Aggregated Mondrian forests for online learning

Abstract. Random Forests (RF) is one of the algorithms of choice in many supervised learning applications, be it classification or regression. The appeal of such methods comes from a combination of several characteristics: a remarkable accuracy in a variety of tasks, a small number of parameters to tune, robustness with respect to features scaling, a reasonable computational cost for training and prediction, and their suitability in high-dimensional settings. The most commonly used RF variants however are "offline" algorithms, which require the availability of the whole dataset at once. In this chapter, we introduce AMF, an online random forest algorithm based on Mondrian Forests. Using a variant of the Context Tree Weighting algorithm, we show that it is possible to efficiently perform an exact aggregation over all prunings of the trees; in particular, this enables to obtain a truly online parameter-free algorithm which is competitive with the optimal pruning of the Mondrian tree, and thus adaptive to the unknown regularity of the regression function. Numerical experiments show that AMF is competitive with respect to several strong baselines on a large number of datasets for multi-class classification.

Contents

3.1 Introduction **139**
3.2 Forests of aggregated Mondrian trees **142**
3.3 Theoretical guarantees **150**
3.4 Practical implementation of AMF **155**
3.5 Numerical experiments **159**
3.6 Conclusion **163**
3.7 Proofs **163**

3.1 Introduction

Introduced by Breiman (2001a), Random Forests (RF) is one of the algorithms of choice in many supervised learning applications. The appeal of these methods comes from their remarkable accuracy in a variety of tasks, the small number (or even the absence) of parameters to tune, their reasonable computational cost at training and prediction time, and their suitability in high-dimensional settings.

Most commonly used RF algorithms, such as the original random forest procedure (Breiman, 2001a), extra-trees (Geurts et al., 2006), or conditional inference forest (Hothorn et al., 2010) are batch algorithms, that require the whole dataset to be available at once. Several online random forests variants have been proposed to overcome this issue and handle data that come sequentially. Utgoff (1989) was the first to extend Quinlan's ID3 batch decision tree algorithm (see Quinlan, 1986) to an online setting. Later on, Domingos and Hulten (2000) introduce Hoeffding Trees that can be easily updated: since observations are available sequentially, a cell is split when (*i*) enough observations have fallen into this cell, (*ii*) the best split in the cell is statistically relevant (a generic Hoeffding inequality being used to assess the quality of the best split).

Since random forests are known to exhibit better empirical performances than individual decision trees, online random forests have been proposed (see, e.g., Saffari et al., 2009; Denil et al., 2013). These procedures aggregate several trees by computing the mean of the tree predictions (regression setting) or the majority vote among trees (classification setting). The tree construction differs from one forest to another but share similarities with Hoeffding trees: a cell is to be split if (*i*) and (*ii*) (defined above) are verified.

One forest of particular interest for this work is the Mondrian Forest (Lakshminarayanan et al., 2014) based on the Mondrian process (Roy and Teh, 2009). Their construction differs from the construction described above since each new observation modifies the tree structure: instead of waiting for enough observations to fall into a cell in order to split it, the properties of the Mondrian process allow to update the Mondrian tree partition each time a sample is collected. Once a Mondrian tree is built, its prediction function uses a hierarchical prior on all subtrees and the average of predictions on all subtrees is computed with respect to this hierarchical prior using an approximation algorithm.

The algorithm we propose, called AMF, and illustrated in Figure 3.1 below on a toy binary classification dataset, differs from Mondrian Forest by the smoothing procedure used on each tree. While the hierarchical Bayesian smoothing proposed in Lakshminarayanan et al. (2014) requires approximations, the prior we choose allows for exact computation of the posterior distribution. The choice of this posterior is inspired by Context Tree Weighting (see, e.g., Willems et al., 1995; Willems, 1998; Helmbold and Schapire, 1997; Catoni, 2004), commonly used in lossless compression to aggregate all subtrees of a prespecified tree, which is both computationally efficient and theoretically sound. Since we are able to compute exactly

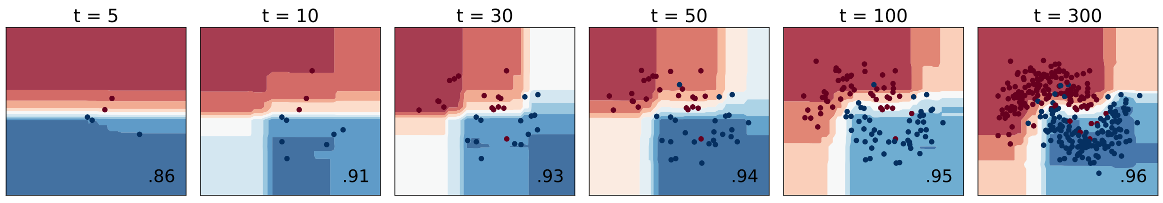

Figure 3.1: Evolution of the decision function of AMF along the online learning steps. We observe the online property of this algorithm, which produces a smooth decision function at each iteration, and leads to a correct AUC on a test set even in the early stages.

the posterior distribution, our approach is drastically different from Bayesian trees (see, for instance, Chipman et al., 1998; Denison et al., 1998; Taddy et al., 2011), and from BART

(Chipman et al., 2010) which implement MCMC methods to approximate posterior distributions on trees. The Context Tree Weighting algorithm has been applied to regression trees by Blanchard (1999) in the case of a fixed-design tree, in which splits are prespecified. This requires to split the dataset into two parts (using the first part to select the best splits and the second to compute the posterior distribution) and to have access to the whole dataset, since the tree structure needs to be fixed in advance.

As noted by Rockova and van der Pas (2017), the theoretical study of Bayesian methods on trees (Chipman et al., 1998; Denison et al., 1998) or sum of trees (Chipman et al., 2010) is less developed. Rockova and van der Pas (2017) analyzes some variant of Bayesian regression trees and sum of trees; they obtain near minimax optimal posterior concentration rates. Likewise, Linero and Yang (2018) analyze Bayesian sums of soft decision trees models, and establish minimax rates of posterior concentration for the resulting SBART procedure. While these frameworks differ from ours (herein results are posterior concentration rates as opposed to regret and excess risk bounds, and the design is fixed), their approach differs from ours primarily in the chosen trade-off between computational complexity and adaptivity of the method: these procedures involve approximate posterior sampling over large functional spaces through MCMC methods, and it is unclear whether the considered priors allow for reasonably efficient posterior computations. In particular, the prior used in Rockova and van der Pas (2017) is taken over all subsets of variables, which is exponentially large in the number of features.

The literature focusing on the original RF algorithm or its related variants is more extensive, even if the data-dependent nature of the algorithm and its numerous components (sampling procedure, split selection, aggregation) make the theoretical analysis difficult. The consistency of stylized RF algorithms was first established by Biau et al. (2008), and later obtained for more sophisticated variants in Denil et al. (2013); Scornet et al. (2015). Note that consistency results do not provide rates of convergence, and hence only offer limited guidance on how to properly tune the parameters of the algorithm. Starting with Biau (2012); Genuer (2012), some recent work has thus sought to quantify the speed of convergence of some stylized variants of RF. Minimax optimal nonparametric rates were first obtained by Arlot and Genuer (2014) in dimension 1 for the Purely Uniformly Random Forests (PURF) algorithm, in conjunction with suboptimal rates in arbitrary dimension (the number of features exceeds 1).

Several recent works (Wager and Walther, 2015; Duroux and Scornet, 2018) also established rates of convergence for variants of RF that essentially amount to some form of Median Forests, where each node contains at least a fixed fraction of observations of its parent. While valid in arbitrary dimension, the established rates are suboptimal. More recently, adaptive minimax optimal rates were obtained by Mourtada et al. (2018) (Chapter 2) in arbitrary dimension for the batch Mondrian Forests algorithm. Our proposed online algorithm, AMF, also achieves minimax rates in an adaptive fashion, namely without knowing the smoothness of the regression function.

In this chapter, we introduce AMF, a random forest algorithm which is fully online and computationally exact: unlike Bayesian trees and sum-of-trees procedures relying on approximate posterior sampling, we are able to compute exactly the prediction function of AMF in a very efficient way. Section 3.2 introduces the setting considered and general notations, and provides a precise construction of the AMF algorithm. A theoretical analysis of AMF is given in Section 3.3, where we establish regret bounds for AMF together with a minimax adaptive upper bound. Section 3.4 introduces a modification of AMF which is used in all the numerical experiments of the chapter, together with a guarantee and a discussion on its computational complexity. Numerical experiments are provided in Section 3.5, on a large number of datasets, that include a comparison of AMF with several strong baselines. Our conclusions are provided in Section 3.6. The proofs of all the results are gathered in Section 3.7.

3.2 Forests of aggregated Mondrian trees

We define in Section 3.2.1 the setting and notations that will be used throughout the chapter, together with the definition of the Mondrian process, introduced by Roy and Teh (2009), which is a key element of our algorithm. In Section 3.2.2, we explicitly describe the prediction function that we want to compute, and prove in Proposition 3.1 that the AMF algorithm described in Section 3.2.3 computes it exactly.

3.2.1 The setting, trees, forests and the Mondrian process

We are interested in an *online supervised learning* problem in which we assume that the dataset is not fixed in advance. In this scenario, we are given an i.i.d. sequence (x_1, y_1), $(x_2, y_2), \ldots$ of $[0,1]^d \times \mathcal{Y}$-valued random variables that come sequentially, such that each (x_t, y_t) has the same distribution as a generic pair (x, y).

Our aim is to design an *online* algorithm that can be updated "on the fly" given new sample points, that is, at each time step $t \geqslant 1$, a *randomized prediction function*

$$\widehat{f}_t(\cdot, \mathbf{\Pi}_t, \mathscr{D}_t) : [0,1]^d \to \widehat{\mathcal{Y}},$$

where $\mathscr{D}_t = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ is the dataset available at time t, where $\mathbf{\Pi}_t$ is a random variable that accounts for the randomization procedure and $\widehat{\mathcal{Y}}$ is a prediction space, see Examples 3.1 and 3.2 below for example. In the rest of the chapter, we omit the explicit dependence in $\mathscr{D}_t$.

We consider prediction rules $(\widehat{f}_t)_{t \geqslant 1}$ that are *random forests*, defined as the averaging of a set of $M \geqslant 1$ randomized decision trees. We let $\widehat{f}_t(x, \Pi_t^{(1)}), \ldots, \widehat{f}_t(x, \Pi_t^{(M)})$ be randomized tree predictors at a point $x \in [0,1]^d$ at time t, associated to the same randomized mechanism, where the $(\Pi_t^{(m)})_{t \geqslant 1}$ for $m = 1, \ldots, M$ are i.i.d. and correspond to a random tree partition, which is described below. Setting $\mathbf{\Pi}_t^{(M)} = (\Pi_t^{(1)}, \ldots, \Pi_t^{(M)})$, the *random forest estimate* $\widehat{f}_t^{(M)}(x, \mathbf{\Pi}_t^{(M)})$ is then defined by

$$\widehat{f}_t^{(M)}(x, \mathbf{\Pi}_t^{(M)}) = \frac{1}{M} \sum_{m=1}^{M} \widehat{f}_t(x, \Pi_t^{(m)}), \tag{3.1}$$

namely taking the average over all tree predictions $\widehat{f}_t(x, \Pi_t^{(m)})$. The online training of each tree can be done in parallel, since they are fully independent of each other and each of them follow the exact same randomized construction. Therefore, we describe only the construction of single tree (and its associated random partition and prediction function) and omit from now on the dependence on $m = 1, \ldots, M$.

The random tree partitions are given by $\Pi_t = (\mathcal{T}_t, \Sigma_t)$, where $\mathcal{T}_t$ is a binary tree and Σ_t contains information about each node in $\mathcal{T}_t$, such as splits, as explained below. Let us now introduce notations and definitions of these objects, for simplicity we first assume that t is fixed, and remove the dependence on t for a little while.

Definition 3.1 (Tree partition)**.** Let $C \subseteq [0,1]^d$ be a hyper-rectangular box of the form $\prod_{j=1}^d [a_j, b_j]$, $-\infty \leqslant a_j \leqslant b_j \leqslant +\infty$ (the interval being open at an infinite extremity). A *tree partition* (or *kd tree*, *guillotine partition*) of C is a pair $(\mathcal{T}, \Sigma)$, where

- $\mathcal{T}$ is a finite ordered binary tree, which is represented as a finite subset of the set $\{0,1\}^* = \bigcup_{n \geqslant 0} \{0,1\}^n$ of all finite words on the alphabet $\{0,1\}$. The set $\{0,1\}^*$ is endowed with a tree structure (and called the complete binary tree): the empty word ϵ is the root, and for any $\mathbf{v} \in \{0,1\}^*$, the left (resp. right) child of $\mathbf{v}$ is $\mathbf{v}0$ (resp. $\mathbf{v}1$), obtained by adding a 0 (resp. 1) at the end of $\mathbf{v}$. We denote by $\mathcal{N}^\circ(\mathcal{T}) = \{\mathbf{v} \in \mathcal{T} : \mathbf{v}0, \mathbf{v}1 \in \mathcal{T}\}$ the set of its *interior nodes* and by $\mathcal{L}(\mathcal{T}) = \{\mathbf{v} \in \mathcal{T} : \mathbf{v}0, \mathbf{v}1 \notin \mathcal{T}\}$ the set of its leaves, which are disjoint by definition.

- $\Sigma = (\sigma_{\mathbf{v}})_{\mathbf{v} \in \mathcal{N}^\circ(\mathcal{T})}$ is a family of *splits* at the interior nodes of $\mathcal{T}$, where each split $\sigma_{\mathbf{v}} = (j_{\mathbf{v}}, s_{\mathbf{v}})$ is characterized by its split dimension $j_{\mathbf{v}} \in \{1, \ldots, d\}$ and its threshold $s_{\mathbf{v}} \in [0,1]$. In Section 3.2.3, we will actually store in $\sigma_{\mathbf{v}} \in \Sigma$ more information about nodes $\mathbf{v} \in \mathcal{T}$.

One can associate to $(\mathcal{T}, \Sigma)$ a partition $(C_{\mathbf{v}})_{\mathbf{v} \in \mathcal{L}(\mathcal{T})}$ of $[0,1]^d$ as follows. For each node $\mathbf{v} \in \mathcal{T}$, its *cell* $C_{\mathbf{v}}$ is a hyper-rectangular region $C_{\mathbf{v}} \subseteq [0,1]^d$ defined recursively: the cell associated to the root ϵ of $\mathcal{T}$ is $[0,1]^d$, and, for each $\mathbf{v} \in \mathcal{N}^\circ(\mathcal{T})$, we define

$$C_{\mathbf{v}0} := \{x \in C_{\mathbf{v}} : x_{j_{\mathbf{v}}} \leqslant s_{j_{\mathbf{v}}}\} \quad \text{and} \quad C_{\mathbf{v}1} := C_{\mathbf{v}} \setminus C_{\mathbf{v}0}.$$

Then, the leaf cells $(C_{\mathbf{v}})_{\mathbf{v} \in \mathcal{L}(\mathcal{T})}$ form a partition of $[0,1]^d$ by construction.

Mondrian partitions are a specific family of random tree partitions whose construction is described below. An infinite Mondrian partition Π of $[0,1]^d$ can be sampled from the infinite Mondrian process, denoted MP from now on, using the procedure `SampleMondrian`$([0,1]^d, \tau = 0)$ described below. If $C = \prod_{j=1}^d C^j$ with intervals $C^j = [a_j, b_j]$, we denote $|C^j| = b_j - a_j$ and $|C| = \sum_{j=1}^d |C^j|$. We denote by $\mathsf{Exp}(\lambda)$ the exponential distribution with intensity $\lambda > 0$ and by $\mathcal{U}([a,b])$ the uniform distribution on a finite interval $[a,b]$.

