sum_k \text{hist}_v[j, b, k]$ with respect to $b$, namely the proportion of labels 1 in each bin. This allows to find the optimal split with complexity $O(b_j \log b_j)$, see Theorem 9.6 in [23], the logarithm coming from the sorting operation, while there are $2^{b_j-1} - 1$ possible splits. This trick is used by EGB libraries as well, using an order of gradient/hessian statistics of the loss considered [16], [17], [18]. For $K$-class classification with $K > 2$, we consider two strategies: (1) one-versus-rest, where we train $MK$ trees instead of $M$, each tree trained with a binary one-versus-rest label, so that trees can find optimal categorical splits and (2) heuristic, where we train $M$ trees and where split finding uses $K$ loops over bin orders that sort $\text{hist}_v[j, b, k]/\sum_k \text{hist}_v[j, b, k']$ (w.r.t $b$) for $k = 0, \ldots, K-1$. The former strategy generally yields better predictions but at a higher computational cost, while the latter is more efficient but also more prone to overfitting.

2) Split Requirements: Nodes must hold at least one itb and one oob sample to apply aggregation with exponential weights, see Section II-C below. A split is discarded if it leads to children with less than $n_{\text{min-leaf}}$ itb or oob samples and we do not split a node with less than $n_{\text{min-split}}$ itb or oob samples. These hyperparameters only weakly impact the performance of the boosting processes and sticking to default values ($n_{\text{min-leaf}} = 1$ and $n_{\text{min-split}} = 2$, following scikit-learn’s [2], [57]) is usually enough (see Section F below).

3) Related Works on Categorical Splits: In [58], an interesting characterization of an optimal categorical split for multiclass classification is introduced, but no efficient algorithm is, to the best of our understanding, available for it. A heuristic algorithm is proposed therein, but it requires to compute, for each split, the top principal component of the covariance matrix of the conditional distribution of labels given bins, which is computationally too demanding for an RF algorithm intended for large datasets. Regularized target encoding is shown in [59] to perform best when compared with many alternative categorical encoding methods. Catboost [18] uses target encoding, which replaces feature modalities by label statistics, so that a natural bin order can be used for split finding. To avoid overfitting on uninformative categorical features, a debiasing technique uses random permutations of samples and computes the target statistic of each element based only on its predecessors in the permutation. However, for multiclass classification, target encoding is influenced by the arbitrarily chosen ordinal encoding of the labels. LightGBM [17] uses a one-versus-rest strategy, which is also one of the approaches used in WW for categorical splits on multiclass tasks. For categorical splits, where bin order depends on labels statistics, WW does not use debiasing as in [18], since aggregation with exponential weights computed on oob samples allows to deal with overfitting.

4) Tree Growth Stopping: We do not split a node and make it a leaf if it contains less than $n_{\text{min-split}}$ itb or oob samples. The same applies when a node’s impurity is not larger than a threshold $\varepsilon$ ($\varepsilon = 0$ by default). When only leaves or non-splittable nodes remain, the growth of the tree is stopped. Trees grow in a depth-first fashion so that chilths $v0$ and $v1$ have memory indexes larger than their parent $v$ (as required by Algorithm 1 below). In practice, trees are grown to full depth in WildWood (i.e. we take $n_{\text{min-split}} = 2$) since we can only benefit from additional splits and do not need to worry about overfitting since it is prevented by the subtrees aggregation we present now.

C. Prediction Function: Aggregation With Exponential Weights

Given a tree $T$ grown as described in Sections II-A and II-B, its prediction function is an aggregation of the predictions given by all possible subtrees rooted at root, denoted $\{T : T \subset T\}$. Figure 4 provides an example of a tree and two such subtrees. While $T$ is grown using itb samples, we use oob samples to perform aggregation with exponential weights, with a branching process prior over subtrees. Thus, the aggregation mechanism gives more importance to shallow subtrees with a good predictive oob performance. The balance between shallowness and prediction quality is modulated through a temperature parameter denoted $\eta$ in Equation (5) below.

1) Node and Subtree Prediction: We define $v_T(x) \in \text{leaves}(T)$ as the leaf of $T$ containing $x \in C$. The prediction of a node $v \in \text{nodes}(T)$ and of a subtree $T \subset T$ is given by

$$\hat{y}_v = h((y_i)_{i \in I_{\text{itb}} : x_i \in C_v}) \quad \text{and} \quad \hat{y}_T(x) = \hat{y}_{v_T(x)}. \quad (1)$$

where $h : \mathbb{N} \rightarrow \mathbb{R}$ is a generic “forecaster” used in each cell and where a subtree prediction is that of its leaf containing $x$. A standard choice for regression ($\hat{Y} = \hat{Y} = \mathbb{R}$)
is the empirical mean forecaster

\[ \hat{y}_v = \frac{1}{n_v} \sum_{i \in I_{\text{tbb}}} y_i, \]

(2)

where \( n_v = |\{i \in I_{\text{tbb}} : x_i \in C_v\}| \). For \( K \)-class classification with \( Y = \{1, \ldots, K\} \) and \( \hat{Y} = \mathcal{P}(Y) \), the set of probability distributions over \( Y \), a standard choice is a Bayes predictive posterior with a prior on \( \mathcal{P}(Y) \) equal to the Dirichlet distribution \( \text{Dir}(\alpha, \ldots, \alpha) \), namely the Jeffreys prior on the multinomial model \( \mathcal{P}(Y) \), which leads to

\[ \hat{y}_v(k) = \frac{n_v(k) + \alpha}{n_v + \alpha K}, \]

(3)

for any \( k \in Y \), where \( n_v(k) = |\{i \in I_{\text{tbb}} : x_i \in C_v, y_i = k\}| \). By default, WW uses \( \alpha = 1/2 \) (the Krichkovsky-Trofimov forecaster [60]), but one can perfectly use any \( \alpha > 0 \), so that all the coordinates of \( \hat{y}_v \) are positive. This is motivated by the fact that WW uses as default the log loss to assess \textit{oob} performance for classification, which requires an arbitrarily chosen clipping value for zero probabilities. Different choices of \( \alpha \) only weakly impact WW’s performance, as illustrated below.

2) \textit{Prediction Function:} Let \( x \in C \). The prediction function \( \hat{f} \) of a tree \( T \) in WW is given by

\[ \hat{f}(x) = \frac{\sum_{T \subset T} \pi(T)e^{-\eta L_T} \hat{y}_T(x)}{\sum_{T \subset T} \pi(T)e^{-\eta L_T}}, \]

(5)

with \( \pi(T) = 2^{-|T|} \),

where the sum is over all subtrees \( T \) of \( T \) rooted at \( \text{root} \), where \( \eta > 0 \) is a temperature parameter and \( |T| \) is the number of nodes in \( T \) minus its number of leaves that are also leaves of \( T \). A default choice for the temperature is \( \eta = 1 \) for the log-loss (see in particular Corollary 1 in Section III below), but it can also be tuned through hyperoptimization, although we do not observe strong performance gains, see Appendix F below.

The prediction function (5) corresponds to an exponentially weighted average over the predictors \( \hat{y}_T(x) \) for \( T \subset T \) rooted at \( \text{root} \) based on \textit{oob} performance and the branching process prior \( \pi \) with branching probability \( 1/2 \). The latter being expressly chosen for its fitness within this paradigm. This definition falls within the framework of PAC-Bayesian theory [61], [62], [63] with theoretical guarantee of the oracle inequality for \( \hat{f} \) on \textit{oob} (see Theorem 2 below). This aggregation procedure can be understood as a \textit{non-greedy way to prune trees}: the weights depend not only on the quality of one single split but also on the performance of each subsequent split.

Naively computing \( \hat{f} \) from Equation (5) is computationally and memory-wise infeasible for a large \( T \), since it involves a sum over all \( T \subset T \) rooted at \( \text{root} \) and requires one weight for each \( T \). Indeed, the number of subtrees of a minimal tree that separates \( n \) points is exponential in the number of nodes, and hence exponential in \( n \). However, it turns out that one can compute \( \hat{f} \) exactly and very efficiently thanks to the prior choice \( \pi \) together with an adaptation of context tree weighting [4], [7], [8], [9].

