Fully-Dynamic Approximate Decision Trees With Worst-Case Update Time Guarantees

Marco Bressan [1] **Mauro Sozio** [2]

Abstract

We study the problem of maintaining a decision tree in the fully-dynamic setting, where the dataset is updated by an adversarial sequence of insertions and deletions. We present the first algorithm with strong guarantees on both the quality of the tree and the worst-case update time (the maximum number of operations the algorithm performs between two dataset updates). For instance, we can maintain a tree where each node has Gini gain within β of the optimum, while guaranteeing an update time $O\big(d\,\beta^{-3}\log^4 n\big)$, where d is the number of features and n the maximum size of the dataset. This is optimal up to polylogarithmic factors, as any dynamic algorithm must have update time in $\Omega(d)$. Similar guarantees hold for the variance and information gain, for classification and regression, as well as for *boosted* trees. This shows that many popular decision trees such as ID3 or C4.5 can be efficiently made dynamic, answering an open question of Bressan, Damay, and Sozio (2023). We also show that, under the 3SUM conjecture or the Orthogonal Vectors Hypothesis, the update time must be polynomial in β^{-1}.

1. Introduction

Decision trees are among the most successful tools in machine learning and data mining (Shalev-Shwartz & Ben-David, 2014; Wu et al., 2008). In the standard *offline* model, decision trees are constructed by feeding a training set of labeled examples to a decision tree construction algorithm. The most popular such algorithms, such as ID3 (Quinlan, 1986) or C4.5 (Quinlan, 1993), work in this way. In many cases, however, one deals with training data that is subject to frequent updates over time. In those cases, any decision tree may quickly become obsolete and need to be replaced. This has led to the study of algorithms for maintaining decision trees in *dynamic* models, where the input is given as a stream of examples, and the algorithm can update the tree after each example is read from the stream. Many such algorithms have been developed, and have proved to deliver high performance with respect to both quality and running time (Domingos & Hulten, 2000; Hulten et al., 2001; Gama et al., 2003; Manapragada et al., 2018; Das et al., 2019; Sun et al., 2020; Haug et al., 2022; Jin & Agrawal, 2003; Rutkowski et al., 2013). All those algorithms, however, have two major limitations. The first one is that they are purely *incremental*: they assume the examples are added to the dataset, and that no example is ever removed. The second one is that they ensure trees of good quality only if the streamed examples are drawn independently from some distribution; for an arbitrary stream, they give no guarantee.

In this work we study the problem of maintaining decision trees in the *fully dynamic* model. This model assumes arbitrary streams of insertions and deletions, and is extensively studied in machine learning and related areas (Cohen-Addad et al., 2019; Henzinger & Kale, 2020; Lattanzi et al., 2020; Duetting et al., 2023). Let X be any domain (say, $X = \mathbb{R}^d$) and $Y = \{0, 1\}$. Any $x \in X$ is an *unlabeled example*, and any $(x, y) \in X \times Y$ is a *labeled example*. In the fully dynamic model, the algorithm starts with some initial multiset S_0 of labeled examples, which we call a *dataset*.[1] Then, at discrete time steps $i = 1, 2, \ldots$, an adversary updates S_{i-1} to S_i by inserting or deleting a labeled example (x_i, y_i). The goal of the algorithm is to maintain, at every time i, a decision tree T_i that is close to the offline tree that some reference algorithm (say, C4.5) would compute from S_i (we discuss this notion of "closeness" below). The computational performance of the algorithm is measured by the *update time* and the *query time*. The update time, t_u, is the maximum time spent by the algorithm to compute T_i from T_{i-1}. The query time, t_q, is the maximum time spent by the algorithm to compute the label $T_i(x)$ predicted by T_i on any given $x \in X$. The ideal goal is $t_u = O(d \operatorname{poly}\log n)$ and

[1]Department of Computer Science, University of Milan, Italy. [2]Institut Polytechnique de Paris, Telecom Paris. Correspondence to: Marco Bressan <marco.bressan@unimi.it>, Mauro Sozio <sozio@telecom-paris.fr>.

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

[1]We use "dataset" as a term for both "multiset" and "sequence", depending on the use we make of them.

$t_q = O(h)$, with $h = \max_i h(T_i)$ and $n = \max_i |S_i|$ being the maximum height of the tree and the maximum dataset size, respectively.[2]

Let us now discuss how to gauge the quality of the tree T_i maintained by the algorithm. A first attempt would be to require that T_i approximates well the labeling of the offline tree, in the sense that the labelings of the two trees disagree on just a small fraction of the examples in S_i. The crucial problem is that decision trees are inherently unstable: adding even a single example to the current dataset can lead to an offline tree completely different in terms of both topology and labeling (Li & Belford, 2002). As a consequence, the algorithm may need to recompute the whole tree from scratch after every couple of updates. This is a well-known issue, and there seems to be no easy workaround. We shall therefore take another route: instead of approximating the labeling given by the offline tree, we approximate the *decisions* taken by the algorithm during its construction.