Algorithm 2 `SampleMondrian`$(C_{\mathbf{v}}, \tau_{\mathbf{v}})$: sample a Mondrian starting from a cell $C_{\mathbf{v}}$ and time $\tau_{\mathbf{v}}$

```
1: Inputs: The cell $C_{\mathbf{v}} = \prod_{1 \leqslant j \leqslant d} C_{\mathbf{v}}^j$ and creation time $\tau_{\mathbf{v}}$ of a node $\mathbf{v}$
2: Sample a random variable $E \sim \mathsf{Exp}(|C_{\mathbf{v}}|)$ and put $\tau_{\mathbf{v}0} = \tau_{\mathbf{v}1} = \tau_{\mathbf{v}} + E$
3: Sample a split coordinate $j_{\mathbf{v}} \in \{1, \ldots, d\}$ with $\mathbb{P}(j_{\mathbf{v}} = j) = |C_{\mathbf{v}}^j| / |C_{\mathbf{v}}|$
4: Sample a split threshold $s_{\mathbf{v}}$ conditionally on $j_{\mathbf{v}}$ as $s_{\mathbf{v}} | j_{\mathbf{v}} \sim \mathcal{U}(C_{\mathbf{v}}^{j_{\mathbf{v}}})$
5: Following Definition 3.1, the split $(j_{\mathbf{v}}, s_{\mathbf{v}})$ defines children cells $C_{\mathbf{v}0}$ and $C_{\mathbf{v}1}$
6: return SampleMondrian$(C_{\mathbf{v}0}, \tau_{\mathbf{v}0})$ ∪ SampleMondrian$(C_{\mathbf{v}1}, \tau_{\mathbf{v}1})$
```

The call to `SampleMondrian`$([0,1]^d, \tau = 0)$ corresponds to a call starting at the root node $\mathbf{v} = \epsilon$, since $C_\epsilon = [0,1]^d$ and the birth time of ϵ is $\tau_\epsilon = 0$. This random partition is built by iteratively splitting cells at some random time, which depends on the linear dimension $C_{\mathbf{v}}$ of the input cell $C_{\mathbf{v}}$. The split coordinate $j_{\mathbf{v}}$ is chosen at random, with a probability of sampling j which is proportional to the side length $|C_{\mathbf{v}}^j| / |C_{\mathbf{v}}|$ of the cell, and the split threshold is sampled uniformly in $C_{\mathbf{v}}^j$. The number of recursions in this procedure is infinite, the Mondrian process MP is a distribution on infinite tree partitions of $[0,1]^d$, see Roy and

Teh (2009) and Roy (2011) for a rigorous construction. The random partition described in Section 3.2.3 below, is, however, not infinite, and depends on the features vectors x_t seen until time t. The implementation of AMF used in all our experiments, described in Section 3.4 below, also considers finite partitions, through the concept of *restricted Mondrian partitions*, introduced in Lakshminarayanan et al. (2014). At this point, the *birth times* $\tau_{\mathbf{v}}$ computed in Algorithm 2 are not used. They will allow to define *time prunings* of a Mondrian partition in Section 3.3.1 below, a notion which is necessary to prove that AMF has adaptation capabilities to the optimal time pruning. Birth times $\tau_{\mathbf{v}}$ are also necessary for the definition of restricted Mondrian partitions in Section 3.4, which is an important ingredient in the actual implementation of AMF.

3.2.2 Aggregation with exponential weights and prediction functions

The prediction function of AMF is an aggregation of the predictions given by all finite subtrees of the infinite Mondrian partition MP. This aggregation step is performed in a purely online fashion, using an aggregation algorithm based on exponential weights, with a branching process prior over the subtrees, see Definition 3.3 below. This weighting scheme gives more importance to subtrees with a good predictive performance.

Let us assume that the realization of an infinite Mondrian partition $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi) \sim \mathsf{MP}$ is available at some fixed step t. We will argue in Section 3.2.3 that it suffices to store a finite partition Π_t, and show how to update it. The definition of the prediction function used in AMF require the notion of *node* and *subtree prediction*, defined below.

Definition 3.2. Given $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi) \sim \mathsf{MP}$, we define

$$\widehat{y}_{\mathbf{v},t} = h((y_s)_{1 \leqslant s \leqslant t-1 \,:\, x_s \in C_{\mathbf{v}}}) \quad \text{and} \quad L_{\mathbf{v},t} = \sum_{1 \leqslant s \leqslant t \,:\, x_s \in C_{\mathbf{v}}} \ell(\widehat{y}_{\mathbf{v},s}, y_s)$$

for each node $\mathbf{v} \in \mathcal{T}^\Pi$ (which defines a cell $C_{\mathbf{v}} \subseteq [0,1]^d$ following Definition 3.1) and each $t \geqslant 1$, where $h : \bigcup_{t \geqslant 0} \mathcal{Y}^t \to \widehat{\mathcal{Y}}$ is a prediction algorithm used in each cell, with $\widehat{\mathcal{Y}}$ its prediction space and $\ell : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbf{R}$ a generic loss function. The prediction at time t of a finite subtree $\mathcal{T} \subset \mathcal{T}^\Pi$ associated to some features vector $x \in [0,1]^d$ is defined by

$$\widehat{y}_{\mathcal{T},t}(x) = \widehat{y}_{\mathbf{v}_{\mathcal{T}}(x),t},$$

where $\mathbf{v}_{\mathcal{T}}(x)$ is the leaf of $\mathcal{T}$ that contains x. We define also the cumulative loss of $\mathcal{T}$ at time t as

$$L_t(\mathcal{T}) = \sum_{s=1}^{t} \ell(\widehat{y}_{\mathcal{T},s}(x_s), y_s) \,.$$

Before defining the prediction function of AMF, let us first make explicit the prediction function h and the loss considered in two specific cases of interest: regression and classification.

Example 3.1 (Regression)**.** In regression, we use empirical mean forecasters

$$\widehat{y}_{\mathbf{v},t+1} = \frac{1}{n_{\mathbf{v},t}} \sum_{1 \leqslant s \leqslant t \,:\, x_s \in C_{\mathbf{v}}} y_s,$$

where $n_{\mathbf{v},t} = |\{1 \leqslant s \leqslant t \,:\, x_s \in C_{\mathbf{v}}\}|$, and we simply put $\widehat{y}_{\mathbf{v},t} = 0$ if $\mathbf{v}$ is empty (namely, $C_{\mathbf{v}}$ contains no data point). The loss is the *quadratic loss* $\ell(\widehat{y}, y) = (\widehat{y} - y)^2$ for any $y \in \mathcal{Y}$ and $\widehat{y} \in \widehat{\mathcal{Y}}$ where $\widehat{\mathcal{Y}} = \mathcal{Y} = \mathbf{R}$.

Example 3.2 (Classification)**.** For multi-class classification, we have labels $y_t \in \mathcal{Y}$ where $\mathcal{Y}$ is a finite set of label modalities (such as $\mathcal{Y} = \{1, \ldots, K\}$) and predictions are in $\widehat{\mathcal{Y}} = \mathcal{P}(\mathcal{Y})$, the set of probability distributions on $\mathcal{Y}$. We use the *Krichevsky-Trofimov* (KT) forecaster (see Tjalkens et al., 1993) in each node $\mathbf{v}$, which predicts

$$\widehat{y}_{\mathbf{v},t+1}(y) = \frac{n_{\mathbf{v},t}(y) + 1/2}{t + |\mathcal{Y}|/2}, \tag{3.2}$$

for any $y \in \mathcal{Y}$, where $n_{\mathbf{v},t}(y) = |\{1 \leqslant s \leqslant t : x_s \in C_{\mathbf{v}}, y_s = y\}|$. For an empty $\mathbf{v}$ we use the uniform distribution on $\mathcal{Y}$. We consider the *logarithmic loss* (also called *cross-entropy* or *self-information* loss) $\ell(\widehat{y}, y) = -\log \widehat{y}(y)$, where $\widehat{y}(y) = \widehat{y}(\{y\}) \in [0, 1]$.

Remark 3.1. The Krichevsky-Trofimov forecaster coincides with the exponential weights algorithm under the logarithmic loss (with $\eta = 1$) on $\mathcal{P}(\mathcal{Y})$ with a prior equal to the Dirichlet distribution $\mathsf{Dir}(\frac{1}{2}, \ldots, \frac{1}{2})$, namely the *Jeffreys prior* on the multinomial model $(\mathcal{Y}, \mathcal{P}(\mathcal{Y}))$.

Definition 3.3. Let $t \geqslant 1$ and $x \in [0, 1]^d$. The prediction function $\widehat{f}_t$ of AMF at step t is given by

$$\widehat{f}_t(x) = \frac{\sum_{\mathcal{T}} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})} \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T}} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})}},$$

where the sum is over all subtrees $\mathcal{T}$ of $\mathcal{T}^{\Pi}$ and where the *prior* π on subtrees is the probability distribution defined by

$$\pi(\mathcal{T}) = 2^{-|\mathcal{T}|}, \tag{3.3}$$

where $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$ and $\eta > 0$ is a parameter called *learning rate.*

Note that π is the distribution of the branching process with branching probability 1/2 at each node of $\mathcal{T}^{\Pi}$, with exactly two children when it branches; this branching process gives finite subtrees almost surely. The learning rate η can be optimally tuned following theoretical guarantees from Section 3.3, see in particular Corollaries 3.1 and 3.2. This aggregation procedure is a *non-greedy way to prune trees*: the weights do not depend only on the quality of one single split but rather on the performance of each subsequent split.

Let us stress that computing $\widehat{f}_t$ from Definition 3.3 seems computationally infeasible in practice, since it involves a sum over all subtrees of $\mathcal{T}^{\Pi}$. Besides, it requires to keep in memory one weight $e^{-\eta L_{t-1}(\mathcal{T})}$ for all subtrees $\mathcal{T}$, which seems prohibitive as well. Indeed, the number of subtrees of the minimal tree that separates n points is exponential in the number of nodes, and hence *exponential in* n. However, the proper choice of the prior in Equation (3.3) allows us to prove that $\widehat{f}_t$ can actually be computed very efficiently, at almost no memory cost, as stated in Proposition 3.1 below, where we prove that the AMF algorithm described in Section 3.2.3 below allows to compute $\widehat{f}_t$ exactly and efficiently.

Proposition 3.1. *Let $t \geqslant 1$ and $x \in [0, 1]^d$. The value $\widehat{f}_t(x)$ from Definition 3.3 can be computed exactly via the* `AmfPredict` *procedure (see Algorithms 3 and 4 from Section 3.2.3 below).*

The proof of Proposition 3.1 is given in Section 3.7. It proves that aggregating predictions of all subtrees weighted by the prior π can be done exactly via Algorithm 4. This prior choice enables to bypass the need to maintain one weight per subtree, and leads to a "collapsed" implementation that only requires to maintain one weight per node (which is exponentially smaller). Note that this algorithm is *exact*, in the sense that it does not require any approximation scheme. Moreover, this online algorithm corresponds to its batch counterpart, in the sense that there is no loss of information coming from the online (or streaming) setting versus the batch setting (where the whole dataset is available at once).

The proof of Proposition 3.1 relies on some standard identities that enable to efficiently compute sums of products over tree structures in a recursive fashion (from Helmbold and Schapire, 1997), recalled in Lemma 3.3 from Section 3.7. Such identities are at the core of the *Context Tree Weighting* algorithm (CTW), which our online algorithm implements (albeit over an evolving tree structure, as explained in Section 3.2.3 below), and which consists of an efficient way to perform Bayesian mixtures of contextual tree models under a branching process prior. The CTW algorithm, based on a sum-product factorization, is a state-of-the art algorithm used in lossless coding and compression. We use a variant of the *Tree Expert* algorithm (Helmbold and Schapire, 1997; Cesa-Bianchi and Lugosi, 2006), which is closely linked to CTW (Willems et al., 1995; Willems, 1998; Catoni, 2004).

3.2.3 AMF: a forest of aggregated Mondrian trees

In an online setting, the number of sample points increases over time, allowing one to capture more details on the distribution of y conditionally on x. This means that the complexity of our models (in this context, the complexity of the decision trees) should increase over time. We will therefore need to consider not just an individual, fixed tree partition Π, but a sequence $(\Pi_t)_{t \geqslant 1}$, indexed by "time" t corresponding to the number of samples available. Furthermore, AMF uses the aggregated prediction function given in Definition 3.3 (independently within each tree $\Pi_t^{(1)}, \ldots, \Pi_t^{(M)}$ from the forest, see Equation (3.1)). When a new sample point (x_t, y_t) becomes available, the algorithm does two things, in the following order:

- *Partition update.* Using x_t, update the decision tree structure from $\Pi_t = (\mathcal{T}_t, \Sigma_t)$ to $\Pi_{t+1} = (\mathcal{T}_{t+1}, \Sigma_{t+1})$, *i.e.* sample new splits in order to ensure that each leaf in the tree *contains at most one point* among $\{x_1, \ldots, x_t\}$. This update uses the recursive properties of Mondrian partitions;
- *Prediction function update.* Using x_t and y_t, update the prediction functions $\widehat{y}_{\mathbf{v},t}$ and weights $w_{\mathbf{v},t}$ and $\overline{w}_{\mathbf{v},t}$ that are necessary for the computation of $\widehat{f}_t$ from Definition 3.3. These updates are *local* and are performed only along the path of nodes leading to the leaf containing x_t. This update is efficient and enables the computation of $\widehat{f}_t$ from Definition 3.3, which aggregates the decision functions of all the prunings of the tree, thanks to a variant of CTW.

Both updates can be implemented on the fly in a *purely sequential manner*. Training over a sequence $(x_1, y_1), \ldots, (x_t, y_t)$ means using each sample once for training, and both updates are *exact* and do not rely on an approximate sampling scheme. Both steps are precisely described in Algorithm 3 below and illustrated in Figure 3.2. Also, in order to ease the reading of this technical part of the chapter, we gather in Table 3.1 notations that are used in this Section.

Partition update. Before seeing the point (x_t, y_t), the algorithm maintains a partition $\Pi_t = (\mathcal{T}_t, \Sigma_t)$, which corresponds to the minimal subtree of the infinite Mondrian partition $\Pi \sim \mathsf{MP}$ that separates all distinct sample points in $\{x_1, \ldots, x_{t-1}\}$. This corresponds to the tree obtained from the infinite tree Π by removing all splits of "empty" cells (that do not

Notation or formula	**Description**
$\mathbf{v} \in \{0,1\}^*$	A node
$\mathcal{T} \subset \{0,1\}^*$	A tree
$\mathbf{v}0$ (resp. $\mathbf{v}1$)	The left (resp. right) child of $\mathbf{v}$
$\mathcal{T}_\mathbf{v}$	A subtree rooted at $\mathbf{v}$
$\mathcal{L}(\mathcal{T})$	The set of leaves of $\mathcal{T}$
$\mathcal{N}^\circ(\mathcal{T})$	The set of the interior nodes of $\mathcal{T}$
$(C_\mathbf{v})_{\mathbf{v}\in\mathcal{L}(\mathcal{T})}$	The cells of the partition defined by $\mathcal{T}$
$\widehat{y}_{\mathbf{v},t}$	Prediction of a node $\mathbf{v}$ at time t
$L_{\mathbf{v},t} = \sum_{1\leqslant s\leqslant t\,:\, X_s\in C_\mathbf{v}} \ell(\widehat{y}_{\mathbf{v},s}, y_s)$	Cumulative loss of the node $\mathbf{v}$ at time t
$w_{\mathbf{v},t} = \exp(-\eta L_{\mathbf{v},t-1})$	Weight stored in node $\mathbf{v}$ at time t
$\overline{w}_{\mathbf{v},t} = \sum_{\mathcal{T}_\mathbf{v}} 2^{-\lvert\mathcal{T}_\mathbf{v}\rvert} \prod_{\mathbf{v}'\in\mathcal{L}(\mathcal{T}_\mathbf{v})} w_{\mathbf{v}',t}$	Average weight stored in node $\mathbf{v}$ at time t

Table 3.1: Notations and definitions used in AMF

contain any point among $\{x_1,\dots,x_{t-1}\}$). As x_t becomes available, this tree is updated as follows (this corresponds to Lines 2–11 in Algorithm 3 below):

- find the leaf in Π_t that contains x_t; it contains at most one point among $\{x_1,\dots,x_{t-1}\}$;
- if the leaf contains no point $x_s \neq x_t$, then let $\Pi_{t+1} = \Pi_t$. Otherwise, let x_s be the unique point among $\{x_1,\dots,x_{t-1}\}$ (distinct from x_t) in this cell. Splits of the cell containing $\{x_s, x_t\}$ are successively sampled (following the recursive definition of the Mondrian distribution), until a split separates x_s and x_t.

Prediction function update. The algorithm maintains weights $w_{\mathbf{v},t}$ and $\overline{w}_{\mathbf{v},t}$ and predictions $\widehat{y}_{\mathbf{v},t}$ in order to compute the aggregation over the tree structure (lines 12–18 in Algorithm 3). Namely, after round $t-1$ (after seeing sample (x_{t-1}, y_{t-1})), each node $\mathbf{v}\in\mathcal{T}_t$ has the following quantities in memory:

- the weight $w_{\mathbf{v},t} = \exp(-\eta L_{\mathbf{v},t-1})$, where $L_{\mathbf{v},t} := \sum_{1\leqslant s\leqslant t\,:\,x_s\in C_\mathbf{v}} \ell(\widehat{y}_{\mathbf{v},s}, y_s)$;
- the averaged weight $\overline{w}_{\mathbf{v},t} = \sum_{\mathcal{T}_\mathbf{v}} 2^{-\lvert\mathcal{T}_\mathbf{v}\rvert} \prod_{\mathbf{v}'\in\mathcal{L}(\mathcal{T}_\mathbf{v})} w_{\mathbf{v}',t}$, where the sum ranges over all subtrees $\mathcal{T}_\mathbf{v}$ rooted at $\mathbf{v}$;
- the forecast $\widehat{y}_{\mathbf{v},t}$ in node $\mathbf{v}$ at time t.