\textit{Theorem 1:} The prediction function (5) can be written as \( \hat{f}(x) = \hat{f}_{\text{root}}(x) \), where \( \hat{f}_{\text{root}}(x) \) satisfies the recursion

\[ \hat{f}_v(x) = \frac{1}{2w_v^\text{den}}\hat{y}_v + \left(1 - \frac{1}{2w_v^\text{den}}\right)\hat{f}_{va}(x) \]

(6)

for \( v, va \in \text{path}(x) \) (\( a \in \{0, 1\} \)) the path in \( T \) going from \( \text{root} \) to \( v \), where \( w_v := \exp(-\eta L_v) \) with \( L_v := \sum_{i \in I_{\text{tbb}}, x_i \in C_v} \ell(\hat{y}_i, y_i) \) and where \( w_v^\text{den} \) are weights satisfying the recursion

\[ w_v^\text{den} = \begin{cases} w_v & \text{if } v \in \text{leaves}(T), \\ \frac{1}{2}w_v + \frac{1}{2}w_v^\text{den} & \text{otherwise} \end{cases} \]

(7)

The proof of Theorem 1 is given in Section A below, a consequence of this Theorem being a very efficient method to compute \( \hat{f}(x) \) described in Algorithms 1 and 2 below. Algorithm 1 computes the weights \( w_v^\text{den} \) using the fact that trees in WW are grown in a depth-first fashion, so that we can loop once, leading to a \( O(\text{nodes}(T)) \) complexity in time and in memory usage, over nodes from a data structure that respects the parenthood order. Direct computations can lead to numerical over- or under-flows (many products of exponentially small or large numbers are involved), so Algorithm 1 works recursively over the logarithms of the weights (line 6 uses a log-sum-exp function that can be made overflow-proof). Algorithm 1 is applied once \( T \) is fully grown, so that WW is ready to produce predictions using Algorithm 2 below. Note that hyperoptimization of \( \eta \) or \( \alpha \), if required, does not need to grow \( T \) again, but only to update \( w_v^\text{den} \) for all \( v \in \text{nodes}(T) \) with Algorithm 1, making hyperoptimization of these parameters particularly efficient. The recursion used in Algorithm 2 has a complexity \( O(\text{path}(x)) \) which is the complexity required to find the leaf \( v_T(x) \) containing \( x \in C \): Algorithm 2 only increases by a factor 2 the prediction complexity of a standard RF (in order to go down to \( v_T(x) \) and up again to \( \text{root} \) along \( \text{path}(x) \)). More details about the construction of Algorithms 1 and 2 can be found in Section A below.
Algorithm 1 Computation of log($w_v^{\text{den}}$) for All v $\in$ nodes($T$)

1: **Inputs:** $T$, $\eta > 0$ and losses $L_v$ for all $v \in$ nodes($T$).
Nodes from nodes($T$) are stored in a data structure nodes that respects parenthesis order: for any $v = \text{nodes}[i_v] \in$ intnodes($T$) and children $va = \text{nodes}[i_{va}]$ for $a \in \{0, 1\}$, we have $i_v > i_v$.
2: for $v \in \text{reversed}(\text{nodes})$ do
3: if $v$ is a leaf then
4: Put $\log(w_v^{\text{den}}) \leftarrow -\eta L_v$
5: else
6: Put $\log(w_v^{\text{den}}) \leftarrow \log\left(\frac{1}{2}e^{\log(w_v^{\text{den}}) + \log(w_{\text{left}}^{\text{den}})} + \frac{1}{2}e^{-\eta L_{\text{left}}}\right)$
7: end if
8: end for
9: return The set of log-weights $\{\log(w_v^{\text{den}}) : v \in$ nodes($T$)\}

Algorithm 2 Computation of $\hat{f}(x)$ for Any $x \in C$

1: **Inputs:** Tree $T$, losses $L_v$ and log-weights $\log(w_v^{\text{den}})$ computed by Algorithm 1
2: Find $v_T(x) \in \text{leaves}(T)$ (the leaf containing $x$) and put $v \leftarrow v_T(x)$
3: Put $\hat{f}(x) \leftarrow \hat{y}_v$ (the node $v$ forecaster, such as (2) for regression or (3) for classification)
4: while $v \neq \text{root}$ do
5: Put $v \leftarrow \text{parent}(v)$
6: Put $\alpha \leftarrow \frac{1}{2}e^{-\eta L_v - \log(w_v^{\text{den}})}$
7: Put $\hat{f}(x) \leftarrow \alpha \hat{y}_v + (1 - \alpha)\hat{f}(x)$
8: end while
9: return The prediction $\hat{f}(x)$

III. THEORETICAL GUARANTEES

This section proposes some theoretical guarantees on the subtrees aggregation (5) used in WW. We say that a loss function $\ell$ is $\eta$-exp-concave for some $\eta > 0$ whenever $z \mapsto \exp(-\eta g(z, y))$ is concave for any $y \in \mathcal{Y}$. We consider a fully-grown tree $T$ computed using $B$ samples and the set of $oob$ samples $(x_i, y_i)_{i \in I_{oob}}$ on which $L_T$ is computed using (4), and we denote $n_{oob} := |I_{oob}|$.

Theorem 2 (Oracle Inequality): Assume that the loss function $\ell$ is $\eta$-exp-concave. Then, the prediction function $\hat{f}$ given by (5) satisfies the oracle inequality

$$\frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(\hat{f}(x_i), y_i) \leq \inf_{T \in \mathcal{T}} \left\{ \frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(g_T(x_i), y_i) + \frac{\log 2}{n_{oob} + 1} \right\},$$

where the infimum is over any subtree $T \subset \mathcal{T}$ and where we recall that $|T|$ is the number of nodes in $T$ minus its number of leaves that are also leaves of $T$.

Theorem 2 proves that, for a general loss function, the prediction function of WW is able to perform nearly as well as the best oracle subtree $T \subset \mathcal{T}$ on $oob$ samples, with a $O(|T|/n_{oob})$ rate which is optimal for model-selection oracle inequalities [64] ($|T| = O(\log N_T)$ with a number of “experts” $N_T = |\{T : T \subset \mathcal{T}\}|$ for a well-balanced $T$). Let us stress again that, while finding an oracle $\arg\min_{T \in \mathcal{T}} \sum_{i \in I_{oob}} \ell(\hat{y}_T(x_i), y_i)$ is computationally infeasible, since it requires to try out all possible subtrees, WW’s prediction function (5) comes at a cost comparable to that of a standard Random Forest, as explained in Section II-C above.

The proof of Theorem 2 is given in Section B below and relies on techniques from PAC-Bayesian theory [61], [62], [63]. The arguments hinge upon the mixability of the loss, a consequence of its exp-concavity, as well as the Donsker-Varadhan variational formula. Note that the statement concerns only the performance of a single tree and not the whole forest. Indeed, extrapolating the benefit brought by the exponential weights’ aggregation within each tree to the forest itself remains a difficult open problem even though it is intuitive enough that improved individual trees should enhance the overall performance. Compared with [10] about online learning, our proof differs in that we do not use results specialized to online learning such as [65] nor online-to-batch conversion [66]. Note that Theorem 2 does not address the generalization error, since it would require to study the generalization error of the random forest itself (and of the fully grown tree $T$), which is a topic way beyond the scope of this paper, and still a very difficult open problem: recent results [41], [42], [43], [67], [68], [69] only study stylized versions of RF (called purely random forests).

Consequences of Theorem 2 are Corollary 1 for the log-loss (classification) and Corollary 2 for the least-squares loss (regression).

**Corollary 1 (Classification):** Consider $K$-class classification ($\mathcal{Y} = \{1, \ldots, K\}$) and consider the prediction function $\hat{f}$ given by (5), where node predictions are given by (3) with $\alpha = 1/2$ (WW’s default), where $\ell$ is the log-loss and where $\eta = 1$. Then, we have

$$\frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(\hat{f}(x_i), y_i) \leq \inf_{T \in \mathcal{T}} \left\{ \frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(g_T(x_i), y_i) + \frac{K + 4\log 2 - 1}{4} \left\| T \right\| + 1 \right\},$$

where $g_T$ is any constant function on the leaves of $T$.

**Corollary 2 (Regression):** Consider regression with $\mathcal{Y} = [-B, B]$ for some $B > 0$ and the prediction function $\hat{f}$ given by (5), where node predictions are given by (2), where $\ell$ is the least-squares loss and where $\eta = 1/(8B^2)$. Then, we have

$$\frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(\hat{f}(x_i), y_i) \leq \inf_{T \in \mathcal{T}} \left\{ \frac{1}{n_{oob}} \sum_{i \in I_{oob}} \ell(g_T(x_i), y_i) + 8(\log 2)B^2 \right\},$$

where $g_T$ is any function constant on the leaves of $T$.

The proofs of Corollaries 1 and 2 are given in Section B. These corollaries motivate the default hyperparameter values of $\eta$, in particular $\eta = 1$ for classification.

IV. EXPERIMENTS

Our implementation of WildWood is available at the GitHub repository https://github.com/
The page contains a review of machine learning algorithms, focusing on optimization techniques and performance metrics. The text is divided into sections covering different aspects of algorithm performance and training times.

### A. Performance on Classification Tasks

For each dataset and algorithm, the performance after tuning the hyperparameters is evaluated. Hyperoptimization is performed as follows: from the training set, we use 4/5 for training and 1/5 for validation and do 50 steps of sequential optimization using the Tree Parzen Estimator implemented in the hyperopt library [73]. More details about hyperoptimization are provided in Section D below. Then, we refit on the whole training set with the best hyperparameters and report scores on the test set. This is performed 5 times in order to report standard deviations. We use the area under the ROC curve (AUC) for $K$-class datasets with $K > 2$ to average the AUC of each class versus the rest. This leads to the test AUC scores displayed in Table I (the same scores with standard deviations are available in Table III).

### B. Training Time

We provide a study of training times over an increasing fraction of the same datasets in Figure 5. We report also in Table II (see also Table V for standard deviations) the test AUC scores obtained with default hyperparameters of all algorithms on the 5 largest considered datasets together with their training times (timings can vary by several orders of magnitude with varying hyperparameters for EGB libraries, as observed by the timing differences between Figure 3 and Table II). Figure 5 further highlights the swiftness of WildWood’s training procedure as compared to other models when trained on increasing fractions of the same dataset. Paradoxically, certain boosting algorithms appear to train faster on bigger data fractions, we attribute this to their stopping criteria on default parameters.