Let us introduce some notation. A decision tree is a full binary tree T. Each internal node v of T holds a *split function* $\sigma_v : X \to \{0, 1\}$ which specifies to which child of v (left or right) each $x \in X$ should go. In this presentation we assume splits in the form $\mathbb{1}_{x_j<t}$, that is, "is the j-th feature of x smaller than t?", but our results allow for more general splits. Every leaf v instead holds a label $\lambda_v \in Y$. The quality of the split σ_v is measured by some function, called *gain*, that measures how much a dataset "improves" by splitting it according to σ_v. For ease of presentation we will use the Gini gain but, again, our results generalize to other gains (the information gain and the variance gain). Let p_S be the fraction of examples in S with label 1. The (Gini) *impurity* of S is $g(S) = 2p_S(1-p_S)$. Thus $g(S) = 0$ when all labels are identical, and $g(S) = \frac{1}{2}$ when half are 0 and half are 1. For a split σ let $(S_0, S_1) = \sigma(S)$ where S_z is the subset of S on which σ evaluates to z. The Gini gain of σ over S is:

$$G(S, \sigma) = g(S) - \left(\frac{|S_0|}{|S|} g(S_0) + \frac{|S_1|}{|S|} g(S_1) \right)$$

The typical decision tree algorithm constructs the tree in a greedy fashion, as follows. First, it starts with T consisting of one leaf v whose label λ_v is the majority label of S. For every node v in T let $S(T, v)$ be the dataset consisting of all examples of S that, following the splits of T, reach v. The algorithm takes a leaf v of T, and finds a split σ that maximizes $G(S(T, v), \sigma)$. If $G(S(T, v), \sigma) \geq \alpha$ for some fixed threshold $\alpha > 0$, then the leaf v is *split*: it is converted to an internal node with split function $\sigma_v = \sigma$, and with two child leaves whose labels are the majority labels of respectively $S(T, v)_0$ and $S(T, v)_1$. If instead $G(S(T, v), \sigma) < \alpha$, then the algorithm moves on to examine the next leaf. This operation is repeated until no leaf is split. We call this a *max-gain construction rule with threshold* α. It is the construction rule used by algorithms such as C4.5 and ID3 (possibly using other types of gain).

All the works listed above aim to maintain decision trees that, at every node, approximate the gain obtained by the max-gain construction rule. In defining a suitable approximation one must be careful, though. If, to decide whether a node is internal, one uses a hard gain threshold, then updating *one single example* in the dataset can make the gain cross the threshold, and thus the node flip from internal to leaf (or viceversa), forcing the algorithm to repeatedly reconstruct the tree. Thus, one should allow some slack. The following definition from (Bressan et al., 2023) captures the idea. For every internal node $v \in V(T)$ let again σ_v be the split associated to v, and let σ_v^* be the best split, that is, the one that maximizes the gain $G(S(T, v), \sigma_v^*)$, and for compactness let $G^*(v) = G(S(T, v), \sigma_v^*)$.

Definition 1.1 ($\boldsymbol{\epsilon}$-feasibility)**.** Let $\boldsymbol{\epsilon} = (\alpha, \beta)$ where $0 < \beta < \alpha \leq 1$. A decision tree T is $\boldsymbol{\epsilon}$-feasible w.r.t. a dataset S if for all $v \in V(T)$:

1. if $G^*(v) \geq \alpha$ then v is internal and its split function σ_v satisfies $G(S(T, v), \sigma_v) \geq G^*(v) - \beta$.
2. if $G^*(v) = 0$ then v is a leaf.

The crucial point of Definition 1.1 is that it requires a node v to be internal, and with gain close to $G^*(v)$, only when $G^*(v) \geq \alpha$, and it requires v to be a leaf only when $G^*(v) = 0$. When instead $G^*(v)$ is between 0 and α, then v is allowed to be a leaf, or an internal node but with gain far from optimal. As a consequence, as long as $0 < G^*(v) < \alpha$ the algorithm may leave v untouched, even if $S(T, v)$ has been updated and the gain at v and/or the choice of making v internal or leaf are not near-optimal anymore. Therefore, if $G^*(v)$ changes slowly with the number of updates, then the algorithm may have the time to rebuild the subtree rooted at v before $G^*(v)$ falls outside $(0, \alpha)$ and Definition 1.1 constrains v again. Thus, $\boldsymbol{\epsilon}$-feasibility seems a good target for dynamic decision tree algorithms. Indeed, all works listed above adopt more or less explicitly some variant of $\boldsymbol{\epsilon}$-feasibility, possibly with some additional constraints (such as that v must be a leaf if $|S(T, v)|$ is small). The main goal of our work is to design an efficient algorithm for maintaining an $\boldsymbol{\epsilon}$-feasible decision tree in the fully dynamic model.