Now, given a new sample point (x_t, y_t), the update is performed as follows: we find the leaf $\mathbf{v}_t = \mathbf{v}_{\Pi_{t+1}}(x_t)$ containing x_t in Π_{t+1} (the partition has been updated with x_t already, since the partition update is performed *before* the prediction function update). Then, we update the values of $w_{\mathbf{v},t}, \overline{w}_{\mathbf{v},t}, \widehat{y}_{\mathbf{v},t}$ for each $\mathbf{v}$ along an *upwards* recursion from $\mathbf{v}_t$ to the root, while *the values of nodes outside of the path are kept unchanged*:

- $w_{\mathbf{v},t+1} = w_{\mathbf{v},t}\exp(-\eta\ell(\widehat{y}_{\mathbf{v},t}, y_t))$;
- if $\mathbf{v} = \mathbf{v}_t$ then $\overline{w}_{\mathbf{v},t+1} = w_{\mathbf{v},t+1}$, otherwise

$$\overline{w}_{\mathbf{v},t+1} = \frac{1}{2} w_{\mathbf{v},t+1} + \frac{1}{2}\overline{w}_{\mathbf{v}0,t+1}\overline{w}_{\mathbf{v}1,t+1};$$

- $\widehat{y}_{\mathbf{v},t+1} = h((y_s)_{1 \leqslant s \leqslant t \,:\, x_s \in C_{\mathbf{v}}})$ using the prediction algorithm $h : \bigcup_{t \geqslant 0} \mathcal{Y}^t \to \widehat{\mathcal{Y}}$, see Definition 3.2. Note that the prediction algorithms given in Examples 3.1 and 3.2 can be updated online using y_t only and do not require to look back at the sequence $y_1, \ldots, y_{t-1}$.

The *partition update* and *prediction function update* correspond to the `AmfUpdate`(x, y) procedure described in Algorithm 3 below. Training AMF over a sequence $(x_1, y_1), \ldots, (x_t, y_t)$

Algorithm 3 `AmfUpdate`(x, y) : update AMF with a new sample $(x, y) \in [0, 1]^d \times \mathcal{Y}$

```
1: Input: a new sample $(x, y) \in [0,1]^d \times \mathcal{Y}$
2: Let $\mathbf{v}(x)$ be the leaf such that $x \in C_{\mathbf{v}(x)}$ and put $\mathbf{v} = \mathbf{v}(x)$
3: while $C_{\mathbf{v}}$ contains some $x' \neq x$ do
4:   Use Lines 1–5 from Algorithm 2 to split $C_{\mathbf{v}}$ and obtain children cells $C_{\mathbf{v}0}$ and $C_{\mathbf{v}1}$
5:   if $\{x, x'\} \subset C_{\mathbf{v}a}$ for some $a \in \{0, 1\}$ then
6:     Put $\mathbf{v} = \mathbf{v}a$, $(w_{\mathbf{v}a}, \overline{w}_{\mathbf{v}a}, \widehat{y}_{\mathbf{v}a}) = (w_{\mathbf{v}}, \overline{w}_{\mathbf{v}}, \widehat{y}_{\mathbf{v}})$ and $(w_{\mathbf{v}(1-a)}, \overline{w}_{\mathbf{v}(1-a)}, \widehat{y}_{\mathbf{v}(1-a)}) = (1, 1, h(\varnothing))$ ($h(\varnothing)$ is the default initial prediction described in Examples 3.1 and 3.2)
7:   else
8:     Let $a \in \{0, 1\}$ be such that $x \in C_{\mathbf{v}a}$ and $x' \in C_{\mathbf{v}(1-a)}$. Put $\mathbf{v} = \mathbf{v}a$ and $(w_{\mathbf{v}a}, \overline{w}_{\mathbf{v}a}, \widehat{y}_{\mathbf{v}a}) = (1, 1, h(\varnothing))$ and $(w_{\mathbf{v}(1-a)}, \overline{w}_{\mathbf{v}(1-a)}, \widehat{y}_{\mathbf{v}(1-a)}) = (w_{\mathbf{v}}, \overline{w}_{\mathbf{v}}, \widehat{y}_{\mathbf{v}})$
9:   end if
10: end while
11: Put $x_{\mathbf{v}} = x$ (memorize the fact that $\mathbf{v}$ contains $x$)
12: Let continueUp ← true
13: while continueUp do
14:   Set $w_{\mathbf{v}} = w_{\mathbf{v}} \exp(-\eta \ell(\widehat{y}_{\mathbf{v}}, y))$
15:   Set $\overline{w}_{\mathbf{v}} = w_{\mathbf{v}}$ if $\mathbf{v}$ is a leaf and $\overline{w}_{\mathbf{v}} = \frac{1}{2} w_{\mathbf{v}} + \frac{1}{2} \overline{w}_{\mathbf{v}0} \overline{w}_{\mathbf{v}1}$ otherwise
16:   Update $\widehat{y}_{\mathbf{v}}$ using $y$ (following Definition 3.2)
17:   If $\mathbf{v} \neq \epsilon$ let $\mathbf{v}$ = parent($\mathbf{v}$), otherwise let continueUp = false
18: end while
```

means using successive calls to `AmfUpdate`$(x_1, y_1), \ldots,$ `AmfUpdate`(x_t, y_t). `AmfUpdate`(x, y) maintains in memory the current state of the Mondrian partition $\Pi = (\mathcal{T}, \Sigma)$. The tree $\mathcal{T}$ contains the parent and children relations between all nodes $\mathbf{v} \in \mathcal{T}$, while each $\sigma_{\mathbf{v}} \in \Sigma$ can contain

$$\sigma_{\mathbf{v}} = (j_{\mathbf{v}}, s_{\mathbf{v}}, \widehat{y}_{\mathbf{v}}, w_{\mathbf{v}}, \overline{w}_{\mathbf{v}}, x_{\mathbf{v}}), \tag{3.4}$$

namely the split coordinate $j_{\mathbf{v}} \in \{1, \ldots, d\}$ and split threshold $s_{\mathbf{v}} \in [0, 1]$ (only if $\mathbf{v} \in \mathcal{N}^\circ(\mathcal{T})$), the prediction function $\widehat{y}_{\mathbf{v}} \in \widehat{\mathcal{Y}}$, aggregation weights $w_{\mathbf{v}}, \overline{w}_{\mathbf{v}} \in (0, +\infty)$ and a vector $x_{\mathbf{v}} \in [0, 1]^d$ if $\mathbf{v} \in \mathcal{L}(\mathcal{T})$. An illustration of Algorithm 3 is given in Figure 3.2 below.

Remark 3.2. The complexity of `AmfUpdate`(x, y) is twice the depth of the tree at the moment it is called, since it requires to follow a downwards path to a leaf, and to go back upwards to the root. As explained in Proposition 3.2 from Section 3.4 below, the depth of the Mondrian tree used in AMF is $\Theta(\log n)$ in expectation at step n of training, which leads to a complexity $\Theta(\log n)$ both for Algorithms 3 and 4, where $\Theta(1)$ corresponds to the update complexity of a single node, while the original MF algorithm uses an update with complexity that is linear in the number of leaves in the tree (which is typically exponentially larger).

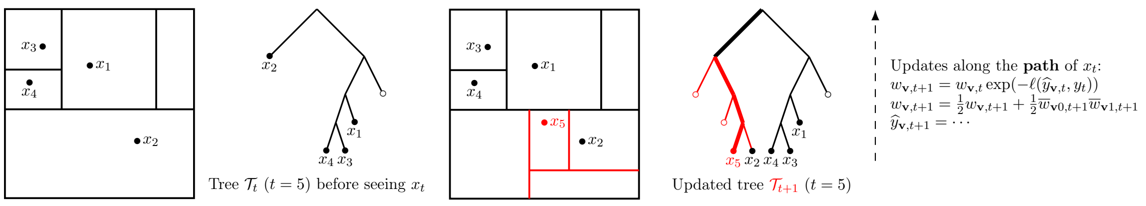
x_3
x_1
x_4
x_2
x_1
x_4 x_3
Tree $\mathcal{T}_t$ ($t = 5$) before seeing x_t
x_3
x_1
x_4
x_5
x_2
x_1
x_5 x_2 x_4 x_3
Updated tree $\mathcal{T}_{t+1}$ ($t = 5$)
Updates along the **path** of x_t:
$w_{\mathbf{v},t+1} = w_{\mathbf{v},t} \exp(-\ell(\widehat{y}_{\mathbf{v},t}, y_t))$
$w_{\mathbf{v},t+1} = \frac{1}{2} w_{\mathbf{v},t+1} + \frac{1}{2} \overline{w}_{\mathbf{v}0,t+1} \overline{w}_{\mathbf{v}1,t+1}$
$\widehat{y}_{\mathbf{v},t+1} = \cdots$

Figure 3.2: Illustration of the `AmfUpdate`(x_t, y_t) procedure from Algorithm 3: update of the partition, weights and node predictions as a new data point (x_t, y_t) for $t = 5$ becomes available. *Left*: tree partition Π_t before seeing (x_t, y_t). *Right*: update of the partition (in red) and new splits to separate x_5 from x_2. Empty circles ($\circ$) denote empty leaves, while leaves containing a point are indicated by a filled circle ($\bullet$). The path of x_t in the tree is indicated in bold. The updates of weights and predictions along the path are indicated, and are computed in an upwards recursion.

Prediction. At any point in time, one can ask AMF to perform prediction for an arbitrary features vector $x \in [0, 1]^d$. Let us assume that AMF did already t training steps on the M trees it contains and let us recall that the prediction produced by AMF is the average of their predictions, see Equation (3.1), where the prediction $\widehat{f}_t(x, \Pi_t^{(m)})$ of each decision tree $m = 1, \ldots, M$ is computed in parallel following Definition 3.3.

The prediction of a decision tree is performed through a call to procedure `AmfPredict`(x) described in Algorithm 4 below. First, we perform a temporary partition update of Π using x, following Lines 2–10 of Algorithm 3, so that we find or create a new leaf node $\mathbf{v}(x)$ such that $x \in C_{\mathbf{v}(x)}$. Let us stress that this update of Π using x is discarded once the prediction for x is produced, so that the decision function of AMF does not change after producing predictions. The prediction is then computed recursively, along an upwards recursion going from $\mathbf{v}(x)$ to the root ϵ, in the following way:

- if $\mathbf{v} = \mathbf{v}(x)$ we set $\widetilde{y}_{\mathbf{v}} = \widehat{y}_{\mathbf{v}}$;
- if $\mathbf{v} \neq \mathbf{v}(x)$ (it is an interior node such that $x \in C_{\mathbf{v}}$), then assuming that $\mathbf{v}a$ ($a \in \{0, 1\}$) is the child of $\mathbf{v}$ such that $x \in C_{\mathbf{v}a}$, we set
$$\widetilde{y}_{\mathbf{v}} = \frac{1}{2} \frac{w_{\mathbf{v}}}{\overline{w}_{\mathbf{v}}} \widehat{y}_{\mathbf{v}} + \frac{1}{2} \frac{\overline{w}_{\mathbf{v}0} \overline{w}_{\mathbf{v}1}}{\overline{w}_{\mathbf{v}}} \widetilde{y}_{\mathbf{v}a} \, .$$

The prediction $\widehat{f}_t(x)$ of the tree is given by $\widetilde{y}_\epsilon$, which is the last value obtained in this recursion. Let us recall that this computes the aggregation with exponential weights of all the decision functions produced by all the prunings of the current Mondrian tree, as described in Definition 3.3 and stated in Proposition 3.1 above. The prediction procedure is summarized in Algorithm 4 below.

The next Section 3.3 provides theoretical guarantees for AMF, but before that, let us provide the following numerical illustration on three toy datasets for binary classification. The aim of this illustration is to exhibit the effect of aggregation in AMF, compared to the same method with no aggregation, the original Mondrian Forest algorithm, batch Random Forest and Extra Trees (see Section 3.5 for a precise description of the implementations used). We observe that AMF with aggregation (AMF(agg)) produces a very smooth decision function in all cases, which generalizes better on this instance (AUCs displayed on the bottom right-hand side of each plot are computed on a 30% test dataset) than all other methods. All

Algorithm 4 `AmfPredict`(x) : predict the label of $x \in [0,1]^d$

```
1: Input: a features vector $x \in [0,1]^d$
2: Follow Lines 2–10 of Algorithm 3 to do a temporary update of the current partition $\Pi$
   using $x$ and let $\mathbf{v}(x)$ be the leaf such that $x \in C_{\mathbf{v}(x)}$
3: Set $\widetilde{y}_{\mathbf{v}} = \widehat{y}_{\mathbf{v}(x)}$
4: while $\mathbf{v} \neq \epsilon$ do
5:   Let $(\mathbf{v}, \mathbf{v}a) = (\mathtt{parent}(\mathbf{v}), \mathbf{v})$ (for some $a \in \{0,1\}$)
6:   Let $\widetilde{y}_{\mathbf{v}} = \frac{1}{2}\frac{w_{\mathbf{v}}}{\overline{w}_{\mathbf{v}}}\widehat{y}_{\mathbf{v}} + \frac{1}{2}\frac{\overline{w}_{\mathbf{v}(1-a)}\overline{w}_{\mathbf{v}a}}{\overline{w}_{\mathbf{v}}}\widetilde{y}_{\mathbf{v}a}$
7: end while
8: Return $\widetilde{y}_\epsilon$
```

the other algorithms display rather non-smooth decision functions, which suggests that the underlying probability estimates are not well-calibrated.

3.3 Theoretical guarantees

In addition to being efficiently implementable in a streaming fashion, AMF is amenable to a thorough end-to-end theoretical analysis. This relies on two main ingredients: (i) a precise control of the geometric properties of the Mondrian partitions and (ii) a regret analysis of the aggregation procedure (exponentially weighted aggregation of all finite prunings of the infinite Mondrian) which in turn yields excess risk bounds and adaptive minimax rates. The guarantees provided below hold for a single tree in the Forest, but hold also for the average of several trees (used in by the forest) by convexity of the loss (see Examples 3.1 and 3.2).

3.3.1 Regret bounds

For now, the sequence $(x_1, y_1), \dots, (x_n, y_n) \in [0,1]^d \times \mathcal{Y}$ is arbitrary, and is in particular not required to be i.i.d. Let us recall that at step t, we have a realization $\Pi_t = (\mathcal{T}_t, \Sigma_t)$ of a finite Mondrian tree, which is the minimal subtree of the infinite Mondrian partition $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi)$ that separates all distinct sample points in $\{x_1, \dots, x_t\}$. Let us recall also that $\widehat{y}_{\mathcal{T},t} : [0,1]^d \to \widehat{\mathcal{Y}}$ are the tree forecasters from Definition 3.2, where $\mathcal{T}$ is some subtree of $\mathcal{T}^\Pi$. We need the following

Definition 3.4. Let $\eta > 0$. A loss function $\ell : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbf{R}$ is said to be *η-exp-concave* if the function $\exp(-\eta\,\ell(\cdot, y)) : \widehat{\mathcal{Y}} \to \mathbf{R}$ is concave for each $y \in \mathcal{Y}$.

The following loss functions are η-exp-concave:

- The *logarithmic loss* $\ell(\widehat{y}, y) = -\log \widehat{y}(y)$, with $\mathcal{Y}$ a finite set and $\widehat{\mathcal{Y}} = \mathcal{P}(\mathcal{Y})$, with $\eta = 1$ (see Example 3.2 above);
- The *quadratic loss* $\ell(\widehat{y}, y) = (\widehat{y} - y)^2$ on $\mathcal{Y} = \widehat{\mathcal{Y}} = [-B, B] \subset \mathbf{R}$, with $\eta = 1/(8B^2)$.

We start with Lemma 3.1, which states that the prediction function used in AMF (see Definition 3.3) satisfies a regret bound where the regret is computed with respect to any pruning $\mathcal{T}$ of $\mathcal{T}^\Pi$.