The results on both Tables I and II show that WW is competitive with respect to all baselines both in terms of performance and computational times: it manages to always reach at least comparable performance with the best algorithms despite only using 10 trees as a default. In this respect, WW maintains high scores at a lower computational cost.

### C. Model Size

We also train random forest models with number of trees from one to ten (and default parameters for the rest) on a few datasets using random forest algorithms (WW and scikit-learn’s implementations of RF [2] and Extra Trees [15]). We plot the mean test AUC over 10 repetitions against model size measured through the associated pickle file in megabytes. The result is displayed on Figure 6 and demonstrates WildWood’s ability to offer lightweight random forest models with improved performance. This is a valuable advantage for applications using learning algorithms with limited memory and computational resources. Notable examples include embedded systems [74], [75], [76] or IoT devices [77], [78], [79].

### D. Regression Experiment

We run an elementary regression experiment comparing WW to RF and Extra Trees again. The task is to recover four noisy test signals: Doppler, Heavisine, Blocks and Bumps...
TABLE I
TEST AUC OF ALL ALGORITHMS AFTER HYPEROPTIMIZATION ON THE CONSIDERED DATASETS. STANDARD-DEVIATIONS ARE REPORTED IN TABLE III AND RESULTS IN TERMS OF LOG-LOSS IN TABLE IV. WE OBSERVE THAT WW HAS BETTER (OR IDENTICAL IN SOME CASES) PERFORMANCES THAN RF ON ALL DATASETS AND THAT IT IS CLOSE TO THAT OF EGB LIBRARIES (BOLD IS FOR BEST EGB PERFORMANCE, UNDERLINE FOR BEST RFn, WWn OR WW TRAINING TIME). STANDARD DEVIATIONS ARE REPORTED IN TABLE V.

|                | XGB | LGBM | CB  | HGB | RF10 | RF100 | WW10 | WW100 | RD10 | RD100 |
|----------------|-----|------|-----|-----|------|-------|------|-------|------|-------|
| adult          | 0.930 | 0.931 | 0.927 | 0.930 | 0.915 | 0.918 | 0.916 | 0.919 | 0.915 | 0.917 |
| bank           | 0.933 | 0.935 | 0.925 | 0.930 | 0.919 | 0.929 | 0.926 | 0.931 | 0.919 | 0.922 |
| breastcancer   | 0.991 | 0.993 | 0.987 | 0.994 | 0.987 | 0.989 | 0.992 | 0.992 | 0.987 | 0.985 |
| car            | 1.000 | 1.000 | 1.000 | 1.000 | 0.997 | 0.998 | 0.998 | 0.998 | 0.993 | 0.993 |
| covtype        | 0.999 | 0.999 | 0.998 | 0.999 | 0.997 | 0.998 | 0.996 | 0.998 | 0.974 | 0.996 |
| default-cb     | 0.780 | 0.783 | 0.780 | 0.779 | 0.765 | 0.775 | 0.774 | 0.778 | 0.772 | 0.773 |
| higgs          | 0.853 | 0.857 | 0.847 | 0.853 | 0.820 | 0.837 | 0.820 | 0.837 | 0.813 | 0.815 |
| internet       | 0.934 | 0.910 | 0.938 | 0.911 | 0.917 | 0.935 | 0.926 | 0.928 | 0.925 | 0.928 |
| kddcup         | 1.000 | 1.000 | 1.000 | 1.000 | 0.997 | 0.998 | 1.000 | 1.000 | 1.000 | 1.000 |
| kick           | 0.777 | 0.770 | 0.777 | 0.771 | 0.749 | 0.764 | 0.756 | 0.763 | 0.752 | 0.754 |
| letter         | 1.000 | 1.000 | 1.000 | 1.000 | 0.997 | 0.999 | 0.997 | 0.999 | 0.995 | 0.993 |
| satimage       | 0.991 | 0.991 | 0.991 | 0.987 | 0.985 | 0.991 | 0.986 | 0.991 | 0.984 | 0.983 |
| sensorless     | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| spambase       | 0.990 | 0.990 | 0.987 | 0.986 | 0.980 | 0.987 | 0.983 | 0.987 | 0.972 | 0.974 |

Fig. 5. Training times on increasing fractions of a few large datasets for WildWood and the other baselines of Table II (except for CatBoost which is not competitive). The results are averaged over 5 runs using default parameters for each algorithm. WildWood clearly outpaces its competitors in almost all cases.

originally analyzed by [80] and [81]. We test different noise intensities measured through the signal-to-noise ratio (SNR) and show the results on Figure 7. We observe that WildWood clearly outperforms its competitors on the signal recovery tasks at low SNR levels. This is thanks to the regularizing effect of aggregation making the algorithm more robust to noise. We note, nonetheless, that this may slightly degrade performance on relatively irregular signals at high SNR (Bumps).

E. Decision Boundaries
Finally, we provide an additional illustration of the interpretability advantage of WW over RF similar to Figure 1
but with real data. For this purpose, we plot the decision boundaries obtained on simple binary classification datasets for which good performance can be attained using only a pair of features. In each dataset, we choose such a pair among the most important features as measured by the Mean Decrease in Impurity (MDI) and fit WW and RF on the datasets restricted to these features. The results are shown on Figure 8. As before, we observe that WW yields more regular decision boundaries making its model predictions less prone to overfitting and more interpretable.

V. CONCLUSION

We introduced WildWood, a new Random Forest algorithm for batch supervised learning. Tree predictions in WildWood are aggregation with exponential weights of the predictions of all subtrees, with weights computed on bootstrap out-of-bag samples. This leads to improved predictions in each individual tree, at a small computational cost, since WildWood’s prediction complexity is similar to that of a standard Random Forest. Moreover, thanks to the histogram strategy, WildWood’s implementation is competitive with strong baselines including popular extreme boosting libraries, both in terms of performance and training times. Note also that WildWood has few hyperparameters to tune and that the performances obtained with default hyperparameters are usually good enough in our experiments.

WildWood’s implementation is still evolving and many improvements coming with future updates are planned, including the computation of feature importance, GPU training, distributed training (we only support single-machine training for now), among other enhancements that will further improve performances and accelerate computations. Room for improvement in WildWood comes from the fact that the overall forest prediction is a simple arithmetic mean of each tree prediction, while we could perform also exponentially weighted aggregation between trees. Future works include a WildWood-based implementation of isolation-forest [82], using the same subtrees aggregation mechanism with the log loss for density estimation, to propose a new algorithm for outliers detection.

APPENDIX A

PROOF OF THEOREM 1 AND CONSTRUCTION OF ALGORITHMS 1 AND 2

The expression in Equation (5) involves sums over all subtrees $T$ of the fully grown tree $T$ (involving an exponential in the number of leaves of $T$). However, it can be computed efficiently because of the specific choice of the prior $\pi$. More precisely, we will use the following lemma [9, Lemma 1] several times to efficiently compute sums of products. Let us recall that $\text{nodes}(T)$ stands for the set of nodes of $T$.

**Lemma 1:** Let $g : \text{nodes}(T) \to \mathbb{R}$ be an arbitrary function and define $G : \text{nodes}(T) \to \mathbb{R}$ as

$$G(v) = \sum_{T' \subseteq T} 2^{-\|T\|} \prod_{v' \in \text{leaves}(T')} g(v'),$$

(8)

where the sum over $T' \subseteq T_0$ means the sum over all subtrees $T'$ of $T$ rooted at $v$. Then, $G(v)$ can be computed recursively as follows:

$$G(v) = \begin{cases} g(v) & \text{if } v \in \text{leaves}(T) \\ \frac{1}{2}g(v) + \frac{1}{2}G(v_0)G(v_1) & \text{otherwise} \end{cases}$$

for each node $v \in \text{nodes}(T)$.
it can be shown that such a pruning of prunings efficiently compute and update the numerator and denominator so that Equation (5) writes

This concludes the proof of Lemma 1.

For the sake of completeness, we include a proof of this statement.

Proof: First, let us notice that the case \( v \in \text{leaves}(T) \) is straightforward since there is only one pruning \( T \) of \( T_v \) which satisfies \( ||T|| = 0 \) (recall that \( ||T|| \) is the number of internal nodes and leaves in \( T \) minus the number of leaves in \( T \) that are also leaves of \( T_v \)). For the second case, we can expand \( G(v) \) by taking into account the pruning which only leaves \( v \) as a leaf, the rest of the prunings can be expressed through pairs of prunings \( T_0 \) and \( T_1 \) of \( T_v \) respectively. Moreover, it can be shown that such a pruning \( T \) satisfies \( ||T|| = 1 + ||T_0|| + ||T_1|| \), thus we get the following expansion:

\[
G(v) = \frac{1}{2} g(v) + \sum_{T_0 \subset T_v, T_1 \subset T_v} 2^{-(1+||T_0||+||T_1||)} \prod_{v' \in T_0} g(v_0v') \prod_{v'' \in T_1} g(v_1v'')
\]

\[
= \frac{1}{2} g(v) + \frac{1}{2} \left( \sum_{T_0 \subset T_v} 2^{-||T_0||} \prod_{v' \in T_0} g(v_0v') \right) \cdot \left( \sum_{T_1 \subset T_v} 2^{-||T_1||} \prod_{v'' \in T_1} g(v_1v'') \right)
\]

\[
= \frac{1}{2} g(v) + \frac{1}{2} G(v_0)G(v_1).
\]

This concludes the proof of Lemma 1.