As mentioned above, all existing algorithms are only incremental (they support insertions, but not deletions) and give guarantees only in probability, assuming that the examples are i.i.d. from some distribution. The only exception is a recent work by the authors, Bressan et al. (2023). That work introduces FUDYADT, a deterministic algorithm that maintains an $\boldsymbol{\epsilon}$-feasible decision tree in the fully-dynamic model in update time $O\big(\beta^{-2} d \log^3 n\big)$. However, this holds

[2] Note that t_q must be constrained, for otherwise a lazy algorithm could achieve $t_u = O(\log n)$ by updating S_i after each update, but computing T_i only at query time.

only in an *amortized* sense. More precisely, after the i-th update the cumulative running time of FUDYADT can be $\Omega(d|S_i|)$, and that time could be entirely spent over the last update. This is not an artifact of the analysis: by construction, when the $\epsilon|S|$-th update over a dataset S is performed, FUDYADT recomputes T entirely, and therefore it truly spends time $\Omega(dn)$ over a single update. Therefore, it is still unknown whether the amortized bounds of FUDYADT can be de-amortized and made worst-case. The main result of our work is to show that this is possible.

1.1. Contributions

A framework for approximate greedy decision trees (Section 3). We start by devising a general notion of approximation for decision trees. Let us pin down some notation first. A *construction rule* is a function ρ that maps every dataset S to a split σ or a label λ. Every construction rule ρ defines an algorithm GREEDY_ρ that proceeds top-down as described above. Given two datasets S, S', their *edit distance* is $\triangle(S, S') = |S \setminus S'| + |S' \setminus S|$, and their *relative edit distance* is $\triangle^*(S, S') = \frac{\triangle(S,S')}{\max(|S|,|S'|)}$. Recall that $S(T, v)$ is the dataset consisting of all examples of S that, following the splits of T, reach v.

Definition 1.2 (ϵ-approximation)**.** Let S be a dataset and ρ a construction rule. A decision tree T is *ϵ-approximate w.r.t.* (S, ρ) if, for every $v \in V(T)$, there exists a dataset S_v such that $\triangle^*(S(T, v), S_v) \leq \epsilon$, and that $\rho(S_v) = \sigma_v$ if v is internal and $\rho(S_v) = \lambda_v$ if v is a leaf.

Intuitively, Definition 1.2 says that T is ϵ-approximate if it was constructed by taking, at every node v, the same decision that ρ would take on a dataset S_v close to $S(T, v)$. The notion of ϵ-approximation turns out to be the right one for us to study. First, we show that it implies $\boldsymbol{\epsilon}$-feasibility for several common construction rules (including the thresholds $\mathbb{1}_{x_j<t}$) and for all gains that behave smoothly under perturbations of the datasets (including Gini gain, information gain, and variance gain). Second, we show that it implies boosting properties under standard weak-learning assumptions, as we explain below. Third, we give an efficient fully-dynamic algorithm that maintains an ϵ-approximate decision tree, as we describe next.

A fully-dynamic algorithm for ϵ-approximate trees (Section 4). Let again ρ be a construction rule. We consider the following problem, that we name *dynamic approximate decision tree problem*. The input is again the sequence of update requests $u_1, u_2, \ldots$. The goal is to maintain, for every $i = 1, 2, \ldots$, a tree T_i that is ϵ-approximate w.r.t. (S_i, ρ). We present a deterministic algorithm for this problem, FUDY-WC, that guarantees query time $t_q = O(h(T_i))$ and worst-case update time:

$$t_u = O\big(\epsilon^{-3} d \log^4 n\big)$$

FUDY-WC is the key technical contribution of our work. As we show below, for an appropriate choice of ϵ the tree maintained by FUDY-WC is in fact $\boldsymbol{\epsilon}$-feasible, where $\boldsymbol{\epsilon} = (\alpha, \beta)$. Thus, FUDY-WC is also fully-dynamic algorithm for our original problem of maintaining $\boldsymbol{\epsilon}$-feasible trees. We also show that FUDY-WC is nearly optimal in terms of its space usage, which is in $O(dn \operatorname{poly}(\log n/\epsilon))$.