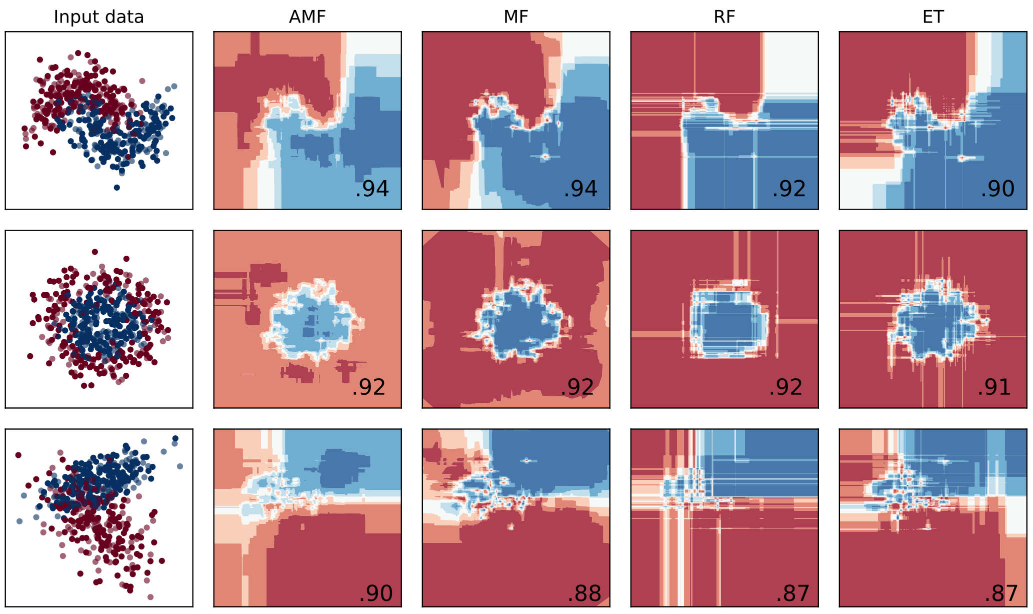

Figure 3.3: Decision functions of AMF, Mondrian Forest (MF), Breiman's batch random forest (RF) and batch Extra Trees (ET), on several toy datasets for binary classification (input data $n = 500$). We observe that AMF, thanks to aggregation, leads to a smooth decision function with a better generalization property (AUC on the test sets, displayed bottom right of each plot, is slightly better in all cases). Let us stress that both AMF and MF do a *single pass* on the data, while RF and ET require many passes. All algorithms use a forest containing 10 trees.

Lemma 3.1. *Consider a η-exp-concave loss function ℓ. Fix a realization $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi) \sim \mathsf{MP}$ and let $\mathcal{T} \subset \mathcal{T}^\Pi$ be a finite subtree. For every sequence $(x_1, y_1), \ldots, (x_n, y_n)$, the prediction functions $\widehat{f}_1, \ldots, \widehat{f}_n$ based on Π and computed by AMF satisfy*

$$\sum_{t=1}^n \ell(\widehat{f}_t(x_t), y_t) - \sum_{t=1}^n \ell(\widehat{y}_{\mathcal{T},t}(x_t), y_t) \leqslant \frac{1}{\eta} |\mathcal{T}| \log 2, \tag{3.5}$$

where we recall that $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$.

Lemma 3.1 is a direct consequence of a standard regret bound for the exponential weights algorithm (see Lemma 3.4 from Section 3.7), together with the fact that the Context Tree Weighting algorithm performed in Algorithms 3 and 4 computes it exactly, as stated in Proposition 3.1. By combining Lemma 3.1 with regret bounds for the online algorithms used in each node, both for the logarithmic loss (see Example 3.2) and the quadratic loss (see Example 3.1), we obtain the following regret bounds with respect to any pruning $\mathcal{T}$ of $\mathcal{T}^\Pi$.

Corollary 3.1 (Classification)**.** *Fix $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi)$ as in Lemma 3.1 and consider the classification setting described in Example 3.2 above. For any finite subtree $\mathcal{T}$ of $\mathcal{T}^\Pi$ and every sequence $(x_1, y_1), \ldots, (x_n, y_n)$, the prediction functions $\widehat{f}_1, \ldots, \widehat{f}_n$ based on Π computed by AMF with $\eta = 1$ satisfy*

$$\sum_{t=1}^n \ell(\widehat{f}_t(x_t), y_t) - \sum_{t=1}^n \ell(g_{\mathcal{T}}(x_t), y_t) \leqslant |\mathcal{T}| \log 2 + \frac{(|\mathcal{T}| + 1)(|\mathcal{Y}| - 1)}{4} \log(4n) \tag{3.6}$$

for any function $g_{\mathcal{T}} : [0,1]^d \to \mathcal{P}(\mathcal{Y})$ which is constant on the leaves of $\mathcal{T}$.

Corollary 3.2 (Regression)**.** *Fix $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi)$ as in Lemma 3.1 and consider the regression setting described in Example 3.1 above with $\mathcal{Y} = [-B, B]$. For every finite subtree $\mathcal{T}$ of $\mathcal{T}^\Pi$ and every sequence $(x_1, y_1), \ldots, (x_n, y_n)$, the prediction functions $\widehat{f}_1, \ldots, \widehat{f}_n$ based on Π computed by AMF with $\eta = 1/(8B^2)$ satisfy*

$$\sum_{t=1}^n \ell(\widehat{f}_t(x_t), y_t) - \sum_{t=1}^n \ell(g_{\mathcal{T}}(x_t), y_t) \leqslant 4B^2(|\mathcal{T}| + 1) \log n \tag{3.7}$$

for any function $g_{\mathcal{T}} : [0,1]^d \to \mathcal{Y}$ which is constant on the leaves of $\mathcal{T}$.

The proofs of Corollaries 3.1 and 3.2 are given in Section 3.7, and rely in particular on Lemmas 3.5 and 3.6 that provide regret bounds for the online predictors $\widehat{y}_{\mathbf{v},t}$ considered in the nodes. Corollaries 3.1 and 3.2 that control the regret with respect to any pruning of $\mathcal{T}^\Pi$ imply in particular regret bounds with respect to any *time pruning* of MP.

Definition 3.5 (Time pruning)**.** For $\lambda > 0$, the *time pruning* Π_λ of Π at time λ is obtained by removing any node $\mathbf{v}$ whose creation time $\tau_{\mathbf{v}}$ satisfies $\tau_{\mathbf{v}} > \lambda$. We denote by $\mathsf{MP}(\lambda)$ the distribution of the tree partition Π_λ of $[0,1]^d$.

The parameter λ corresponds to a complexity parameter, allowing to choose a subtree of $\mathcal{T}^\Pi$ where all leaves have a creation time not larger than λ. We obtain the following regret bound for the regression setting (a similar statement holds for the classification setting), where the regret is with respect to any time pruning Π_λ of Π.

Corollary 3.3. *Consider the same regression setting as in Corollary 3.2. Then, AMF with $\eta = 1/(8B^2)$ satisfies*

$$\mathbb{E}\Big[\sum_{t=1}^{n} \ell(\widehat{f}_t(x_t), y_t)\Big] \leqslant \mathbb{E}\Big[\inf_g \sum_{t=1}^{n} \ell(g(x_t), y_t)\Big] + 8B^2(1+\lambda)^d \log n \,, \tag{3.8}$$

where the expectations on both sides are over the random sampling of the partition $\Pi_\lambda \sim \mathsf{MP}(\lambda)$, and the infimum is taken over all functions $g : [0,1]^d \to \mathbf{R}$ that are constant on the cells of Π_λ.

Corollary 3.3 controls the regret of AMF with respect to a time pruned Mondrian partition $\Pi_\lambda \sim \mathsf{MP}(\lambda)$ for *any* $\lambda > 0$. This result is one of the main ingredients allowing to prove that AMF is able to adapt to the unknown smoothness of the regression function, as stated in Theorem 3.2 below. Corollary 3.3, proven in Section 3.7, follows from the fact that $\mathbb{E}[|\mathcal{L}(\Pi_\lambda)|] = (1+\lambda)^d$ whenever $\Pi_\lambda \sim \mathsf{MP}(\lambda)$, where $|\mathcal{L}(\Pi_\lambda)|$ stands for the number of leaves in the partition Π_λ (Proposition 2.2). Let us pause for a minute and discuss the choice of the prior used in AMF, compared to what is done in literature with Bayesian approaches for instance.

Prior choice. The use of a branching process prior on prunings of large trees is common in literature on Bayesian regression trees. Indeed, Chipman et al. (1998) choose a branching process prior on subtrees, with a splitting probability of each node $\mathbf{v}$ of the form

$$\alpha(1 + d_{\mathbf{v}})^{-\beta} \tag{3.9}$$

for some $\alpha \in (0,1)$ and $\beta \geqslant 0$, where $d_{\mathbf{v}}$ is the depth of note $\mathbf{v}$. Note that the locations of the splits themselves are also parameters of the Bayesian model, which enables more flexible estimation, but prevents efficient closed-form computations. The same prior on subtrees is used in the BART algorithm (Chipman et al., 2010), which considers sums of trees. We note that several values are proposed for the parameters (α, β), although there does not appear to be any definitive choice or criterion (Chipman et al. 1998 considers several examples with $\beta \in [\frac{1}{2}, 2]$, while Chipman et al. 2010 suggest $(\alpha, \beta) = (0.95, 2)$ for BART). The prior π considered here (see Definition 3.3) has a splitting probability $1/2$ for each node, so that $(\alpha, \beta) = (1/2, 0)$. One appeal of the regret bounds stated above is that it offers guidance on the choice of parameters. Indeed, it follows as a by-product of our analysis that the regret of AMF (with any prior π) with respect to a subtree $\mathcal{T}$ is $O(\log \pi(\mathcal{T})^{-1})$. This suggests to choose π in AMF as flat as possible, namely $(\alpha, \beta) = (1/2, 0)$.

3.3.2 Adaptive minimax rates through online to batch conversion

In this Section, we show how to turn the algorithm described in Section 3.2 into a supervised learning algorithm with generalization guarantees that entail as a by-product adaptive minimax rates for nonparametric estimation. Namely, this section is concerned with bounds on the risk (expected prediction error on unseen data) rather than the regret of the sequence of prediction functions that was studied in Section 3.3.1. Therefore, we assume in this Section that the sequence $(x_1, y_1), (x_2, y_2), \ldots$ consists of i.i.d. random variables in $[0,1]^d \times \mathcal{Y}$, such that each (x_t, y_t) that comes sequentially is distributed as some generic pair (x, y). The quality of a *prediction function* $g : [0,1]^d \to \mathcal{Y}$ is measured by its risk defined as

$$R(g) = \mathbb{E}[\ell(g(x), y)] \,. \tag{3.10}$$

Online to batch conversion. Our supervised learning algorithm remains *online* (it does not require the knowledge of a fixed number of points n in advance). It is also virtually parameter-free, the only parameter being the learning rate η (set to 1 for the log-loss). In order to obtain a supervised learning algorithm with provable guarantees, we use *online to batch* conversion from Cesa-Bianchi et al. (2004), which turns any regret bound for an online algorithm into an excess risk bound for the average or a randomization of the past values of the online algorithm. As explained below, it enables to obtain fast rates for the excess risk, provided that the online procedure admits appropriate regret guarantees.

Lemma 3.2 (Online to batch conversion)**.** *Assume that the loss function $\ell : \widehat{\mathcal{Y}} \times \mathcal{Y} \to \mathbf{R}^+$ is measurable, with $\widehat{\mathcal{Y}}$ a measurable space, and let $\mathcal{G}$ be a class of measurable functions $[0,1]^d \to \widehat{\mathcal{Y}}$. Given $f_1, \ldots, f_n$ where $f_t : ([0,1]^d \times \mathcal{Y})^{t-1} \to \widehat{\mathcal{Y}}^{[0,1]^d}$, we denote $\widehat{f}_t = f_t((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$. Let $\widetilde{f}_n = \widehat{f}_{I_n}$ with I_n a random variable uniformly distributed on $\{1, \ldots, n\}$. Then, we have*

$$\mathbb{E}[R(\widetilde{f}_n)] - R(g) = \frac{1}{n}\mathbb{E}\Big[\sum_{t=1}^{n} \big(\ell(\widehat{f}_t(x_t), y_t) - \ell(g(x_t), y_t)\big)\Big], \tag{3.11}$$

which entails that the expected excess risk of $\widetilde{f}_n$ with respect to any $g \in \mathcal{G}$ is equal to the expected per-round regret of $\widehat{f}_1, \ldots, \widehat{f}_n$ with respect to g.

Although this result is well-known (Cesa-Bianchi et al., 2004), we provide for completeness a proof of this specific formulation in Section 3.7. In our case, $\mathcal{G}$ will be the (random) family of functions that are constant on the leaves of some pruning of an infinite Mondrian partition Π, and $\widehat{f}_1, \ldots, \widehat{f}_n$ will be the sequence of prediction functions of AMF. Note that, when conditioning on Π which is used to define both the class $\mathcal{G}$ and the algorithm, both $\mathcal{G}$ and the maps $f_1, \ldots, f_n$ become deterministic, so that we can apply Lemma 3.2 conditionally on Π. In what follows, we denote by $\widetilde{f}_n$ the outcome of the online to batch conversion applied to our online procedure.

Oracle inequality and minimax rates. Let us show now that $\widetilde{f}_n$ achieves adaptive minimax rates under nonparametric assumptions, which complements and improves previous results (Mourtada et al., 2017, 2018). Indeed, AMF addresses the practical issue of optimally tuning the complexity parameter λ of Mondrian trees, while remaining a very efficient online procedure. As the next result shows, the procedure $\widetilde{f}_n$, which is virtually parameter-free, performs at least almost as well as the Mondrian tree with the best λ chosen with hindsight. For the sake of conciseness, Theorems 3.1 and 3.2 are stated only in the regression setting, although a similar result holds for the log-loss.

Theorem 3.1. *Consider the same setting as in Corollary 3.2, the only difference being the fact that the sequence $(x_1, y_1), \ldots, (x_n, y_n)$ is i.i.d. and consider the online to batch conversion $\widetilde{f}_n$ from Lemma 3.2 applied to AMF. For every $\lambda > 0$ and every function g_λ which is constant on the cells of a random partition $\Pi_\lambda \sim \mathsf{MP}(\lambda)$, we have*

$$\mathbb{E}[R(\widetilde{f}_n)] - \mathbb{E}[R(g_\lambda)] \leqslant 8B^2(1+\lambda)^d \frac{\log n}{n}. \tag{3.12}$$

The proof of Theorem 3.1 is given in Section 3.7. It provides an oracle bound which is distribution-free, since it requires no assumption on the joint distribution of (x, y) apart from

$\mathcal{Y} = [-B, B]$. Combined with previous results on Mondrian partitions (Mourtada et al., 2018), which enable to control the approximation properties of Mondrian trees, Theorem 3.1 implies that $\widetilde{f}_n$ is adaptive with respect to the smoothness of the regression function, as shown in Theorem 3.2.

Theorem 3.2. *Consider the same setting as in Theorem 3.1 and assume that the regression function $f^*(\cdot) = \mathbb{E}[y|x = \cdot]$ is β-Hölder with $\beta \in (0, 1]$ unknown. Then, we have*

$$\mathbb{E}[(\widetilde{f}_n(x) - f^*(x))^2] = O\Big(\Big(\frac{\log n}{n}\Big)^{2\beta/(d+2\beta)}\Big), \tag{3.13}$$

which is (up to the $\log n$ term) the minimax optimal rate of estimation over the class of β-Hölder functions.

The proof of Theorem 3.2 is given in Section 3.7. Theorem 3.2 states that the online to batch conversion $\widetilde{f}_n$ of AMF is adaptive to the unknown Hölder smoothness $\beta \in (0, 1]$ of the regression function since it achieves, up to the $\log n$ term, the minimax rate $n^{-2\beta/(d+2\beta)}$, see Stone (1982). It would be theoretically possible to modify the procedure in order to ensure adaptivity to higher regularities (say, up to some order $\bar{\beta} \in \mathbf{N} \setminus \{0\}$), by replacing the constant estimates inside each node by polynomials (of order $\bar{\beta} - 1$). However, this would lead to a numerically involved procedure, that is beyond the scope of the chapter. In addition, it is known that averaging can reduce the bias of individual randomized tree estimators for twice differentiable functions, see Arlot and Genuer (2014) and Mourtada et al. (2018) for Mondrian Forests. Such results cannot be applied to AMF, since its decision function involves a more complicated process of aggregation over all subtrees.

3.4 Practical implementation of AMF

This section describes a modification of AMF that we use in practice, in particular for all the numerical experiments performed in the present chapter. Because of extra technicalities involved with the modified version described below, we are not able to provide theoretical guarantees similar to what is done in Section 3.3. Indeed, the procedure described in this section exhibits a more intricate behaviour: new splits may be inserted above previous splits, which affects the underlying tree structure as well as the underlying prior over subtrees. This section mainly modifies the procedures described in Algorithms 3 and 4 so that splits are sampled only within the range of the features seen in each node, see Section 3.4.1, with motivations to do so described below. Moreover, we provide in Section 3.4.2 a guarantee on the average computational complexity of AMF through a control of the expected depth of the Mondrian partition.