Let us introduce \( w_T = \pi(T) \exp(-\eta L_T) \) for any \( T \subset T \), so that Equation (5) writes

\[
\hat{f}(x) = \frac{\sum_{T \subset T} w_T \hat{y}_T(x)}{\sum_{T \subset T} w_T},
\]

where the sums hold over all the subtrees \( T \) of \( T \) rooted at root (the root of the full tree \( T \)). We will show how to efficiently compute and update the numerator and denominator in Equation (9). Note that \( w_T \) may be written as

\[
w_T = \pi(T) \exp(-\eta L_T)
\]

\[
= 2^{-||T||} \exp\left(-\eta \sum_{i \in I_oob} \ell(\hat{y}_{v_T}(x_i), y_i)\right)
\]

\[
= 2^{-||T||} \exp\left(-\eta \sum_{v \in \text{leaves}(T)} \sum_{i \in I_oob : x_i \in C_v} \ell(\hat{y}_v(x_i), y_i)\right)
\]

\[
= 2^{-||T||} \exp\left(-\eta \sum_{v \in \text{leaves}(T)} \sum_{i \in I_oob : x_i \in C_v} \ell(\hat{y}_v, y_i)\right)
\]

\[
= 2^{-||T||} \prod_{v \in \text{leaves}(T)} \exp(-\eta L_v)
\]

Equality (10) comes from the fact that the set of cells \( \{C_v : v \in \text{leaves}(T)\} \) is a partition of \( C \) by construction, and that the stopping criterion used to build \( T \) ensures that each leaf node in \( \text{leaves}(T) \) contains at least one sample from \( I_oob \) (see Section II-B). Equality (11) comes from the fact that the prediction of a node is constant and equal to \( \hat{y}_v \) for any \( x \in C_v \).

A. Denominator of Equation (9)

For each node \( v \in \text{nodes}(T) \), denote

\[
w_v^{\text{den}} = \sum_{T \subset T_v} 2^{-||T||} \prod_{v' \in \text{leaves}(T)} w_{v'},
\]
where once again the sum over \( T \subset \mathcal{T}_v \) means the sum over all subtrees \( T \) of \( \mathcal{T} \) rooted at \( v \). We have that (12) entails
\[
w_{\text{root}}^\text{den} = \sum_{T \subset \mathcal{T}_\text{root}} 2^{-\|T\|} \prod_{v \in \text{leaves}(T)} w_v
= \sum_{T \subset \mathcal{T}_\text{root}} w_T = \sum_{T \subset \mathcal{T}} w_T.
\]
(14)

So, we can compute recursively \( w_{\text{root}}^\text{den} \) very efficiently, using a recursion on the weights \( w_{\text{root}}^\text{den} \), with Lemma 1 with \( g(v) = w_v \). This leads to the recursion stated in Theorem 1, see Equation (7).

Now, we can exploit the fact that decision trees are built in a depth-first fashion in WildWood: all the nodes \( v \in \mathcal{T} \) are stored in a “flat” array, and by construction both the child nodes \( v_0 \) and \( v_1 \) have indexes that are larger than the one of \( v \). So, we can simply loop over the array of nodes in reverse order, and compute \( w_{\text{root}}^\text{den} = w_v \) if \( v \in \text{leaves}(T) \) and \( w_{\text{root}}^\text{den} = \frac{1}{2} w_v + \frac{1}{2} w_{v_0} w_{v_1} \) otherwise: we are guaranteed to have computed \( w_{v_0} \) and \( w_{v_1} \) before computing \( w_v \). This algorithm is described in Algorithm 1. Since these computations involve a large number of products with exponentiated numbers, it typically leads to strong over- and under-flows: we describe in Algorithm 1 a version of this algorithm which works recursively over the logarithms of the weights. At the end of this loop, we end up at \( v = \text{root} \) and have computed \( w_{\text{root}}^\text{den} = \sum_{T \subset \mathcal{T}} w_T \) with a very efficient \( O(\|\text{nodes}(T)\|) \) complexity. Note also that it is sufficient to store both \( w_v \) and \( w_{\text{root}}^\text{den} \) for all \( v \in \mathcal{T} \), which makes for a \( O(\|\text{nodes}(T)\|) \) memory consumption.

B. Numerator of Equation (9)

The numerator of Equation (9) almost follows the exact same argument as the denominator, but since it depends on the input vector \( x \in C \) of features for which we want to produce a prediction, it is performed at inference time. Recall that \( \text{path}(x) \) is the sequence of nodes that leads to the leaf \( \text{v}_{\text{T}}(x) \) containing \( x \in C \) and define, for any \( v \in \text{nodes}(T), \)
\[
\hat{w}_v(x) = w_v \hat{y}_v(x) \quad \text{if} \quad v \in \text{path}(x),
\]
and \( \hat{w}_v(x) = w_v \) otherwise. We have
\[
\sum_{T \subset \mathcal{T}} w_T \hat{y}_T(x) = \sum_{T \subset \mathcal{T}_\text{root}} w_T \hat{y}_{\text{root}}(x)
= \sum_{T \subset \mathcal{T}_\text{root}} 2^{-\|T\|} \prod_{v \in \text{leaves}(T)} w_v \hat{y}_v(x)
= \sum_{T \subset \mathcal{T}_\text{root}} 2^{-\|T\|} \prod_{v \in \text{leaves}(T)} \hat{w}_v(x).
\]
(15)

Note that (15) comes from (12) while (16) comes from the definition of \( \hat{w}_v(x) \) (note that a single term from the product over \( v \in \text{leaves}(T) \) corresponds to \( v = v_{\text{T}}(x) \) since \( \{C_v : v \in \text{leaves}(T)\} \) is a partition of \( C \). We are now in position to use again Lemma 1 with \( g(v) = \hat{w}_v(x) \). Defining
\[
w_v^\text{num}(x) = \sum_{T \subset \mathcal{T}_v} 2^{-\|T\|} \prod_{v' \in \text{leaves}(T)} \hat{w}_{v'}(x),
\]
we can conclude that
\[
w_v^\text{num}(x) = \sum_{T \subset \mathcal{T}} w_T \hat{y}_T(x)
\]
and that the following recurrence holds:
\[
w_v^\text{num}(x) = \begin{cases} w_v \hat{y}_v(x) & \text{if } v \in \text{leaves}(T) \\ \hat{w}_v(x) + \frac{1}{2} w_{v_0} \hat{w}_{v_1}(x) & \text{otherwise} \end{cases}
\]
(18)

This recurrence allows to compute \( w_v^\text{num}(x) \) from \( \hat{w}_v(x) \), but note that a direct use of this formula would lead to a complexity \( O(\|\text{nodes}(T)\|) \) to produce a prediction for a single input \( x \in C \). It turns out we can do much better than that.

Indeed, whenever \( v \notin \text{path}(x) \), we have by definition that \( \hat{w}_v(x) = w_v \) and that \( \hat{w}_v(x) = w_v \) for any descendant \( v' \) of \( v \), which entails by induction that \( w_{v_{\text{den}}}(x) = w_v \) for any \( v \notin \text{path}(x) \). Therefore, we only need to explain how to compute \( w_v^\text{num}(x) \) for \( v \in \text{path}(x) \). This is achieved recursively, thanks to (18), starting at the leaf \( \text{v}_{\text{T}}(x) \) and going up in the tree to \( \text{root} \):
\[
w_v^\text{num}(x) = \begin{cases} w_v \hat{y}_v(x) & \text{if } v = \text{root}(x) \\ \frac{1}{2} w_v \hat{y}_v(x) + \frac{1}{2} w_{v_{\text{den}}(1-a)} w_{v_{\text{den}}(a)} & \text{otherwise} \end{cases}
\]
(19)

where \( a \in \{0, 1\} \) is s.t. \( v \in \text{path}(x) \). Let us explain where this comes from: firstly, one has obviously that \( \text{leaves}(T) \cap \text{path}(x) = \{v_{\text{T}}(x)\} \), so that \( w_v^\text{num}(x) = g(v) = w_v \hat{y}_v(x) \) for \( v = v_{\text{T}}(x) \). Secondly, we go up in the tree along \( \text{path}(x) \) and use again (18): whenever \( v \in \text{intnodes}(T) \) and \( v_{\text{root}} = \text{path}(x) \) for \( a \in \{0, 1\} \), we have \( w_v^\text{num}(x) = w_{v_{\text{den}}(1-a)} \) since \( v(1-a) \notin \text{path}(x) \). This recursion has a complexity \( O(\|\text{path}(x)\|) \) where \( |\text{path}(x)| \) is the number of nodes in \( \text{path}(x) \), and is typically orders of magnitude smaller than \( |\text{nodes}(T)| \) (in a well-balanced binary tree, one has the relation \( |\text{path}(x)| = O(\log_2(\|\text{nodes}(T)\|)) \)). Moreover, we observe that the recursions used in (7) and (19) only need to save both \( w_v \) and \( w_v^\text{den} \) for any \( v \in \text{nodes}(T) \).