3.4.1 Restriction to splits within the range of sample points

Algorithm 3 from Section 3.2.3 samples successive splits on the whole domain $[0, 1]^d$. In particular, when a new features vector x_t is available, it samples splits of the leaf $\mathbf{v}_t$ containing x_t until a split successfully separates x_t from the other point $x_s \neq x_t$ contained in $\mathbf{v}_t$ (unless $\mathbf{v}_t$ was empty). In the process, several splits outside of the box containing x_s and x_t can be performed. These splits are somewhat superfluous, since they induce empty leaves and delay the split that separates these two points. Removing those splits is critical to the performance of the method, in particular when the ambient dimension of the features is not small. In such cases, many splits may be needed to separate the feature points. On the other hand, only keeping those splits that are necessary to separate the sample points may yield a more adaptive partition, which can better adapt to a possible low-dimensional structure of the distribution of x.

We describe below a modified algorithm that samples splits in the range of the features vectors seen in each cell, exactly as in the original Mondrian Forest algorithm (Lakshminarayanan et al., 2014). In particular, each leaf will contain exactly one sample point by construction (possibly with repetition if $x_s = x_t$ for some $s \neq t$) and no empty leaves. Formally, this procedure amounts to considering the *restriction* of the Mondrian partition to the finite set of points $\{x_1, \ldots, x_t\}$ (Lakshminarayanan et al., 2014), where it is shown that such a restricted Mondrian partition can be updated efficiently in an online fashion, thanks to properties of the Mondrian process. This update exploits the creation time $\tau_{\mathbf{v}}$ of each node, as well as the range of the features vectors $R_{\mathbf{v}}$ seen inside each node (as opposed to only leaves). Moreover, this procedure can possibly split an interior node and not only a leaf. The algorithm considered here is a modification of the procedure `ExtendMondrianTree`$(\mathcal{T}, \lambda, (x_t, y_t))$ described in Lakshminarayanan et al. (2014), where we use $\lambda = +\infty$ and where we perform the exponentially weighted aggregation of subtrees described in Section 3.2.

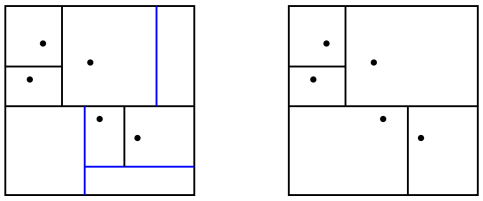

Figure 3.4: Unrestricted (left) *vs.* restricted (right) Mondrian partitions. Dots (•) represent sample points. In both cases, cells containing one sample point are no longer split. In addition, the restricted Mondrian partition is obtained by removing from the unrestricted partition all splits (in blue) that create empty leaves.

We call the former partition (from Section 3.2.3) an *unrestricted* Mondrian partition, while the one described here will be referred to as a *restricted* Mondrian partition. The difference between the two is illustrated in Figure 3.4. The tree $\mathcal{T}$ contains, as before, parent and children relations between all nodes $\mathbf{v} \in \mathcal{T}$, while each $\sigma_{\mathbf{v}} \in \Sigma$ contains

$$\sigma_{\mathbf{v}} = (j_{\mathbf{v}}, s_{\mathbf{v}}, \tau_{\mathbf{v}}, \widehat{y}_{\mathbf{v}}, w_{\mathbf{v}}, \overline{w}_{\mathbf{v}}, R_{\mathbf{v}}), \tag{3.14}$$

which differs from Equation (3.4) since we keep in memory the creation time $\tau_{\mathbf{v}}$ of $\mathbf{v}$, and the range

$$R_{\mathbf{v}} = \prod_{j=1}^{d} [a_{\mathbf{v}}^j, b_{\mathbf{v}}^j]$$

of features vectors in $C_{\mathbf{v}}$ instead of $x_{\mathbf{v}}$ (a past sample point). Another advantage of the restricted Mondrian partition is that the algorithm is *range-free*, since it does not require to assume that all features vectors are in $[0, 1]^d$ (we simply use as initial root cell $C_\epsilon = \mathbf{R}^d$).

Algorithms 5 and 6 below implement AMF with a restricted Mondrian partition, and are used instead of the previous Algorithm 3 in our numerical experiments. These algorithms, together with Algorithm 7 below for prediction, maintain in memory, as in Section 3.2, the current state of the Mondrian partition $\Pi = (\mathcal{T}, \Sigma)$, which contains the tree structure $\mathcal{T}$ (containing parent/child relationships between nodes) and data $\sigma_{\mathbf{v}} \in \Sigma$ for all nodes, see Equation (3.14). An illustration of Algorithms 5 and 6 is provided in Figure 3.5. We use the notation $x_+ = \max(x, 0)$ for any $x \in \mathbf{R}$.

Algorithm 5 `AmfUpdate`(x, y) : update AMF with a new sample $(x, y) \in \mathbf{R}^d \times \mathcal{Y}$

```
1: Input: a new sample $(x, y) \in \mathbf{R}^d \times \mathcal{Y}$
2: if $\mathcal{T} = \varnothing$ then
3:     Put $\mathcal{T} = \{\epsilon\}$ and $(\tau_\epsilon, \widehat{y}_\epsilon, w_\epsilon, \overline{w}_\epsilon, R_\epsilon) = (0, h(\varnothing), 1, 1, \{x\})$
4: else
5:     Call NodeUpdate$(\epsilon, x)$ from Algorithm 6
6: end if
7: Let $\mathbf{v}(x)$ be the leaf such that $x \in C_{\mathbf{v}(x)}$ and put $\mathbf{v} = \mathbf{v}(x)$
8: Let continueUp = true.
9: while continueUp do
10:    Set $w_{\mathbf{v}} = w_{\mathbf{v}} \exp(-\eta \ell(\widehat{y}_{\mathbf{v}}, y))$
11:    Set $\overline{w}_{\mathbf{v}} = w_{\mathbf{v}}$ if $\mathbf{v}$ is a leaf and $\overline{w}_{\mathbf{v}} = \frac{1}{2} w_{\mathbf{v}} + \frac{1}{2} \overline{w}_{\mathbf{v}0} \overline{w}_{\mathbf{v}1}$ otherwise
12:    Update $\widehat{y}_{\mathbf{v}}$ using $y$ (following Definition 3.2)
13:    If $\mathbf{v} \neq \epsilon$ let $\mathbf{v} =$ parent$(\mathbf{v})$, otherwise let continueUp = false
14: end while
```

In algorithm 5, Line 3 initializes the tree the first time `AmfUpdate` is called, otherwise the recursive procedure `NodeUpdate` is used to update the restricted Mondrian partition, starting at the root ϵ. Lines 7–14 perform the update of the aggregation weights in the same way as what we did in Section 3.2.3.

In Algorithm 6, Line 2 computes the range extension of x with respect to $R_{\mathbf{v}}$. In particular, if $x \in R_{\mathbf{v}}$, then no split will be performed and we go directly to Line 15. Otherwise, if x is outside of $R_{\mathbf{v}}$, a split of $\mathbf{v}$ is performed whenever $\tau_{\mathbf{v}} + E < \tau_{\mathbf{v}0}$ (a new node created at time $\tau_{\mathbf{v}} + E$ can be inserted before the creation time $\tau_{\mathbf{v}0}$ of the current child $\mathbf{v}0$ of $\mathbf{v}$). In this case, we sample the split coordinate j proportionally to Δ_j (coordinates with the largest extension are more likely to be used to split $\mathbf{v}$) and we sample the split threshold uniformly at random within the corresponding extension (Line 7 or Line 9). Now, at Line 11, we move downwards the whole tree rooted at $\mathbf{v}$: any node at index $\mathbf{v}\mathbf{v}'$ for any $\mathbf{v}' \in \mathcal{T}_{\mathbf{v}}$ is renamed as $\mathbf{v}(1-a)\mathbf{v}'$. For instance, if $a = 0$ (Line 7, the extension is on the left of the current range), the node $\mathbf{v}0$ is renamed as $\mathbf{v}10$, the node $\mathbf{v}1$ as $\mathbf{v}11$, etc. Then, at Line 12, new nodes $\mathbf{v}0$ and $\mathbf{v}1$ are created, where $\mathbf{v}a$ is a new leaf containing x and $\mathbf{v}(1-a)$ is a new node which is the root of the subtree we moved downwards at Line 11. Line 12 also initializes $\sigma_{\mathbf{v}a}$ and copies $\sigma_{\mathbf{v}}$ into $\sigma_{\mathbf{v}(1-a)}$. The process performed in Lines 11–12 therefore simply inserts two new nodes below $\mathbf{v}$ (since we just split node $\mathbf{v}$): a leaf containing x, and another node rooting the tree that was rooted at $\mathbf{v}$ before the split. Line 13 updates the range of $\mathbf{v}$ using x and exits the procedure. If no split is performed, Line 15 updates the range of $\mathbf{v}$ using x and calls `NodeUpdate` on the child of $\mathbf{v}$ containing x.

The prediction algorithm described in Algorithm 7 below is a modification of Algorithm 4, where we use `NodeUpdate` instead of Algorithm 3. Finally, the algorithm used in our experiments do not use the online to batch conversion from Section 3.3.2: it simply uses the current tree, namely the most recent updated Mondrian tree partition $\Pi_{t+1} = (\mathcal{T}_{t+1}, \Sigma_{t+1})$

Algorithm 6 `NodeUpdate`$(\mathbf{v}, x)$: update node $\mathbf{v}$ using x

```
1: Input: a node $\mathbf{v} \in \mathcal{T}$ from the current tree and a features vector $x \in \mathbf{R}^d$
2: Let $\Delta_j = (x_j - b_{\mathbf{v}}^j)_+ + (a_{\mathbf{v}}^j - x_j)_+$ and $\Delta = \sum_{j=1}^d \Delta_j$
3: Sample $E \sim \mathsf{Exp}(\Delta)$ and put $E = +\infty$ if $\Delta = 0$ (namely $x \in R_{\mathbf{v}}$)
4: if $\mathbf{v}$ is a leaf or $\tau_{\mathbf{v}} + E < \tau_{\mathbf{v}0}$ then
5:     Sample a split coordinate $J \in \{1, \ldots, d\}$ with $\mathbb{P}(J = j) = \Delta_j / \Delta$
6:     if $x_J < a_{\mathbf{v}}^J$ then
7:         Put $a = 0$ and sample the split threshold $S|J \sim \mathcal{U}([x_J, a_{\mathbf{v}}^J])$
8:     else
9:         Put $a = 1$ and sample the split threshold $S|J \sim \mathcal{U}([b_{\mathbf{v}}^J, x_J])$
10:    end if
11:    Set $\mathcal{T}_{\mathbf{v}(1-a)} = \mathcal{T}_{\mathbf{v}}$, namely nodes $\mathbf{v}\mathbf{v}'$ are renamed as $\mathbf{v}(1-a)\mathbf{v}'$ for any $\mathbf{v}' \in \mathcal{T}_{\mathbf{v}}$
12:    Create nodes $\mathbf{v}0$ and $\mathbf{v}1$ and put $(\tau_{\mathbf{v}a}, \widehat{y}_{\mathbf{v}a}, w_{\mathbf{v}a}, \overline{w}_{\mathbf{v}a}, R_{\mathbf{v}a}) = (\tau_{\mathbf{v}} + E, h(\varnothing), 1, 1, \{x\})$,
       put $\sigma_{\mathbf{v}(1-a)} = \sigma_{\mathbf{v}}$ (see Equation 3.14) but set $\tau_{\mathbf{v}(1-a)} = \tau_{\mathbf{v}} + E$
13:    Put $a_{\mathbf{v}}^j = \min(a_{\mathbf{v}}^j, x_j)$ and $b_{\mathbf{v}}^j = \max(b_{\mathbf{v}}^j, x_j)$
14: else
15:    Put $a_{\mathbf{v}}^j = \min(a_{\mathbf{v}}^j, x_j)$ and $b_{\mathbf{v}}^j = \max(b_{\mathbf{v}}^j, x_j)$
16:    Let $a \in \{0, 1\}$ be such that $x \in C_{\mathbf{v}a}$ and call NodeUpdate$(\mathbf{v}a, x)$
17: end if
```

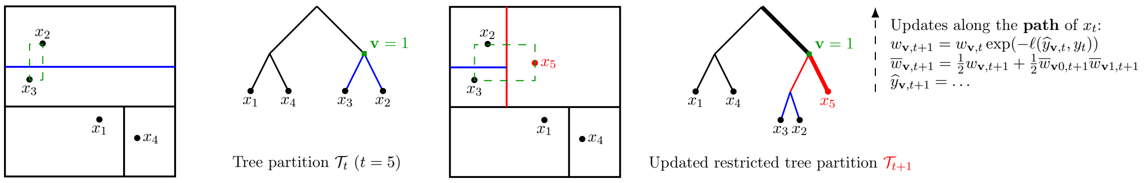
x_2
x_3
x_1
x_4
$\mathbf{v} = 1$
x_1 x_4 x_3 x_2
Tree partition $\mathcal{T}_t$ $(t = 5)$
x_2
x_5
x_3
x_1
x_4
$\mathbf{v} = 1$
x_1 x_4 x_5
x_3 x_2
Updated restricted tree partition $\mathcal{T}_{t+1}$
Updates along the **path** of x_t:
$w_{\mathbf{v},t+1} = w_{\mathbf{v},t} \exp(-\ell(\widehat{y}_{\mathbf{v},t}, y_t))$
$\overline{w}_{\mathbf{v},t+1} = \frac{1}{2} w_{\mathbf{v},t+1} + \frac{1}{2} \overline{w}_{\mathbf{v}0,t+1} \overline{w}_{\mathbf{v}1,t+1}$
$\widehat{y}_{\mathbf{v},t+1} = \ldots$

Figure 3.5: Illustration of the `AmfUpdate`(x_t, y_t) procedure from Algorithms 5 and 6: update of the partition, weights and node predictions as a new data point (x_t, y_t) for $t = 5$ becomes available. *Left*: tree partition Π_t before seeing (x_t, y_t). *Right*: update of the partition using (x_t, y_t). The path of x_t in the tree is indicated in bold. In green is the node $\mathbf{v} = 1$ and dashed lines indicates its range R_1. Since x_5 is outside of R_1 at $t = 5$, the range is extended. A new split (in red) is sampled in the extended range, since its creation time $\tau_1 + E$ is smaller than the one of the next split τ_{10} and two new nodes named 10 and 11 are inserted below 1, while the previous nodes 10 and 11 are moved as 100 and 101. The weights and node predictions are then updated using an upwards path from the new leaf containing x to the root (in bold). All leaves contain exactly one single point.

after calling `AmfUpdate`$(x_1, y_1), \ldots,$ `AmfUpdate`(x_t, y_t), where (x_t, y_t) is the last sample seen.

3.4.2 Computational complexity

The next Proposition provides a bound on the average depth of a Mondrian tree. This is of importance, since the computational complexities of `AmfUpdate` and `AmfPredict` are linear with respect to this depth, see below for a discussion.

Proposition 3.2. *Assume that x has a density p satisfying the following property: there exists*

Algorithm 7 `AmfPredict`(x): predict the label of $x \in [0,1]^d$

```
1: Input: a features vector $x \in [0,1]^d$
2: Call NodeUpdate$(\epsilon, x)$ in order to obtain a temporary update of the current partition $\Pi$
   using $x$ and let $\mathbf{v}(x)$ be the leaf such that $x \in C_{\mathbf{v}(x)}$
3: Set $\widetilde{y}_{\mathbf{v}} = \widehat{y}_{\mathbf{v}(x)}$
4: while $\mathbf{v} \neq \epsilon$ do
5:     Let $(\mathbf{v}, \mathbf{v}a) = (\mathtt{parent}(\mathbf{v}), \mathbf{v})$ (for some $a \in \{0,1\}$)
6:     Let $\widetilde{y}_{\mathbf{v}} = \frac{1}{2}\frac{w_{\mathbf{v}}}{\overline{w}_{\mathbf{v}}}\widehat{y}_{\mathbf{v}} + \frac{1}{2}\frac{\overline{w}_{\mathbf{v}(1-a)}\overline{w}_{\mathbf{v}a}}{\overline{w}_{\mathbf{v}}}\widetilde{y}_{\mathbf{v}a}$
7: end while
8: Return $\widetilde{y}_\epsilon$
```

a constant $M > 0$ such that, for every $x', x'' \in [0,1]^d$ which only differ by one coordinate,

$$\frac{p(x')}{p(x'')} \leqslant M\,. \tag{3.15}$$

Then, the depth $D_n^\Pi(x)$ of the leaf containing a random point x in the Mondrian tree restricted to the observations $x_1, \dots, x_n, x$ satisfies

$$\mathbb{E}[D_n^\Pi(x)] \leqslant \frac{\log n}{\log[(2M)/(2M-1)]} + 2M\,.$$

Assumption (3.15) is satisfied when p is upper and lower bounded: $c \leqslant p \leqslant C$ with $M = C/c$, but this assumption is weaker: for instance, it only implies that $M^{-d} \leqslant p \leqslant M^d$, which is a milder property when the dimension d is large. The proof of Proposition 3.2 is given in Section 3.7. Since the lower bound is also trivially $\Omega(\log n)$ (a binary tree with n nodes has at least a depth $\log_2 n$), Proposition 3.2 entails that $\mathbb{E}[D_n^\Pi] = \Theta(\log n)$. If the number of features is d, then the update complexity of a single tree is $\Theta(d \log n)$, which makes full online training time $\Theta(nd \log n)$ over a dataset of size n. Prediction is $\Theta(\log n)$ since it requires a downwards and upwards path on the tree (see Algorithms 4 and 7).