Finally, we have using (14) and (17) that
\[
f(x) = \frac{\sum_{T \subset \mathcal{T}} w_T \hat{y}_T(x)}{\sum_{T \subset \mathcal{T}} w_T} = \frac{w_{\text{root}}^\text{root}(x)}{w_{\text{root}}^\text{den}} =: f_{\text{root}}(x),
\]
and we want to compute \( \hat{f}_{\text{root}}(x) \) recursively from \( \hat{f}(x) \) where \( v \in \text{path}(x) \). First, whenever \( v = v_{\text{T}}(x) \) we have
\[
f_v(x) = \frac{w_v^\text{num}(x)}{w_v^\text{den}} = \frac{w_v \hat{y}_v(x)}{w_v} = \hat{y}_v(x),
\]
while for \( v \neq v_{\text{T}}(x) \) and \( v \in \text{path}(x) \), we write
\[
f_v(x) = \frac{w_v^\text{num}(x)}{w_v^\text{den}} = \frac{1}{2} w_v \hat{y}_v(x) + \frac{1}{2} w_{v_{\text{den}}(1-a)} w_{v_{\text{den}}(a)}
= \frac{1}{2} w_v \hat{y}_v(x) + \frac{1}{2} w_{v_{\text{den}}(1-a)} w_{v_{\text{den}}(a)}
\]
(20)

where (20) comes from (19) while (22) comes from (7). This proves the recursion stated in Equation (6) from Theorem 1, and to Algorithm 2. This concludes the proof of Theorem 1.
APPENDIX B

PROOFS OF THE RESULTS FROM SECTION III

The proof of Theorem 2 is partly inspired from the proof of Theorem 2 in [83], that we generalize to exp-concave losses, while only least-squares regression is considered therein. Let \( E \) be a measurable space and \( P, Q \) be probability measures on it. The Kullback-Leibler divergence between \( P \) and \( Q \) is defined by

\[
\text{KL}(P, Q) = \int_E \log \left( \frac{dP}{dQ} \right) dP
\]

whenever \( P \) is absolutely continuous with respect to \( Q \) and equal to \( +\infty \) otherwise. Also, if \( h : E \to \mathbb{R} \) is a measurable function such that \( \int_E h dP \) is well-defined on \( \mathbb{R} \cup \{-\infty, +\infty\} \), we introduce

\[
P_h := \frac{h}{\int_E h dP} \cdot P,
\]

the probability measure on \( E \) with density \( h/\int h dP \) with respect to \( P \). A classical result is the Donsker-Varadhan variational formula [84], which is at the core of the proofs of many PAC-Bayesian theorems [62], [63] and that we use here as well in the proof of Theorem 2. It states that

\[
\log \left( \int_E \exp(h) dQ \right) + \text{KL}(P, Q) = \int h dP = \text{KL}(P, Q_{\exp(h)})
\]

holds for any probability measures \( P \) and \( Q \) on \( E \) and any measurable function \( h : E \to \mathbb{R} \). This entails in particular that

\[
\log \left( \int_E \exp(h) dQ \right) = \sup_{P} \left\{ \int h dP - \text{KL}(P, Q) \right\},
\]

where the supremum is over all probability measures on \( E \), and where the supremum is achieved for \( P = Q_{\exp(h)} \) whenever the term on the left-hand side is finite.

Proof of Theorem 2: Recall that the oob loss of a subtree \( T \subset T \) is given by

\[
L_T = \sum_{i \in I_{oob}} \ell(\hat{y}_T(x_i), y_i)
\]

and let us introduce

\[
p_T = \frac{\pi(T) \exp(-\eta L_T)}{\sum_T \pi(T') \exp(-\eta L_{T'})}
\]

for any subtree \( T \subset T \). First, we use the fact that \( \ell \) is a \( \eta \)-exp-concave loss function, hence \( \eta \)-mixable, see Section 3.3 from [83], which entails, since \( p_T \) is a probability measure over the set of all subtrees \( T \subset T \), that

\[
\ell \left( \sum_T p_T \hat{y}_T(x_i), y_i \right) \leq -\frac{1}{\eta} \log \left( \sum_T p_T \exp(-\eta \ell(\hat{y}_T(x_i), y_i)) \right),
\]

where the sums over \( T \) are over all subtrees \( T \subset T \). Now, summing this inequality over \( i \in I_{oob} \) and using the convexity of the log-sum-exp function leads to

\[
\sum_{i \in I_{oob}} \ell \left( \sum_T p_T \hat{y}_T(x_i), y_i \right) \leq -\frac{1}{\eta} \sum_{i \in I_{oob}} \log \left( \sum_T p_T \exp(-\eta \ell(\hat{y}_T(x_i), y_i)) \right)
\]

where the supremum is over all probability measures on \( \mathcal{I} \). Using (23) with \( h = -\eta \ell(x, y) \), we have

\[
S_{\eta oob} := \frac{1}{n_{\text{ooob}}} \sum_{i \in I_{\text{ooob}}} \ell(f(x_i), y_i) \leq -\frac{1}{\eta} \log \left( \sum_T \pi(T) \exp \left( -\left(n_{\text{ooob}} + 1\right) \rho(T) \right) \right)
\]

Thus

\[
\frac{S_{\eta oob}}{n_{\text{ooob}}} \leq -\frac{1}{\eta(n_{\text{ooob}} + 1)} \log \left( \sum_T \pi(T) \exp \left( -\left(n_{\text{ooob}} + 1\right) \rho(T) \right) \right).
\]

Using (23) with \( h = -(n_{\text{ooob}} + 1) \rho(T) \) and \( Q = \pi \), we have

\[
\log \left( \sum_T \pi(T) \exp \left( -\left(n_{\text{ooob}} + 1\right) \rho(T) \right) \right) = -\sum_T P(T)(n_{\text{ooob}} + 1) \rho(T)
\]

for any probability measure \( P \) over the set of subtrees of \( T \). This leads to

\[
\frac{1}{n_{\text{ooob}}} \sum_{i \in I_{\text{ooob}}} \ell(f(x_i), y_i) \leq \frac{1}{n_{\text{ooob}}} \sum_T P(T)L_T + \frac{1}{\eta(n_{\text{ooob}} + 1)} \text{KL}(P, \pi)
\]

for any \( P \), since \( \text{KL}(P, \pi_{\exp(h)}) \geq 0 \). So for the particular choice \( P = \delta_T \) (the Dirac mass at \( T \)) for any subtree \( T \subset T \), we have

\[
\frac{1}{n_{\text{ooob}}} \sum_{i \in I_{\text{ooob}}} \ell(f(x_i), y_i) \leq \frac{1}{n_{\text{ooob}}} \sum_T P(T)L_T + \frac{1}{\eta(n_{\text{ooob}} + 1)} \log \left( \pi(T)^{-1} \right)
\]

\[
\leq \frac{1}{n_{\text{ooob}}} L_T + \frac{\log 2}{\eta(n_{\text{ooob}} + 1)} \|T\|,
\]

which concludes the proof of Theorem 2.

The proof of Corollary 1 requires the next Lemma.

Lemma 2: Consider classification with \( \mathcal{Y} = \{1, \ldots, K\} \) and a node \( v \in \text{nodes}(T) \). Denote \( \nu_v(k) \) the number of
TABLE III

The Same Test AUC Scores as in Table I of All Algorithms After Hyperoptimization on the Considered Datasets. Standard-Deviation Displayed Between Parentheses Are Computed From 5 Trainings With Different Random Seeds. We Observe That WH Has Better (or Identical in Some Cases) Performances Than RF on All Datasets and That It Is Close to That of EGB Libraries (Bold Is for Best EGB Performance, Underline for Best RFn, Wn or Rdn Performance)

|       | XGB | LGB| CB  | KGB | RF10 | RF100 | Wn10 | Wn100 | Rd10 | Rd100 |
|-------|-----|----|-----|-----|------|-------|------|-------|------|-------|
| adult | 0.930 (2.7e-04) | 0.931 (1.2e-04) | 0.927 (2.9e-04) | 0.930 (2.9e-04) | 0.915 (5.0e-04) | 0.918 (1.7e-04) | 0.916 (4.1e-04) | 0.919 (1.4e-04) | 0.915 (3.4e-04) | 0.917 (1.5e-04) |
| bank  | 0.933 (1.5e-04) | 0.935 (4.1e-05) | 0.925 (6.5e-04) | 0.930 (7.4e-04) | 0.919 (6.0e-04) | 0.929 (2.1e-04) | 0.926 (3.8e-04) | 0.931 (1.6e-04) | 0.919 (9.0e-04) | 0.922 (2.2e-04) |
| breastcancer | 0.991 (4.4e-04) | 0.993 (1.1e-04) | 0.987 (6.7e-03) | 0.994 (0.0e+00) | 0.987 (2.1e-03) | 0.992 (4.1e-04) | 0.989 (2.6e-03) | 0.992 (3.1e-04) | 0.987 (2.9e-03) | 0.985 (3.3e-04) |
| car   | 0.999 (2.3e-04) | 1.000 (3.8e-05) | 1.000 (6.0e-05) | 1.000 (0.0e+00) | 0.997 (1.1e-03) | 0.998 (2.8e-04) | 0.998 (6.3e-04) | 0.998 (1.5e-04) | 0.993 (8.7e-04) | 0.993 (1.4e-04) |
| covtype | 0.999 (7.5e-06) | 0.999 (3.6e-06) | 0.998 (2.5e-05) | 0.999 (5.2e-06) | 0.997 (1.4e-04) | 0.998 (1.4e-04) | 0.996 (1.9e-04) | 0.998 (3.7e-05) | 0.974 (7.6e-04) | 0.996 (4.2e-05) |
| default:cb | 0.780 (3.5e-04) | 0.783 (1.2e-04) | 0.780 (3.9e-04) | 0.779 (6.7e-04) | 0.765 (3.9e-03) | 0.775 (7.4e-04) | 0.774 (4.4e-04) | 0.778 (1.6e-03) | 0.777 (9.0e-04) | 0.773 (3.5e-04) |
| bighs | 0.853 (6.7e-05) | 0.857 (1.8e-05) | 0.847 (2.2e-05) | 0.853 (7.0e-05) | 0.820 (9.9e-05) | 0.837 (4.5e-05) | 0.820 (1.5e-04) | 0.837 (6.4e-05) | 0.813 (1.1e-04) | 0.815 (3.3e-05) |
| internet | 0.934 (2.9e-04) | 0.910 (1.8e-04) | 0.938 (1.3e-03) | 0.911 (1.1e-16) | 0.917 (2.3e-03) | 0.935 (6.1e-04) | 0.926 (1.5e-03) | 0.928 (7.9e-04) | 0.923 (4.5e-04) | 0.928 (8.9e-04) |
| kddcup | 1.000 (6.1e-08) | 1.000 (4.1e-07) | 1.000 (5.8e-07) | 1.000 (5.7e-07) | 0.997 (5.2e-05) | 0.999 (2.1e-03) | 1.000 (1.0e-05) | 1.000 (6.0e-06) | 1.002 (1.6e-05) | 1.000 (6.8e-06) |
| kick  | 0.777 (7.6e-04) | 0.770 (2.8e-04) | 0.777 (6.8e-04) | 0.771 (1.6e-03) | 0.749 (1.1e-03) | 0.764 (7.9e-04) | 0.756 (8.3e-04) | 0.763 (5.4e-04) | 0.755 (9.7e-04) | 0.754 (5.2e-04) |
| letter | 1.000 (1.3e-05) | 1.000 (2.7e-06) | 1.000 (4.1e-06) | 1.000 (2.1e-05) | 0.997 (2.6e-04) | 0.999 (1.2e-04) | 0.997 (3.2e-04) | 0.999 (5.5e-05) | 0.995 (4.8e-04) | 0.993 (2.4e-04) |
| satimage | 0.991 (2.1e-04) | 0.991 (1.6e-05) | 0.991 (2.3e-04) | 0.987 (0.0e+00) | 0.983 (1.3e-03) | 0.992 (3.8e-04) | 0.986 (1.2e-03) | 0.991 (1.7e-04) | 0.984 (6.4e-04) | 0.983 (9.8e-04) |
| sensorless | 1.000 (4.9e-07) | 1.000 (1.4e-07) | 1.000 (4.4e-06) | 1.000 (2.9e-06) | 1.000 (2.3e-05) | 1.000 (1.0e-07) | 1.000 (1.1e-04) | 1.000 (1.4e-05) | 0.999 (5.4e-05) | 1.000 (7.3e-06) |
| spacebase | 0.990 (1.5e-04) | 0.990 (5.2e-05) | 0.987 (1.2e-03) | 0.986 (0.0e+00) | 0.980 (1.8e-03) | 0.987 (2.3e-04) | 0.983 (1.0e-03) | 0.987 (2.2e-04) | 0.972 (1.1e-03) | 0.974 (2.9e-04) |
samples of class $k$ in node $v$. We consider the Krichevsky-Trofimov estimator

$$\hat{y}(k) = \frac{n_v(k) + 1/2}{n_v + K/2}$$

where $n_v = \sum_{k=1}^{K} n_v(k)$ and the log loss $\ell(y', y) = -\log y(y')$. Then, the following inequality holds

$$\sum_i \ell(\hat{y}, y_i) - \inf_p \sum_i \ell(p, y_i) \leq \frac{K-1}{2}.$$

**Proof:** We know that the optimal $p$ is given by $p_k = n_v(k)/n_v$. Indeed, it is the solution to the following constrained convex optimization problem

$$\min_p -\sum_{k=1}^{K} n_v(k) \log p_k \quad \text{subject to} \quad \sum_{k=1}^{K} p_k = 1,$$

where we consider non-negativity to be already enforced by the objective and imposing $p_k \leq 1$ is redundant with the constraint $\sum_{k=1}^{K} p_k = 1$. We can write the Lagrangian function as

$$L(p, \lambda) = -\sum_{k=1}^{K} n_v(k) \log p_k + \lambda \left( \sum_{k=1}^{K} p_k - 1 \right)$$

and one can check the KKT conditions when taking $p_k = n_v(k)/n_v$ and $\lambda = n_v$. Since we are dealing with a convex problem with linear constraints, this is a sufficient optimality condition. Straightforward computations give

$$\sum_i \ell(\hat{y}, y_i) - \inf_p \sum_i \ell(p, y_i) = \sum_{k=1}^{K} -n_v(k) \log \left( \frac{n_v(k) + 1/2}{n_v + K/2} \right) - \sum_{k=1}^{K} -n_v(k) \log \frac{n_v(k)}{n_v}$$

$$= \sum_{k=1}^{K} -n_v(k) \left( \log \frac{n_v}{n_v + K/2} + \log \frac{n_v(k) + 1/2}{n_v} \right) - \log \frac{n_v(k)}{n_v}$$

$$= -n_v \log \frac{n_v}{n_v + K/2} + \sum_{k=1}^{K} -n_v(k) \left( \log \frac{n_v(k) + 1/2}{n_v} \right) - \log \frac{n_v(k)}{n_v}$$

$$= n_v \log \frac{n_v + K/2}{n_v} + \sum_{k=1}^{K} n_v(k) \log \frac{n_v(k)}{n_v(k) + 1/2}.$$

Now, using the concavity of the logarithm gives

$$\sum_i \ell(\hat{y}, y_i) - \inf_p \sum_i \ell(p, y_i) \leq n_v \log \frac{n_v + K/2}{n_v} + n_v \log \left( \sum_{k=1}^{K} \frac{n_v(k)}{n_v} \frac{n_v}{n_v(k) + 1/2} \right),$$

and the fact that $x \mapsto x/(x + 1/2)$ is non-decreasing and $n_v(k) \leq n_v$ leads to

$$\sum_i \ell(\hat{y}, y_i) - \inf_p \sum_i \ell(p, y_i) \leq n_v \log \frac{n_v + K/2}{n_v} + n_v \log \frac{n_v}{n_v + 1/2}$$

$$= n_v \log \frac{n_v + 1/2 + (K - 1)/2}{n_v + 1/2}$$

$$= n_v \log \left( 1 + \frac{K - 1}{2n_v + 1} \right) \leq \frac{K - 1}{2}. $$

This concludes the proof of Lemma 2. □

**Proof of Corollary 1:** The log-loss is trivially 1-exp-concave, so that we can choose $\eta = 1$. Following Theorem 2, it remains to bound the regret of the tree forecaster $T$ with respect to the optimal labeling of its leaves. For classification and the log loss, we use Lemma 2 to obtain

$$\sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(\hat{y}_T(x_i), y_i) - \inf_p \sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(p, y_i) \leq \frac{K - 1}{2}$$

for any subtree $T$ and any $v \in \text{leaves}(T)$. Now, summing over $v \in \text{leaves}(T)$, of cardinality $(|T| + 1)/2$, leads to

$$\sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(\hat{y}_T(x_i), y_i) - \sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(p, y_i) \leq |T| \log 2 + \frac{(K - 1)(|T| + 1)}{4}, $$

which concludes the proof of Corollary 1. □

**Proof of Corollary 2:** The square loss is $1/(8B^2)$-exp-concave on $[-B, B]$, see [85], so we can choose $\eta = 1/(8B^2)$. Following Theorem 2, it remains to bound the regret of the tree forecaster $T$ with respect to the optimal labeling of its leaves. For regression with the least-squares loss, since we use the empirical mean forecaster (2), we have

$$\sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(\hat{y}_T(x_i), y_i) - \inf_b \sum_{i \in I \text{ and } x_i \in \text{leaves}(v)} \ell(b, y_i) = 0$$

for any subtree $T$ and any leaf $v \in \text{leaves}(T)$. The rest of the proof follows that of Corollary 1. □

**APPENDIX C**

**SUPPLEMENTARY DETAILS ON EXPERIMENTS**

We report in Table III the same test AUC scores as in Table I of all algorithms after hyperoptimization on the considered datasets. Standard-deviations displayed between parentheses are computed from 5 trainings with different random seeds. We observe that WW has better (or identical in some cases) performances than RF on all datasets and that it is close to that of EGB libraries (bold is for best EGB performance, underline for best RF’s, With or RDN performance). Table IV displays the results of the same experiment measured using the log loss.