3.5 Numerical experiments

This Section proposes a thorough comparison of AMF with several baselines on several datasets for multi-class classification. We describe all the considered algorithms in Section 3.5.1, including both online methods and batch methods. A comparison of the average losses (assessing the online performance of algorithms) on several datasets is given in Section 3.5.3 for online methods only. Batch and online methods are compared in Section 3.5.4 and an experiment comparing the sensitivity of all methods with respect to the number of trees used is given in Section 3.5.5. All these experiments are performed for multi-class classification problems, using datasets described in Section 3.5.2.

3.5.1 Algorithms

In this Section, we describe precisely the procedures considered in our experiments for online and batch classification problems.

AMF. This is the AMF algorithm with restricted Mondrian partitions described in Algorithms 5 and 7. AMF is implemented in Python and C++ in our open-source `tick` library, available at `https://github.com/X-DataInitiative/tick`, and is documented here `https://x-datainitiative.github.io/tick/`. We use `OnlineForestClassifier` from the `tick.online` module, with default parameters: we use 10 trees; we use aggregation with exponential weights with learning rate $\eta = 1$; we don't split nodes containing only a single data class.

Dummy. We consider a dummy baseline that only estimates the distribution of the labels (without taking into account the features) in an online manner. At step $t+1$, it simply computes the Krichevsky-Trofimov forecaster (see Example 3.2) $\widehat{y}_{t+1}(k) = (n_t(k) + 1/2)/(t + K/2)$ of the classes $k = 1, \ldots, K$, where $n_t(k) = \sum_{s=1}^{t} \mathbf{1}(y_s = k)$.

MF (Mondrian Forest). This is the Mondrian Forest Lakshminarayanan et al. (2014, 2016) proposed in the `scikit-garden` library, available at `https://github.com/scikit-garden/scikit-garden`. We use `MondrianForestClassifier` in our experiments, with the default settings proposed with the method: 10 trees are used; no depth restriction is used on the trees; we stop growing the trees if all nodes have less than 2 samples; all trees are trained using the entire dataset (no bootstrap).

SGD (Stochastic Gradient Descent). This is logistic regression trained with a single pass of stochastic gradient descent. We use `SGDClassifier` from the `scikit-learn` library, see Pedregosa et al. (2011) and `https://scikit-learn.org`. We use a constant learning rate 0.1 and the default choice of ridge penalization with strength 0.0001, since it provides good results on all the datasets.

RF (Random Forests). This is Random Forest (Breiman, 2001a) for classification. We use the implementation available in the `scikit-learn` library, namely `RandomForestClassifer` from the `sklearn.ensemble` module. This is a reference implementation, which is highly optimized and among the fastest implementations available in the open-source community. Details on this implementation are available in Louppe (2014). Note that this is a batch algorithm, that cannot be trained sequentially, which requires a large number of passes through the data to optimize some impurity criterion (default is the Gini index). We use the default parameters of the procedure (with 10 trees).

ET (Extra Trees). This is the Extra Trees algorithm (Geurts et al., 2006). Once-again, we use the implementation available in the `scikit-learn` library, namely `ExtraTreesClassifier` from the `sklearn.ensemble` module. As for RF, it is a reference implementation from the open source community. We use the default parameters of the procedure (with 10 trees).

3.5.2 Considered datasets

The datasets we use are from the UCI Machine Learning repository, see Dua and Graff (2019) and are described in Table 3.2 below.

dataset	#samples	#features	#classes
adult	32561	107	2
bank	45211	51	2
car	1728	21	4
cardio	2126	24	3
churn	3333	71	2
default_cb	30000	23	2
letter	20000	16	26
satimage	5104	36	6
sensorless	58509	48	11
spambase	4601	57	2

Table 3.2: List of datasets from the UCI Machine Learning repository considered in our experiments.

3.5.3 Online learning: comparison of averaged losses

We compare the curves of averaged losses over time of all the considered online algorithms. At each round t, we reveal a new sample (x_t, y_t) and update all algorithms using this new sample. Then, we ask all algorithms to give a prediction $\widehat{y}_{t+1}$ of the label y_{t+1} associated to x_{t+1}, and compute the log-loss $\ell(\widehat{y}_{t+1}, y_{t+1})$ incurred by all algorithms. Along the rounds $t = 1, \ldots, n-1$ when the considered data has sample size n, we compute the average loss $\frac{1}{t-1}\sum_{s=1}^{t-1} \ell(\widehat{y}_{s+1}, y_{s+1})$. This is what is displayed in Figure 3.6 below, on 10 datasets for the online procedures AMF, Dummy, SGD and MF.

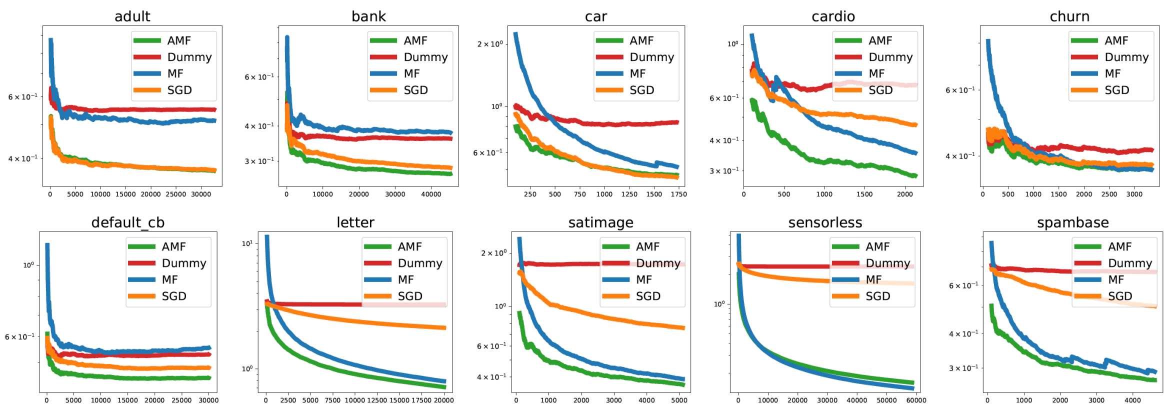

Figure 3.6: Average losses of all the online algorithms considered on 10 datasets for multi-class classification. The x-axis corresponds to the step t (number of samples revealed) and the y-axis is the value of average log loss obtained until this step (the lower the better). AMF almost always exhibits the smallest average loss on all the considered datasets.

On most datasets, AMF exhibits the smallest average loss, and is always competitive with respect to the considered baselines. As a comparison, the performance of SGD and MF strongly varies depending on the dataset: the "robustness" of AMF comes from the aggregation algorithm it uses, which always produces a non-overfitting and smooth decision function, as illustrated in Figure 3.3 above, even in the early iterations. This is confirmed by the early values of the average losses observed in all displays in Figure 3.6, where we see that it is always the smallest compared to all the baselines.

3.5.4 Online versus Batch learning

In this Section, we consider a "batch" setting, where we hold out a test dataset (containing 30% of the whole data), and we consider only binary classification problems, in which all methods are assessed using the area under the ROC curve (AUC) on the test dataset. We consider the datasets `adult`, `bank`, `default`, `spambase` and all the methods (online and batch) described in Section 3.5.1. The performances of batch methods (RF and ET) are assessed only once using the test set, since these methods are not trained in an online fashion, but rather at once. Therefore, the test AUCs of these batch methods are displayed in Figure 3.7 as a constant horizontal line along the iterations. Online methods (AMF, Dummy, SGD and MF) are tested every 100 iterations: each time 100 samples are revealed, we produce predictions on the full test dataset, and report the corresponding test AUCs in Figure 3.7. We observe that, as more samples are revealed, the online methods improve their test AUCs.

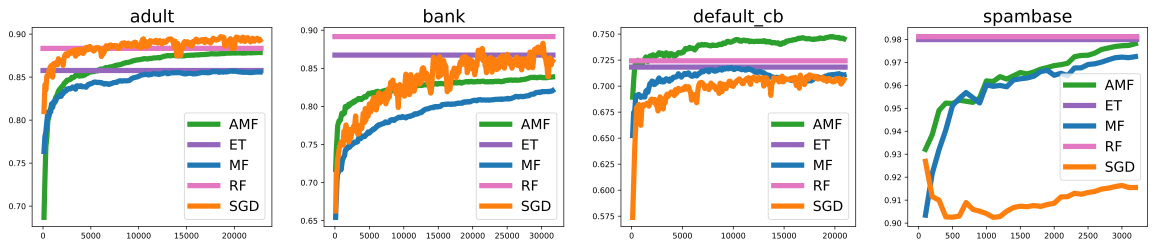

Figure 3.7: Area under the ROC curve (AUC) obtained on a held-out testing dataset (30% of the whole data) obtained by batch methods (RF and ET) and online methods (SGD, MF and AMF) on four binary classification datasets. The x-axis corresponds to the online steps (number of samples seen) over the train dataset. AMF is very competitive and always achieve a good AUC after a few steps. In the `defaultcb` dataset, AMF even improves the test AUC of RF and ET.

We observe that the batch methods RF and ET generally perform best; this ought to be expected, since their splits are optimized using training data, while those of AMF and MF are chosen on-the-fly. However, the performance of AMF is very competitive against all baselines. In particular, it performs better than MF and even improves upon ET and RF on the `default_cv` dataset.

3.5.5 Sensitivity to the number of trees

The aim of this Section is to exhibit another positive effect of the aggregation algorithm used in AMF. Indeed, we illustrate in Figure 3.8 below the fact that AMF can achieve good performances using less trees than MF, RF and ET. This comes from the fact that even a single tree in AMF can be a good classifier, since the aggregation algorithm used in it (see Section 3.2.2) aggregates all the prunings of the Mondrian tree. This allows to avoid overfitting, even when a single tree is used, as opposed to the other tree-based methods considered here. We consider in Figure 3.8 the same experimental setting as in Section 3.5.4, and compare the test AUCs obtained on four binary classification problems for an increasing number of trees in all methods. The test AUCs obtained by all algorithms with $1, 2, 5, 10, 20$ and 50 trees are displayed in Figure 3.8, where the x-axis corresponds to the number of trees and the y-axis corresponds to the test AUC.

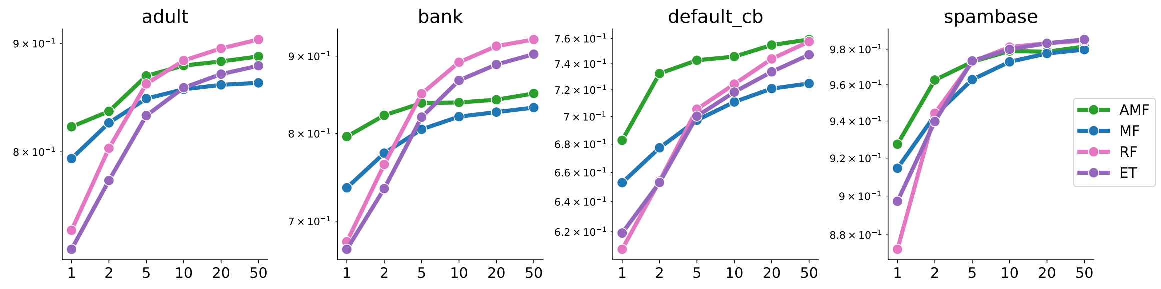

Figure 3.8: Area under the ROC curve (AUC) obtained on a held-out testing dataset (30% of the whole data) obtained by AMF, MF, RF and ET as a function of the number of trees used. We observe that AMF is less sensitive to the number of trees used in the forest than all the baselines, and that it has good performances even when using one or two trees.

We observe that when using one or two trees, AMF performs better than all the baselines. The performance of RF strongly increases with an increasing number of trees, and ends up with the best performances with 50 trees. The performance of AMF also improves when more trees are used (averaging over several realizations of the Mondrian partition certainly helps prediction), but the aggregation algorithm makes AMF somehow less sensitive to the number of trees in the forest.

3.6 Conclusion

In this chapter we introduced AMF, an online random forest algorithm based on a combination of Mondrian forests and an efficient implementation of an aggregation algorithm over all the prunings of the Mondrian partition. This algorithm is almost parameter-free, and has strong theoretical guarantees expressed in terms of regret with respect to the optimal time-pruning of the Mondrian partition, and in terms of adaptation with respect to the smoothness of the regression function. We illustrated on a large number of datasets the performances of AMF compared to strong baselines, where AMF appears as an interesting procedure for online learning.

A limitation of AMF, however, is that it does not perform feature selection. It would be interesting to develop an online feature selection procedure that could indicate along which coordinates the splits should be sampled in Mondrian trees, and prove that such a procedure performs dimension reduction in some sense. This is a challenging question in the context of online learning which deserves future investigations.

3.7 Proofs

This Section gathers the proofs of all the results of the Chapter, following their order of appearance, namely the proofs of Proposition 3.1, Lemma 3.1, Corollaries 3.1, 3.2 and 3.3, Lemma 3.2 and Theorems 3.1 and 3.2.

3.7.1 Proof of Proposition 3.1

Consider a realization $\Pi = (\mathcal{T}^\Pi, \Sigma^\Pi) \sim \mathsf{MP}$ of the infinite Mondrian partition, and assume that we are at step $t \geqslant 1$, namely we observed $(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})$ and performed the updates described in Algorithm 3 on each sample. Given $x \in [0,1]^d$, we want to predict the label (or its distribution) using

$$\widehat{f}_t(x) = \frac{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})} \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})}}, \tag{3.16}$$

see Definition 3.3, where we recall that $\pi(\mathcal{T}) = 2^{-|\mathcal{T}|}$ with $|\mathcal{T}|$ the number of nodes in $\mathcal{T}$ and where we recall that the sum in (3.16) is an infinite sum over all subtrees $\mathcal{T}$ of $\mathcal{T}^\Pi$. See also Definition 3.2 for the tree prediction $\widehat{y}_{\mathcal{T},t}(x)$.

Reduction to a finite sum. Let $\mathbf{T}$ denote the minimal subtree of $\mathcal{T}^\Pi$ that separates the elements of $\{x_1, \ldots, x_{t-1}, x\}$ (if $x = x_t$ then $\mathbf{T} = \mathcal{T}_{t+1}$). Also, for every finite tree $\mathcal{T}$, denote $\mathcal{T}|_{\mathbf{T}} := \mathcal{T} \cap \mathbf{T}$. For any subtree $\mathcal{T}$ of $\mathbf{T}$, we have

$$\sum_{\mathcal{T}' : \mathcal{T}'|_{\mathbf{T}} = \mathcal{T}} \pi(\mathcal{T}') = 2^{-\|\mathcal{T}\|} =: \pi_{\mathbf{T}}(\mathcal{T}), \tag{3.17}$$

where $\|\mathcal{T}\|$ denotes the number of nodes of $\mathcal{T}$ which are not leaves of $\mathbf{T}$; note that $\pi_{\mathbf{T}}$ is a probability distribution on the subtrees of $\mathbf{T}$, since π is a probability distribution on finite subtrees of $\{0,1\}^*$. To see why Equation (3.17) is true, consider the following representation of π: let $(B_{\mathbf{v}})_{\mathbf{v} \in \{0,1\}^*}$ be an i.i.d. family of Bernoulli random variables with parameter $1/2$; a node $\mathbf{v}$ is said to be *open* if $B_{\mathbf{v}} = 1$, and *closed* otherwise. Then, denote $\mathcal{T}'$ the subtree of $\{0,1\}^*$ all of whose interior nodes are open, and all of whose leaves are closed; clearly, $\mathcal{T}' \sim \pi$. Now, $\mathcal{T}'|_{\mathbf{T}} = \mathcal{T}$ if and only if all interior nodes of $\mathcal{T}$ are open and all leaves of $\mathcal{T}$ except leaves of $\mathbf{T}$ are closed. By independence of the $B_{\mathbf{v}}$, this happens with probability $2^{-\|\mathcal{T}\|}$.