We report also in Table V the same training time and test AUC as in Table II, with standard-deviations displayed between parentheses computed from 5 trainings with different random seeds, all with default hyperparameters of each
TABLE IV

THE RESULTS OF TABLE I MEASURED THROUGH LOG-LOSS CORRESPONDING TO ALL ALGORITHMS AFTER HYPEROPTIMIZATION ON THE CONSIDERED DATASETS. WE OBSERVE THAT WW OFTEN IMPROVES THE PERFORMANCE OF RF OR ACHIEVES A CLOSE RESULT (SMALLER IS BETTER, BOLD IS FOR BEST EGB PERFORMANCE, UNDERLINE FOR BEST RFn, WWn OR RDn PERFORMANCE).

|             | XGB | LGBM | CB | HGB | RF10 | RF100 | WW10 | WW100 | RD10 | RD100 |
|-------------|-----|------|----|-----|------|-------|------|-------|------|-------|
| adult       | 0.273 | 0.271 | 0.281 | 0.275 | 0.302 | 0.294 | 0.296 | 0.293 | 0.299 | 0.296 |
| bank        | 0.200 | 0.196 | 0.208 | 0.202 | 0.214 | 0.207 | 0.206 | 0.201 | 0.213 | 0.210 |
| breastcancer| 0.126 | 0.101 | 0.138 | 0.119 | 0.155 | 0.122 | 0.135 | 0.126 | 0.164 | 0.176 |
| car         | 0.048 | 0.051 | 0.051 | 0.015 | 0.163 | 0.085 | 0.078 | 0.078 | 0.194 | 0.190 |
| covtype     | 0.087 | 0.075 | 0.126 | 0.079 | 0.164 | 0.118 | 0.141 | 0.123 | 0.495 | 0.233 |
| default-cb  | 0.430 | 0.429 | 0.430 | 0.431 | 0.439 | 0.432 | 0.434 | 0.432 | 0.435 | 0.435 |
| higgs       | 0.472 | 0.467 | 0.481 | 0.484 | 0.517 | 0.499 | 0.518 | 0.499 | 0.525 | 0.523 |
| internet    | 1.446 | 1.552 | 1.413 | 1.573 | 1.564 | 1.450 | 1.505 | 1.520 | 1.561 | 1.560 |
| kddcup      | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| kick        | 0.293 | 0.298 | 0.293 | 0.295 | 0.306 | 0.297 | 0.308 | 0.306 | 0.303 | 0.302 |
| letter      | 0.116 | 0.111 | 0.113 | 0.138 | 0.473 | 0.276 | 0.358 | 0.274 | 0.691 | 0.807 |
| sitimage    | 0.227 | 0.234 | 0.228 | 0.265 | 0.333 | 0.261 | 0.313 | 0.265 | 0.372 | 0.365 |
| sensorless  | 0.004 | 0.004 | 0.004 | 0.005 | 0.033 | 0.025 | 0.035 | 0.027 | 0.074 | 0.075 |
| spambase    | 0.115 | 0.121 | 0.136 | 0.137 | 0.203 | 0.216 | 0.178 | 0.160 | 0.215 | 0.210 |

algorithm. We observe that WW is generally among the fastest algorithms, for performances comparable to ones of all baselines (bold is for best EGB training time or performance, underline for best RF or WW training time or performance).

APPENDIX D

SUPPLEMENTARY DETAILS ABOUT HYPERPARAMETER TUNING

In this Section, we provide extra information about hyperparameters optimization. For XGBoost, LightGBM and CatBoost, with all other hyperparameters fixed, we use early stopping by monitoring the log loss on the validation set, the maximum number of boosting iterations being set at 5,000. The best number of iterations is used together with other best hyperparameters to refit over the whole training set before evaluation on the test set. For scikit-learn’s Random Forest and WildWood, we report results both for 10 and 100 trees, note that the default choice is 10 for WildWood (since subtrees aggregation allows to use fewer trees than RF) while default is 100 in scikit-learn. We list the hyperparameters search space of each algorithm below.

A. XGBoost

- **eta**: log-uniform distribution \([e^{-7}, 1]\);
- **max_depth**: discrete uniform distribution \([2, 10]\);
- **subsample**: uniform \([0.5, 1]\);
- **colsample_bytree**: uniform \([0.5, 1]\);
- **colsample_bylevel**: uniform \([0.5, 1]\);
- **min_child_weight**: log-uniform distribution \([e^{-16}, e^5}\);
TABLE VI
MAIN CHARACTERISTICS OF THE DATASETS USED IN EXPERIMENTS, INCLUDING NUMBER OF SAMPLES, NUMBER OF FEATURES, NUMBER OF CATEGORICAL FEATURES, NUMBER OF CLASSES AND THE GINI INDEX OF THE CLASS DISTRIBUTION ON THE WHOLE DATASETS (RESCALED BETWEEN 0 AND 1), IN ORDER TO QUANTIFY CLASS UNBALANCING

| Dataset   | # Samples | # Features | # Categorical features | # Classes | Gini |
|-----------|-----------|------------|------------------------|-----------|------|
| adult     | 48,842    | 14         | 8                      | 2         | 0.36 |
| bank      | 45,211    | 16         | 10                     | 2         | 0.21 |
| banknote  | 1,372     | 4          | 0                      | 2         | 0.49 |
| breastcancer | 569     | 30         | 0                      | 2         | 0.47 |
| car       | 1,728     | 6          | 6                      | 4         | 0.46 |
| covtype   | 581,012   | 54         | 0                      | 7         | 0.62 |
| default_cb | 30,000   | 23         | 3                      | 2         | 0.34 |
| higgs     | 11,000,000| 28         | 0                      | 2         | 0.50 |
| HCV       | 615       | 13         | 2                      | 2         | 0.19 |
| internet  | 10,108    | 70         | 70                     | 46        | 0.88 |
| ionosphere| 351       | 34         | 0                      | 2         | 0.46 |
| kddcup99  | 4,898,431 | 41         | 7                      | 23        | 0.58 |
| kick      | 72,983    | 32         | 18                     | 2         | 0.22 |
| letter    | 20,000    | 16         | 0                      | 26        | 0.96 |
| satimage  | 5,104     | 36         | 0                      | 6         | 0.81 |
| sensorless| 58,509    | 48         | 0                      | 11        | 0.91 |
| spambase  | 4,601     | 57         | 0                      | 2         | 0.48 |

TABLE VII
THE URLs OF ALL THE DATASETS USED IN THE PAPER, GIVING DIRECT DOWNLOAD LINKS AND SUPPLEMENTARY DETAILS

| Dataset | URL |
|---------|-----|
| adult   | https://archive.ics.uci.edu/ml/datasets/Adult |
| bank    | https://archive.ics.uci.edu/ml/datasets/bank+marketing |
| banknote| https://archive.ics.uci.edu/ml/datasets/banknote+authentication |
| breastcancer | https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic) |
| car     | https://archive.ics.uci.edu/ml/datasets/car+evaluation |
| covtype | https://archive.ics.uci.edu/ml/datasets/covertyp |
| default_cb | https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients |
| HCV     | https://archive.ics.uci.edu/ml/datasets/HCV+data |
| higgs   | https://archive.ics.uci.edu/ml/datasets/HIGGS |
| internet| https://kdd.ics.uci.edu/databases/internet_usage/internet_usage.html |
| ionosphere| https://archive.ics.uci.edu/ml/datasets/ionosphere |
| kddcup99| https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html |
| kick    | https://www.openml.org/d/41162 |
| letter  | https://archive.ics.uci.edu/ml/datasets/letter-recognition |
| satimage| https://archive.ics.uci.edu/ml/datasets/Statlog+(LandSat+Satellite) |
| sensorless | https://archive.ics.uci.edu/ml/datasets/sonar+drive+diagnosis |
| spambase| https://archive.ics.uci.edu/ml/datasets/spambase |

- alpha: 0 with probability 0.5, and log-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;
- lambda: 0 with probability 0.5, and log-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;
- gamma: 0 with probability 0.5, and log-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;
- $\beta$: $\log$-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;

B. LightGBM
- learning_rate: log-uniform distribution $[e^{-7}, 1]$;
- num_leaves: discrete log-uniform distribution $[1, e^7]$;
- feature_fraction: uniform $[0.5, 1]$;
- bagging_fraction: uniform $[0.5, 1]$;
- min_data_in_leaf: discrete log-uniform distribution $[1, e^6]$;
- min_sum_hessian_in_leaf: log-uniform distribution $[e^{-16}, e^5]$;
- lambda_l1: 0 with probability 0.5, and log-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;
- lambda_l2: 0 with probability 0.5, and log-uniform distribution $[e^{-16}, e^2]$ with probability 0.5;

C. CatBoost
- learning_rate: log-uniform distribution $[e^{-7}, 1]$;
- random_strength: discrete uniform distribution over $\{1, 20\}$;
- one_hot_max_size: discrete uniform distribution over $\{0, 25\}$;
- 12_leaf_reg: log-uniform distribution $[1, 10]$;
- bagging_temperature: uniform $[0, 1]$.
Table VIII
Areas under the ROC Curves (AUC) obtained on test samples with WildWood (using 100 trees in the forest) on the adult, bank and car datasets with combinations of several hyperparameters. We observe that WildWood’s performance does not vary significantly with respect to these hyperparameters.

| Dataset | Hyperparameter | Hyperparameter | Hyperparameter | AUC |
|---------|----------------|----------------|----------------|-----|
| Adult   | n_min_leaf     | alpha          | beta           | 0.913 |
| Bank    | n_min_leaf     | alpha          | beta           | 0.926 |
| Car     | n_min_leaf     | alpha          | beta           | 0.929 |

D. HistGradientBoosting
- learning_rate: log-uniform distribution \([e^{-4}, 1]\);
- max_leaf_nodes: discrete log-uniform distribution \([1, e^7]\);
- min_samples_leaf: discrete log-uniform distribution \([1, e^6]\);
- l2_regularization: 0 with probability 0.5, and log-uniform distribution \([e^{-16}, e^2]\) with probability 0.5;

E. RandomForest
- max_features: uniform among None, sqrt, log2, 0.25, 0.5 and 0.75;
- max_depth: uniform among None, sqrt and log2, the latter two are meant in terms of the train sample size;
- min_samples_leaf: uniform over \(\{1, 5, 10\}\) and we set min_samples_split = 2 × min_samples_leaf;

F. WildWood
- multiclass: multinomial with probability 0.5, and ovr with probability 0.5;
- min_samples_leaf: uniform over \(\{1, 5, 10\}\) and we set min_samples_split = 2 × min_samples_leaf;
- step: log-uniform distribution \([e^{-3}, e^0]\);
- dirichlet: log-uniform distribution \([e^{-7}, e^2]\);
- cat_split_strategy: binary with probability 0.5, and all with probability 0.5;
- max_features: uniform among None, sqrt, log2, 0.25, 0.5 and 0.75;
- max_depth: uniform among None, sqrt and log2, the latter two are meant in terms of the train sample size;

Appendix E
Datasets
The main characteristics of the datasets used in the paper are summarized in Table VI. We provide in Table VII the URL of all the datasets used, most of them are from the UCI machine learning repository [72].