In addition, note that if $\mathcal{T}'$ is a finite subtree of $\{0,1\}^*$ and $\mathcal{T} = \mathcal{T}'|_{\mathbf{T}}$, then $\widehat{y}_{\mathcal{T}',t}(x) = \widehat{y}_{\mathcal{T},t}(x)$. Indeed, let $\mathbf{v}_{\mathcal{T}'}(x)$ be the leaf of $\mathcal{T}'$ that contains x; if $\mathbf{v}_{\mathcal{T}'}(x) \in \mathbf{T}$, then $\mathbf{v}_{\mathcal{T}'}(x) = \mathbf{v}_{\mathcal{T}}(x)$ and hence $\widehat{y}_{\mathcal{T}',t}(x) = \widehat{y}_{\mathbf{v}_{\mathcal{T}'}(x),t} = \widehat{y}_{\mathbf{v}_{\mathcal{T}}(x),t} = \widehat{y}_{\mathcal{T},t}(x)$; otherwise, by definition of $\mathbf{T}$, both $\mathbf{v}_{\mathcal{T}'}(x)$ and $\mathbf{v}_{\mathcal{T}}(x)$ only contain the x_s ($s \leqslant t-1$) such that $x_s = x$, so that again $\widehat{y}_{\mathbf{v}_{\mathcal{T}'}(x),t} = \widehat{y}_{\mathbf{v}_{\mathcal{T}}(x),t}$. Similarly, this result for $x = x_t$ also holds for x_s, $s \leqslant t-1$, so that $L_{t-1}(\mathcal{T}') = L_{t-1}(\mathcal{T})$. From the points above, it follows that

$$\begin{aligned}
\widehat{f}_t(x) &= \frac{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})} \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \pi(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})}} \\
&= \frac{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \sum_{\mathcal{T}' : \mathcal{T}'|_{\mathbf{T}} = \mathcal{T}} \pi(\mathcal{T}') e^{-\eta L_{t-1}(\mathcal{T}')} \widehat{y}_{\mathcal{T}',t}(x)}{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \sum_{\mathcal{T}' : \mathcal{T}'|_{\mathbf{T}} = \mathcal{T}} \pi(\mathcal{T}') e^{-\eta L_{t-1}(\mathcal{T}')}} \\
&= \frac{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \sum_{\mathcal{T}' : \mathcal{T}'|_{\mathbf{T}} = \mathcal{T}} \pi(\mathcal{T}') e^{-\eta L_{t-1}(\mathcal{T})} \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subset \mathcal{T}^\Pi} \sum_{\mathcal{T}' : \mathcal{T}'|_{\mathbf{T}} = \mathcal{T}} \pi(\mathcal{T}') e^{-\eta L_{t-1}(\mathcal{T})}} \\
&= \frac{\sum_{\mathcal{T} \subset \mathbf{T}} \pi_{\mathbf{T}}(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})} \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subset \mathbf{T}} \pi_{\mathbf{T}}(\mathcal{T}) e^{-\eta L_{t-1}(\mathcal{T})}} .
\end{aligned} \tag{3.18}$$

Computation for the finite tree $\mathbf{T}$. The expression in Equation (3.18) involves finite sums, over all subtrees of $\mathbf{T}$ (involving an exponential in the number of leaves of $\mathbf{T}$, namely t, terms). However, it can be computed efficiently because of the specific choice of the prior π. More precisely, we will use the following lemma (Helmbold and Schapire, 1997, Lemma 1) several times to efficiently compute sums of products. Let us recall that $\mathcal{N}(\mathbf{T})$ stands for the set of nodes of $\mathbf{T}$.

Lemma 3.3. *Let $g : \mathcal{N}(\mathbf{T}) \to \mathbf{R}$ be an arbitrary function and define $G : \mathcal{N}(\mathbf{T}) \to \mathbf{R}$ as*

$$G(\mathbf{v}) = \sum_{\mathcal{T}_\mathbf{v}} 2^{-\|\mathcal{T}_\mathbf{v}\|} \prod_{\mathbf{w} \in \mathcal{L}(\mathcal{T}_\mathbf{v})} g(\mathbf{w}), \tag{3.19}$$

where the sum is over all subtrees $\mathcal{T}_\mathbf{v}$ of $\mathbf{T}$ rooted at $\mathbf{v}$. Then, $G(\mathbf{v})$ can be computed recursively as follows:

$$G(\mathbf{v}) = \begin{cases} g(\mathbf{v}) & \text{if } \mathbf{v} \in \mathcal{L}(\mathbf{T}) \\ \frac{1}{2} g(\mathbf{v}) + \frac{1}{2} G(\mathbf{v}0) G(\mathbf{v}1) & \text{otherwise,} \end{cases}$$

for each node $\mathbf{v} \in \mathcal{N}(\mathbf{T})$.

Let us introduce

$$w_t(\mathcal{T}) = \pi_\mathbf{T}(\mathcal{T}) \exp(-\eta L_{t-1}(\mathcal{T})),$$

so that Equation (3.16) writes

$$\widehat{f}_t(x) = \frac{\sum_{\mathcal{T} \subset \mathbf{T}} w_t(\mathcal{T}) \widehat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subset \mathbf{T}} w_t(\mathcal{T})}, \tag{3.20}$$

where the sums hold over all subtrees $\mathcal{T}$ of $\mathbf{T}$. We will show how to efficiently compute and update the numerator and denominator in Equation (3.20). Note that $w_t(\mathcal{T})$ may be written as

$$w_t(\mathcal{T}) = 2^{-\|\mathcal{T}\|} \prod_{\mathbf{v} \in \mathcal{L}(\mathcal{T})} w_{\mathbf{v},t} \tag{3.21}$$

with $w_{\mathbf{v},t} = \exp(-\eta L_{\mathbf{v},t-1})$, where $L_{\mathbf{v},t} := \sum_{s \leqslant t \,:\, X_t \in C_\mathbf{v}} \ell(\widehat{y}_{\mathbf{v},s}, y_s)$.

Denominator of Equation (3.20). For each node $\mathbf{v} \in \mathcal{N}(\mathbf{T})$ and every $t \geqslant 1$, denote

$$\overline{w}_{\mathbf{v},t} = \sum_{\mathcal{T}_\mathbf{v}} 2^{-\|\mathcal{T}_\mathbf{v}\|} \prod_{\mathbf{v}' \in \mathcal{L}(\mathcal{T}_\mathbf{v})} w_{\mathbf{v}',t} \tag{3.22}$$

so that (3.21) entails

$$\overline{w}_{\epsilon,t} = \sum_{\mathcal{T}} w_t(\mathcal{T}). \tag{3.23}$$

Using Equation (3.22), the weights $\overline{w}_{\mathbf{v},t}$ can be computed recursively using Lemma 3.3. We denote by $\texttt{path}(x_t)$ the path from ϵ to $\mathbf{v}_\mathbf{T}(x_t)$ (from the root to the leaf containing x_t). Note that, by definition of $w_{\mathbf{v},t}$, if $\mathbf{v} \notin \texttt{path}(x_t)$ (namely $x_t \notin C_\mathbf{v}$), we have $w_{\mathbf{v},t+1} = w_{\mathbf{v},t}$. In addition, if $\mathbf{v} \notin \texttt{path}(x_t)$, so are all its descendants, so that (by induction, and using the above recursive formula) $\overline{w}_{\mathbf{v},t+1} = \overline{w}_{\mathbf{v},t}$. In other words, *only the nodes of* $\texttt{path}(x_t)$ *have updated weights.*

As a result, at each round $t \geqslant 1$, after seeing $(x_t, y_t) \in [0,1]^d \times \mathcal{Y}$, the weights $w_{\mathbf{v},t}$ and $\overline{w}_{\mathbf{v},t}$ are updated for $\mathbf{v} \in \texttt{path}(x_t)$ as follows (note that they are all initialized at $w_{\mathbf{v},1} = \overline{w}_{\mathbf{v},1} = 1$):

- for every $\mathbf{v} \notin \texttt{path}(x_t)$, $w_{\mathbf{v},t+1} = w_{\mathbf{v},t}$ and $\overline{w}_{\mathbf{v},t+1} = \overline{w}_{\mathbf{v},t}$;
- for every $\mathbf{v} \in \texttt{path}(x_t)$, $w_{\mathbf{v},t+1} = w_{\mathbf{v},t} \exp(-\eta \ell(\widehat{y}_{\mathbf{v},t}, y_t))$;
- for every $\mathbf{v} \in \texttt{path}(x_t)$, we have

$$\overline{w}_{\mathbf{v},t+1} = \begin{cases} w_{\mathbf{v},t+1} & \text{if } \mathbf{v} \in \mathcal{L}(\mathbf{T}) \quad (\text{namely } \mathbf{v} = \mathbf{v_T}(x_t)), \\ \frac{1}{2} w_{\mathbf{v},t+1} + \frac{1}{2} \overline{w}_{\mathbf{v}0,t+1} \overline{w}_{\mathbf{v}1,t+1} & \text{otherwise.} \end{cases}$$

The weights $w_{\mathbf{v},t}, \overline{w}_{\mathbf{v},t}$ as well as the predictions $\widehat{y}_{\mathbf{v},t}$ are updated recursively in an “upwards” traversal of $\texttt{path}(x_t)$ in $\mathbf{T}$ (from $\mathbf{v_T}(x_t)$ to ϵ), as indicated in Algorithm 3.

Note that when updating the structure of the tree, the weights $w_{\mathbf{v},t+1}, \overline{w}_{\mathbf{v},t+1}$ and predictions $\widehat{y}_{\mathbf{v},t+1}$ for the newly created nodes in $\mathcal{T}_{t+1} \setminus \mathcal{T}_t$ (which are offsprings of $\mathbf{v}_{\mathcal{T}_t}(x_t)$ created from the splits necessary to separate x_t from the other point $x_s \in C_{\mathbf{v}_{\mathcal{T}_t}(x_t)}$) can be set depending on whether these nodes contain x_s or x_t. This does not affect the values of $w_{\mathbf{v},t}$ and $\widehat{y}_{\mathbf{v},t}$ at other nodes, but only the values of $\overline{w}_{\mathbf{v},t}$ for $\mathbf{v} \in \texttt{path}(x_t)$ that are computed in the upwards recursion.

Numerator of Equation (3.20). The numerator of Equation (3.20) can be computed in the same fashion as the denominator. Let $w'_{\mathbf{v},t} = w_{\mathbf{v},t}\widehat{y}_{\mathbf{v},t}$ if $\mathbf{v} \in \texttt{path}(x)$, and $w'_{\mathbf{v},t} = w_{\mathbf{v},t}$ otherwise. Additionally, let

$$\widehat{w}_{\mathbf{v},t} = \sum_{\mathcal{T}_\mathbf{v}} 2^{-\|\mathcal{T}_\mathbf{v}\|} \prod_{\mathbf{v}' \in \mathcal{L}(\mathcal{T}_\mathbf{v})} w'_{\mathbf{v}',t} \,.$$

Note that we have

$$\widehat{w}_{\epsilon,t} = \sum_{\mathcal{T}} 2^{-\|\mathcal{T}\|} \prod_{\mathbf{v}' \in \mathcal{L}(\mathcal{T})} w'_{\mathbf{v}',t} = \sum_{\mathcal{T}} w_t(\mathcal{T}) \widehat{y}_{\mathbf{v}_\mathcal{T}(x),t} = \sum_{\mathcal{T}} w_t(\mathcal{T}) \widehat{y}_t(\mathcal{T}) \,. \tag{3.24}$$

Lemma 3.3 with $g(\mathbf{v}) = w'_{\mathbf{v},t}$ (so that $G(\mathbf{v}) = \widehat{w}_{\mathbf{v},t}$) enables to recursively compute $\widehat{w}_{\mathbf{v},t}$ from $w'_{\mathbf{v},t}$. First, note that $w'_{\mathbf{v},t} = w_{\mathbf{v},t}$ for every $\mathbf{v} \notin \texttt{path}(x)$. Since every descendant $\mathbf{v}'$ of $\mathbf{v}$ is also outside of $\texttt{path}(x)$, it follows by induction that $\widehat{w}_{\mathbf{v},t} = \overline{w}_{\mathbf{v},t}$ for every $\mathbf{v} \notin \texttt{path}(x)$. It then remains to show how to compute $\widehat{w}_{\mathbf{v},t}$ for $\mathbf{v} \in \texttt{path}(x)$. This is done again recursively, starting from the leaf $\mathbf{v_T}(x)$ up to the root ϵ:

$$\widehat{w}_{\mathbf{v},t} = \begin{cases} w_{\mathbf{v},t}\widehat{y}_{\mathbf{v},t} & \text{if } \mathbf{v} = \mathbf{v_T}(x) \\ \frac{1}{2} w_{\mathbf{v},t}\widehat{y}_{\mathbf{v},t} + \frac{1}{2} \overline{w}_{\mathbf{v}(1-a),t} \widehat{w}_{\mathbf{v}a,t} & \text{otherwise, where } a \in \{0,1\} \text{ is such that } \mathbf{v}a \in \texttt{path}(x) \end{cases}$$

Finally, we define

$$\widetilde{y}_{\mathbf{v},t}(x) = \frac{\widehat{w}_{\mathbf{v},t}}{\overline{w}_{\mathbf{v},t}}$$

for each node $\mathbf{v} \in \mathbf{T}$. It follows from Equations (3.20), (3.23) and (3.24) that $\widehat{f}_t(x) = \widetilde{y}_{\epsilon,t}(x)$. Additionally, the recursive expression for $\overline{w}_{\mathbf{v},t}$ and $\widehat{w}_{\mathbf{v},t}$ imply that $\widetilde{y}_{\mathbf{v},t}$ can be computed recursively as well, in the upwards traversal from $\mathbf{v_T}(x)$ to ϵ: we set

$$\widetilde{y}_{\mathbf{v},t}(x) = \widehat{y}_{\mathbf{v},t}$$

for $\mathbf{v} = \mathbf{v_T}(x)$, otherwise we set

$$\widetilde{y}_{\mathbf{v},t}(x) = \frac{1}{2}\frac{w_{\mathbf{v},t}}{\overline{w}_{\mathbf{v},t}}\widehat{y}_{\mathbf{v},t} + \frac{1}{2}\frac{\overline{w}_{\mathbf{v}a,t}\overline{w}_{\mathbf{v}(1-a),t}}{\overline{w}_{\mathbf{v},t}}\widetilde{y}_{\mathbf{v}a,t}(x) = \frac{1}{2}\frac{w_{\mathbf{v},t}}{\overline{w}_{\mathbf{v},t}}\widehat{y}_{\mathbf{v},t} + \left(1 - \frac{1}{2}\frac{w_{\mathbf{v},t}}{\overline{w}_{\mathbf{v},t}}\right)\widetilde{y}_{\mathbf{v}a,t}(x),$$

where $a \in \{0, 1\}$ is such that $\mathbf{v}a \in \mathtt{path}(x)$. The recursions constructed above are precisely the ones describing AMF in Algorithms 3 and 4 from Section 3.2.3, so that this concludes the proof of Proposition 3.1. $\square$

3.7.2 Proofs of Lemma 3.1, Corollaries 3.1, 3.2, 3.3, Lemma 3.2, Theorems 3.1, 3.2 and Proposition 3.2

We start with some well-known lemmas that are used to bound the regret: Lemma 3.4 controls the regret with respect to each tree forecaster, while Lemmas 3.5 and 3.6 bound the regret of each tree forecaster with respect to the optimal labeling of its leaves.

Lemma 3.4 (Vovk, 1998)**.** *Let $\mathcal{E}$ be a countable set of experts and $\pi = (\pi_i)_{i\in\mathcal{E}}$ be a probability measure on $\mathcal{E}$. Assume that ℓ is η-exp-concave. For every $t \geqslant 1$, let $y_t \in \mathcal{Y}$, $\widehat{y}_{i,t} \in \widehat{\mathcal{Y}}$ be the prediction of expert $i \in \mathcal{E}$ and $L_{i,t} = \sum_{s=1}^t \ell(\widehat{y}_{i,s}, y_s)$ be its cumulative loss. Consider the predictions defined as*

$$\widehat{y}_t = \frac{\sum_{i\in\mathcal{E}} \pi_i\, e^{-\eta L_{i,t-1}}\widehat{y}_{i,t}}{\sum_{i\in\mathcal{E}} \pi_i\, e^{-\eta L_{i,t-1}}}. \tag{3.25}$$

Then, irrespective of the values of $y_t \in \mathcal{Y}$ and $\widehat{y}_{i,t} \in \widehat{\mathcal{Y}}$, we have the following regret bound

$$\sum_{t=1}^n \ell(\widehat{y}_t, y_t) - \sum_{t=1}^n \ell(\widehat{y}_{i,t}, y_t) \leqslant \frac{1}{\eta}\log\frac{1}{\pi_i} \tag{3.26}$$

for each $i \in \mathcal{E}$ and $n \geqslant 1$.