Note that the HCV data set labels were binarized by considering the class “Blood Donor” against all the others.

Appendix F
Sensitivity of Hyperparameters of WildWood
In Table VIII we illustrate the effects of hyperparameters on WW’s performance on a few datasets, measured by the test AUC. We can observe in this table that it is only weakly affected by different combinations of hyperparameters.

Appendix G
Supplementary Details About Assets Used (Versions and Licenses)
The versions and licenses of the libraries used in our experiments are:
- catboost (0.25.1), Apache License 2.0
- hyperopt (0.2.5), license: https://github.com/hyperopt/hyperopt/blob/master/LICENSE.txt
- joblib (0.17), BSD-3-Clause License
lightgbm (3.2.1), MIT License
matplotlib (3.3.1), license: https://github.com/matplotlib/matplotlib/blob/master/LICENSE/LICENSE
numba (0.52), BSD 2-Clause “Simplified” License
numpy (1.19.2), BSD-3-Clause License
pandas (1.2.4), BSD-3-Clause License
python (3.7.9), Python Software Foundation Licence version 2
scikit-learn (0.24.2), BSD 3-Clause License
scipy (1.5.4), BSD 3-Clause License
seaborn (0.11), BSD-3-Clause License
xgboost (1.4.1), Apache License 2.0

All the datasets used are publicly accessible and have no copyright restrictions.

REFERENCES

[1] L. Breiman, “Random forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
[2] G. Louppe, “Understanding random forests: From theory to practice,” Ph.D. thesis, Dept. Elect. Eng. Comput. Sci., Fac. Appl. Sci., Univ. Liège, Liège, Belgium, 2014.
[3] G. Blau and E. Scornet, “A random forest guided tour,” TEST, vol. 25, no. 2, pp. 197–227, Jun. 2016.
[4] O. Catoni, Statistical Learning Theory and Stochastic Optimization: École d’Été de Probabilités de Saint-Flour 31 (Lecture Notes in Mathematics), vol. 1851. Berlin, Germany: Springer-Verlag, 2004.
[5] Y. Zhou, Z. Zhou, and G. Hooker, “Approximation trees: Statistical stability in model distillation,” 2018, arXiv:1808.07573.
[6] U. Johansson, C. Sönströd, and T. Löfström, “One tree to explain them all,” in Proc. IEEE Congr. Evol. Comput. (CEC), Jun. 2011, pp. 1444–1451.
[7] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, “The context-tree weighting method: Basic properties,” IEEE Trans. Inf. Theory, vol. 41, no. 3, pp. 653–664, May 1995.
[8] F. M. J. Willems, “The context-tree weighting method: Extensions,” IEEE Trans. Inf. Theory, vol. 44, no. 2, pp. 792–798, Mar. 1998.
[9] D. P. Helmbold and R. E. Schapire, “Predicting nearly as well as the best pruning of a decision tree,” Mach. Learn., vol. 27, no. 1, pp. 51–68, 1997.
[10] J. Mortarida, S. Gaifas, and E. Scornet, “AMF: Aggregated Mondrian forests for online learning,” J. Roy. Stat. Soc. B. Stat. Methodol., vol. 83, no. 3, pp. 505–533, Jul. 2021.
[11] H. A. Chipman, E. I. George, and R. E. McCulloch, “Bayesian CART model search,” J. Amer. Stat. Assoc., vol. 93, no. 443, pp. 935–948, Sep. 1998.
[12] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith, “A Bayesian CART algorithm,” Biometrika, vol. 85, no. 2, pp. 363–377, Jun. 1998.
[13] H. A. Chipman, I. E. George, and R. E. McCulloch, “BART: Bayesian additive regression trees,” Ann. Appl. Statist., vol. 4, no. 1, pp. 266–298, Mar. 2010.
[14] M. A. Taddy, R. B. Gramacy, and N. G. Polson, “Dynamic trees for learning and design,” J. Amer. Stat. Assoc., vol. 106, no. 493, pp. 109–123, Mar. 2011.
[15] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, Aug. 2016, pp. 785–794.
[16] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds, Red Hook, NY, USA: Curran Associates, 2017.
[17] L. Breiman, F. Heidelberger, and P. Shih, “Efficient online random forests,” in Proc. Adv. Neural Inf. Process. Syst., vol. 63, no. 1, pp. 3–42, Apr. 2006.
[18] T. Chen and G. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, Aug. 2016, pp. 785–794.
[19] C. G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” J. Mach. Learn. Res., vol. 18, no. 1, pp. 876–883, Jun. 2017.
[20] L. Breiman, “Random forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
[21] G. Louppe, “Understanding random forests: From theory to practice,” Ph.D. thesis, Dept. Elect. Eng. Comput. Sci., Fac. Appl. Sci., Univ. Liège, Liège, Belgium, 2014.
[22] D. P. Helmbold and R. E. Schapire, “Predicting nearly as well as the best pruning of a decision tree,” Mach. Learn., vol. 27, no. 1, pp. 51–68, 1997.
[23] J. Mortarida, S. Gaifas, and E. Scornet, “AMF: Aggregated Mondrian forests for online learning,” J. Roy. Stat. Soc. B. Stat. Methodol., vol. 83, no. 3, pp. 505–533, Jul. 2021.
[24] H. A. Chipman, E. I. George, and R. E. McCulloch, “Bayesian CART model search,” J. Amer. Stat. Assoc., vol. 93, no. 443, pp. 935–948, Sep. 1998.
[25] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith, “A Bayesian CART algorithm,” Biometrika, vol. 85, no. 2, pp. 363–377, Jun. 1998.
[26] H. A. Chipman, I. E. George, and R. E. McCulloch, “BART: Bayesian additive regression trees,” Ann. Appl. Statist., vol. 4, no. 1, pp. 266–298, Mar. 2010.
[27] M. A. Taddy, R. B. Gramacy, and N. G. Polson, “Dynamic trees for learning and design,” J. Amer. Stat. Assoc., vol. 106, no. 493, pp. 109–123, Mar. 2011.
[28] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” Mach. Learn., vol. 63, no. 1, pp. 3–42, Apr. 2006.
[29] T. Chen and G. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, Aug. 2016, pp. 785–794.
[30] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds, Red Hook, NY, USA: Curran Associates, 2017.
[31] L. Breiman, F. Heidelberger, and P. Shih, “Efficient online random forests,” in Proc. Adv. Neural Inf. Process. Syst., vol. 63, no. 1, pp. 3–42, Apr. 2006.
[32] T. Chen and G. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, Aug. 2016, pp. 785–794.
[33] C. G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” J. Mach. Learn. Res., vol. 18, no. 1, pp. 876–883, Jun. 2017.
[34] L. Breiman, “Random forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
[35] G. Louppe, “Understanding random forests: From theory to practice,” Ph.D. thesis, Dept. Elect. Eng. Comput. Sci., Fac. Appl. Sci., Univ. Liège, Liège, Belgium, 2014.
[36] D. P. Helmbold and R. E. Schapire, “Predicting nearly as well as the best pruning of a decision tree,” Mach. Learn., vol. 27, no. 1, pp. 51–68, 1997.
[37] J. Mortarida, S. Gaifas, and E. Scornet, “AMF: Aggregated Mondrian forests for online learning,” J. Roy. Stat. Soc. B. Stat. Methodol., vol. 83, no. 3, pp. 505–533, Jul. 2021.
[38] H. A. Chipman, E. I. George, and R. E. McCulloch, “Bayesian CART model search,” J. Amer. Stat. Assoc., vol. 93, no. 443, pp. 935–948, Sep. 1998.
[39] D. G. T. Denison, B. K. Mallick, and A. F. M. Smith, “A Bayesian CART algorithm,” Biometrika, vol. 85, no. 2, pp. 363–377, Jun. 1998.
[40] H. A. Chipman, I. E. George, and R. E. McCulloch, “BART: Bayesian additive regression trees,” Ann. Appl. Statist., vol. 4, no. 1, pp. 266–298, Mar. 2010.