Lemma 3.5 (Tjalkens et al., 1993)**.** *Let ℓ be the logarithmic loss on the finite set $\mathcal{Y}$, and let $y_t \in \mathcal{Y}$ for every $t \geqslant 1$. The* Krichevsky-Trofimov (KT) *forecaster, which predicts*

$$\widehat{y}_t(y) = \frac{n_{t-1}(y) + 1/2}{(t-1) + |\mathcal{Y}|/2}, \tag{3.27}$$

with $n_{t-1}(y) = |\{1 \leqslant s \leqslant t-1 : y_s = y\}|$, satisfies the following regret bound with respect to the class $\mathcal{P}(\mathcal{Y})$ of constant experts (which always predict the same probability distribution on $\mathcal{Y}$):

$$\sum_{t=1}^n \ell(\widehat{y}_t, y_t) - \inf_{p\in\mathcal{P}(\mathcal{Y})}\sum_{t=1}^n \ell(p, y_t) \leqslant \frac{|\mathcal{Y}|-1}{2}\log(4n) \tag{3.28}$$

for each $n \geqslant 1$.

Lemma 3.6 (Cesa-Bianchi and Lugosi, 2006, p. 43)**.** *Consider the square loss $\ell(\widehat{y}, y) = (\widehat{y}-y)^2$ on $\mathcal{Y} = \widehat{\mathcal{Y}} = [-B, B]$, with $B > 0$. For every $t \geqslant 1$, let $y_t \in [-B, B]$. Consider the strategy defined by $\widehat{y}_1 = 0$, and for each $t \geqslant 2$,*

$$\widehat{y}_t = \frac{1}{t-1}\sum_{s=1}^{t-1} y_s\,. \tag{3.29}$$

The regret of this strategy with respect to the class of constant experts (which always predict some $b \in [-B, B]$) is upper bounded as follows:

$$\sum_{t=1}^{n} \ell(\widehat{y}_t, y_t) - \inf_{b \in [-B,B]} \sum_{t=1}^{n} \ell(b, y_t) \leqslant 8B^2(1 + \log n) \tag{3.30}$$

for each $n \geqslant 1$.

Proof of Lemma 3.1. This follows from Proposition 3.1 and Lemma 3.4. □

Proof of Corollary 3.1. Since the logarithmic loss is 1-exp-concave, Lemma 3.1 implies

$$\sum_{t=1}^{n} \ell(\widehat{f}_t(x_t), y_t) - \sum_{t=1}^{n} \ell(\widehat{y}_{\mathcal{T},t}(x_t), y_t) \leqslant |\mathcal{T}| \log 2 \tag{3.31}$$

for every subtree $\mathcal{T}$. It now remains to bound the regret of the tree forecaster $\mathcal{T}$ with respect to the optimal labeling of its leaves. By Lemma 3.5, for every leaf $\mathbf{v}$ of $\mathcal{T}$,

$$\sum_{1 \leqslant t \leqslant n \,:\, x_t \in C_{\mathbf{v}}} \ell(\widehat{y}_{\mathcal{T},t}(x_t), y_t) - \inf_{p_{\mathbf{v}} \in \mathcal{P}(\mathcal{Y})} \sum_{1 \leqslant t \leqslant n \,:\, x_t \in C_{\mathbf{v}}} \ell(p_{\mathbf{v}}, y_t) \leqslant \frac{|\mathcal{Y}| - 1}{2} \log(4N_{\mathbf{v},n})$$

where $N_{\mathbf{v},n} = |\{1 \leqslant t \leqslant n : x_t \in C_{\mathbf{v}}\}|$ (assuming that $N_{\mathbf{v},n} \geqslant 1$). Summing the above inequality over the leaves $\mathbf{v}$ of $\mathcal{T}$ such that $N_{\mathbf{v},n} \geqslant 1$ yields

$$\sum_{t=1}^{n} \ell(\widehat{y}_{\mathcal{T},t}(x_t), y_t) - \inf_{g_{\mathcal{T}}} \sum_{t=1}^{n} \ell(g_{\mathcal{T}}(x_t), y_t) \leqslant \frac{|\mathcal{Y}| - 1}{2} \sum_{\mathbf{v} \in \mathcal{L}(\mathcal{T}) \,:\, N_{\mathbf{v},n} \geqslant 1} \log(4N_{\mathbf{v},n}) \tag{3.32}$$

where $g_{\mathcal{T}}$ is any function constant on the leaves of $\mathcal{T}$. Now, letting $L = |\{\mathbf{v} \in \mathcal{L}(\mathcal{T}) : N_{\mathbf{v},n} \geqslant 1\}| \leqslant |\mathcal{L}(\mathcal{T})| = \frac{|\mathcal{T}|+1}{2}$, we have by concavity of the log

$$\begin{aligned}
\sum_{\mathbf{v} \in \mathcal{L}(\mathcal{T}) \,:\, N_{\mathbf{v},n} \geqslant 1} \log(4N_{\mathbf{v},n}) &\leqslant L \log \left(\frac{\sum_{\mathbf{v} \in \mathcal{L}(\mathcal{T}) \,:\, N_{\mathbf{v},n} \geqslant 1} 4N_{\mathbf{v},n}}{L} \right) \\
&= L \log \left(\frac{4n}{L} \right) \leqslant \frac{|\mathcal{T}| + 1}{2} \log(4n) \,.
\end{aligned}$$

Plugging this in (3.32) and combining with Equation (3.31) leads to the desired bound (3.6). □

Proof of Corollary 3.2. The proof proceeds similarly to that of Corollary 3.1, by combining Lemmas 3.1 and 3.6 and using the fact that the square loss is $\eta = 1/(8B^2)$-exp-concave on $[-B, B]$. □

Proof of Corollary 3.3. First, we reason conditionally on the Mondrian process Π. By applying Corollary 3.2 to $\mathcal{T} = \Pi_\lambda$, we obtain, since the number of nodes of Π_λ is $2|\mathcal{L}(\Pi_\lambda)| - 1$:

$$\sum_{t=1}^{n} \ell(\widehat{f}_t(x_t), y_t) - \inf_{g} \sum_{t=1}^{n} \ell(g(x_t), y_t) \leqslant 8B^2 |\mathcal{L}(\Pi_\lambda)| \log n \,, \tag{3.33}$$

where the infimum spans over all functions $g : [0,1]^d \to \widehat{\mathcal{Y}}$ which are constant on the cells of Π_λ. Corollary 3.3 follows by taking the expectation over Π and using the fact that $\Pi_\lambda \sim \mathsf{MP}(\lambda)$ implies $\mathbb{E}[|\mathcal{L}(\Pi_\lambda)|] = (1 + \lambda)^d$ (by Proposition 2.2 in Chapter 2). □

Proof of Lemma 3.2. For every $t = 1, \ldots, n$, $\widehat{f}_t$ is $\mathcal{F}_{t-1} := \sigma(x_1, y_1, \ldots, x_{t-1}, y_{t-1})$-measurable and since (x_t, y_t) is independent of $\mathcal{F}_t$:

$$\mathbb{E}[\ell(\widehat{f}_t(x_t), y_t)] = \mathbb{E}[\mathbb{E}[\ell(\widehat{f}_t(x_t), y_t)|\mathcal{F}_{t-1}]] = \mathbb{E}[R(\widehat{f}_t)]\,,$$

so that, for every $g \in \mathcal{G}$,

$$\frac{1}{n}\mathbb{E}\Big[\sum_{t=1}^{n}\big(\ell(\widehat{f}_t(x_t), y_t) - \ell(g(x_t), y_t)\big)\Big] = \frac{1}{n}\sum_{t=1}^{n}\mathbb{E}[R(\widehat{f}_t)] - R(g) = \mathbb{E}[R(\widetilde{f}_n)] - R(g)\,. \qquad \square$$

Proof of Theorem 3.1. This is a direct consequence of Lemma 3.2 and Corollary 3.2. $\square$

Proof of Theorem 3.2. Recall that the sequence $(x_1, y_1), \ldots, (x_n, y_n)$ is i.i.d and distributed as a generic pair $(x, y) \in [0,1]^d \times \mathcal{Y}$. Since $f^*(\cdot) = \mathbb{E}[y|x = \cdot]$, we have

$$R(f) = \mathbb{E}[(f(x) - f^*(x))^2] + R(f^*) \tag{3.34}$$

for every function $f : [0,1]^d \to \mathbf{R}$. Now, let $\lambda > 0$ be arbitrary. Consider the estimator $\widetilde{f}_n$ defined in Lemma 3.2, and the function h^*_λ constant on the cells of a random partition $\Pi_\lambda \sim \mathsf{MP}(\lambda)$, with optimal predictions on the leaves given by $h^*_\lambda(u) = \mathbb{E}[y|x \in C_{\mathbf{v}}]$ for $u \in C_{\mathbf{v}}$, for every leaf $\mathbf{v}$ of Π_λ. Since $R(\widetilde{f}_n) - R(f^*) = R(\widetilde{f}_n) - R(h^*_\lambda) + R(h^*_\lambda) - R(f^*)$, Equation (3.34) gives, after taking the expectation over the random sampling of the Mondrian process Π_λ,

$$\mathbb{E}[(\widetilde{f}_n(x) - f^*(x))^2] = \mathbb{E}[R(\widetilde{f}_n)] - \mathbb{E}[R(h^*_\lambda)] + \mathbb{E}[(h^*_\lambda(x) - f^*(x))^2]\,. \tag{3.35}$$

Let $D_\lambda(u)$ denote the diameter of the cell $C_\lambda(u)$ of $u \in [0,1]^d$ in the Mondrian partition Π_λ used to define h^*_λ. Assume that f^* is β-Holder with constant $L > 0$, namely $|f^*(u) - f^*(v)| \leqslant L|u - v|^\beta$ for any $u, v \in [0,1]^d$. Since $h^*_\lambda(u) = \mathbb{E}[f^*(x)|x \in C_\lambda(u)]$, we have $|h^*_\lambda(u) - f^*(u)| \leqslant LD_\lambda(u)^\beta$, so that

$$\mathbb{E}[(h^*_\lambda(x) - f^*(x))^2] \leqslant L^2\mathbb{E}[D_\lambda(x)^{2\beta}]\,. \tag{3.36}$$

Now, since $u \mapsto u^\beta$ is concave,

$$\mathbb{E}[D_\lambda(x)^{2\beta}] \leqslant \mathbb{E}[D_\lambda(x)^2]^\beta \leqslant \Big(\frac{4d}{\lambda^2}\Big)^\beta \tag{3.37}$$

where the last inequality comes from Corollary 2.1 in Chapter 2. Integrating with the distribution of x and using (3.36) gives $\mathbb{E}[(h^*_\lambda(x) - f^*(x))^2] \leqslant (4d)^\beta L^2/\lambda^{2\beta}$. In addition, Theorem 3.1 gives $\mathbb{E}[R(\widetilde{f}_n)] - \mathbb{E}[R(h^*_\lambda)] \leqslant 8B^2(1+\lambda)^d(\log n)/n$. Combining these inequalities with (3.35) leads to

$$\mathbb{E}[(\widetilde{f}_n(x) - f^*(x))^2] \leqslant \frac{(4d)^\beta L^2}{\lambda^{2\beta}} + \frac{8B^2(1+\lambda)^d \log n}{n}\,. \tag{3.38}$$

Note that the bound (3.38) holds for every value of $\lambda > 0$. In particular, for $\lambda \asymp (n/\log n)^{1/(d+2\beta)}$, it yields the $O\big((\log(n)/n)^{2\beta/(d+2\beta)}\big)$ bound on the estimation risk of Theorem 3.2. $\square$

Proof of Proposition 3.2. First, we reason conditionally on the realization of an infinite Mondrian partition Π, by considering the randomness with respect to the sampling of the feature points $x_1, \ldots, x_n, x$. For every depth $j \geqslant 0$, denote by N_j the number of points among $x_1, \ldots, x_n$ that belong to the cell of depth j of the unrestricted Mondrian partition containing x, and $\mathbf{v}_j \in \{0,1\}^j$ the corresponding node. In addition, for every $\mathbf{v} \in \{0,1\}^*$, denote

$p_{\mathbf{v}} = \mathbb{P}(x \in C_{\mathbf{v}0} | x \in C_{\mathbf{v}})$. In addition, for $j \geqslant 0$, since conditionally on $C_{\mathbf{v}_j}, N_j$, the points x and $\{x_i : x_i \in C_{\mathbf{v}_j}\}$ are distributed i.i.d. following the conditional distribution of x given $\{x \in C_{\mathbf{v}_j}\}$:

$$\begin{aligned}\mathbb{E}[N_{j+1}|C_{\mathbf{v}_j}, N_j, \Pi] &= \mathbb{P}(\mathbf{v}_{j+1} = \mathbf{v}_j 0|\mathbf{v}_j) \times N_j \mathbb{P}(x_1 \in C_{\mathbf{v}_j 0}|x_1 \in C_{\mathbf{v}_j}) \\ &\quad + \mathbb{P}(\mathbf{v}_{j+1} = \mathbf{v}_j 1|\mathbf{v}_j) \times N_j \mathbb{P}(x_1 \in C_{\mathbf{v}_j 1}|x_1 \in C_{\mathbf{v}_j}) \\ &= N_j \big(p_{\mathbf{v}_j}^2 + (1-p_{\mathbf{v}_j})^2\big) \\ &= N_j \big(1 - 2p_{\mathbf{v}_j}(1-p_{\mathbf{v}_j})\big) . \end{aligned} \tag{3.39}$$

Now, note that $p_{\mathbf{v}_j}(1-p_{\mathbf{v}_j})$ is determined by $C_{\mathbf{v}_j}$ and its split in Π, while N_j is determined by $C_{\mathbf{v}_j}$ and $x_1, \dots, x_n$. Now, let U_j be the ratio of the volume of $C_{\mathbf{v}_j 0}$ by that of $C_{\mathbf{v}_j}$; by construction of the Mondrian process, $U_j \sim \mathcal{U}([0,1])$ conditionally on $C_{\mathbf{v}_j}$. In addition, the assumption (3.15) implies (by integrating over the coordinate of the split, fixing the other coordinates) that $p_{\mathbf{v}_j} \geqslant M^{-1} U_j$, $1 - p_{\mathbf{v}_j} \geqslant M^{-1}(1-U_j)$. It follows that

$$p_{\mathbf{v}_j}(1-p_{\mathbf{v}_j}) \geqslant \frac{1}{2}\{p_{\mathbf{v}_j} \wedge (1-p_{\mathbf{v}_j})\} \geqslant \frac{1}{2M}\{U_j \wedge (1-U_j)\}$$

so that

$$\mathbb{E}[p_{\mathbf{v}_j}(1-p_{\mathbf{v}_j})|C_{\mathbf{v}_j}] \geqslant \frac{1}{2M}\mathbb{E}[U_j \wedge (1-U_j)|C_{\mathbf{v}_j}] = \frac{1}{4M} .$$

Using the fact that N_j and $p_{\mathbf{v}_j}$ are independent conditionally on $C_{\mathbf{v}_j}$, it follows from (3.39) that

$$\mathbb{E}[N_{j+1}|C_{\mathbf{v}_j}] = \mathbb{E}[N_j|C_{\mathbf{v}_j}]\big(1 - 2\mathbb{E}[p_{\mathbf{v}_j}(1-p_{\mathbf{v}_j})]\big) \leqslant \Big(1 - \frac{1}{2M}\Big)\mathbb{E}[N_j|C_{\mathbf{v}_j}].$$

By induction on $k \geqslant 0$, using the fact that by definition $N_0 = n$,

$$\mathbb{E}[N_k] \leqslant n\Big(1 - \frac{1}{2M}\Big)^k . \tag{3.40}$$

Now, note that if $N_k = 0$, then the depth $D_n^{\Pi}(x)$ of x in the Mondrian partition Π restricted to $x_1, \dots, x_n, x$ is at most k. Thus, inequality (3.40) implies:

$$\begin{aligned}\mathbb{E}[D_n^{\Pi}(x)] &= \sum_{k\geqslant 1} \mathbb{P}(D_n^{\Pi}(x) \geqslant k) \leqslant \sum_{k \geqslant 1} \mathbb{P}(N_k \geqslant 1) \\ &\leqslant \sum_{k\geqslant 1} \mathbb{E}[N_k] \wedge 1 \leqslant \sum_{k\geqslant 1} \left\{ n\Big(1 - \frac{1}{2M}\Big)^k \right\} \wedge 1 \end{aligned} \tag{3.41}$$

Now, let k_0 be the smallest $k \geqslant 1$ such that $n(1 - 1/(2M))^{k_0} \leqslant 1$. We have

$$k_0 = \left\lceil \frac{\log n}{\log\{(2M)/(2M-1)\}} \right\rceil,$$

so that $k_0 - 1 \leqslant \log(n)/\log\{(2M)/(2M-1)\}$. Hence, inequality (3.41) becomes:

$$\begin{aligned}\mathbb{E}[D_n^{\Pi}(x)] &\leqslant (k_0 - 1) + \sum_{k \geqslant 0} \underbrace{n\Big(1 - \frac{1}{2M}\Big)^{k_0}}_{\leqslant 1} \Big(1 - \frac{1}{2M}\Big)^k \\ &\leqslant \frac{\log n}{\log[(2M)/(2M-1)]} + 2M \end{aligned}$$

which establishes Proposition 3.2. □