We remark that FUDY-WC is substantially different from the algorithm of (Bressan et al., 2023), FUDYADT, which attains an *amortized* update time of $O\big(\beta^{-2} d \log^3 n\big)$. In particular, FUDY-WC does not simply take the total work done by FUDYADT and "spread" it over the sequence of updates (actually, we show that this approach does not work). Rather, FUDY-WC is based on solving a certain relaxation of the original problem; this is a novel ingredient, not used by FUDYADT. It should also be noted that FUDYADT and its $O(d \operatorname{poly} \log n)$ amortized update time do not constitute evidence that a $O(d \operatorname{poly} \log n)$ update time could be obtained in the worst case. Indeed, under widely accepted conjectures, several dynamic problems have update time that is poly-logarithmic in the amortized case but polynomial in the worst case (Hanauer et al., 2022).

Applications: popular decision trees made dynamic (Section 5). By combining our results above, we obtain the first fully-dynamic algorithm for maintaining $\boldsymbol{\epsilon}$-feasible decision trees with strong worst-case performance guarantees. More precisely, if $\boldsymbol{\epsilon} = (\alpha, \beta)$, then for $\epsilon = \frac{\beta}{100}$ FUDY-WC maintains an $\boldsymbol{\epsilon}$-feasible decision tree in query time $t_q = O(\beta^{-1} \log n)$ and worst-case update time:

$$t_u = O(\beta^{-3} d \log^4 n)$$

This is just a factor $O\big(\beta^{-1} \log n\big)$ away from the update time of (Bressan et al., 2023); but, again, their bound is amortized, while ours is worst-case. We prove similar bounds for multi-class decision trees, where $Y = \{0, \ldots, k\}$, and/or for G being the information gain; as well as for regression trees, where $Y = \mathbb{R}$ and G is the variance gain. Table 1 gives a summary of these bounds. In a nutshell, this shows that popular decision trees such as ID3 and C4.5 can be efficiently made fully dynamic.

task	update time	query time
classification (Gini gain)	$O\Big(\frac{d \log^4 n}{\epsilon^3}\Big)$	$O\big(\frac{\log n}{\epsilon}\big)$
classification (information gain)	$O\Big(\frac{d \log^7 n}{\epsilon^3}\Big)$	$O\Big(\frac{\log^2 n}{\epsilon}\Big)$
regression (variance gain)	$O\Big(\frac{dc^6 \log^4 n}{\epsilon^3}\Big)$	$O\big(\frac{\log n}{\epsilon}\big)$

Table 1. Update and query times for maintaining an ϵ-approximate tree, as well as for maintaining a $\boldsymbol{\epsilon}$-approximate tree if $\epsilon = \beta$. Here $(\alpha, \beta) = \boldsymbol{\epsilon}$, and c is a bound on the absolute value of the labels.

For constant $\epsilon > 0$, our bounds are optimal up to $\operatorname{poly} \log n$

factors, at least when $X = \{0,1\}^d$ and when G is the Gini gain. Indeed, in that case our update time is $O(d \operatorname{poly} \log n)$, and our query time is $O(h) = O(d)$ (because, with $X = \{0,1\}^d$, the tree never splits twice on the same feature). And, by a lower bound of (Bressan et al., 2023), any algorithm has update/query time $\Omega(d)$. The space used is almost optimal, too: we use space $O(dn \operatorname{poly}(\log n/\epsilon))$, and again (Bressan et al., 2023) give a lower bound of $\Omega\big(d\, n/\log n\big)$. Thus, we achieve simultaneously a near-optimal worst-case update and query time and a near-optimal space usage.

Our results also yield efficient fully-dynamic *boosted trees*. We start from the *weak learning assumption*; it essentially says that, for any distribution over the input data, one can find a split that correlates with the labels. It is well-known that, under this assumption, for any $\delta > 0$ one can construct a tree of height $O(\log \frac{1}{\delta})$ that has loss at most δ over S (Kearns & Mansour, 1999; Takimoto & Maruoka, 1998). This is called *tree boosting*. Using our results we can show that, for $\epsilon = \Theta(\delta)$, a decision tree that is ϵ-approximate w.r.t. the construction rule used by boosting has loss at most δ and height $O(\log \frac{1}{\delta})$, too. Thus, FUDY-WC can be used to maintain dynamically a boosted tree that, at any time, is correct on all but a fraction δ of the points in the dataset. Loosely speaking, this means that boosted trees can be made fully dynamic for free.

Conditional lower bounds based on 3SUM and OVH (Section 6). We complement our upper bounds with conditional lower bounds based on two widely accepted conjectures from computational complexity theory: the 3SUM conjecture (3SUM) and the Orthogonal Vector Hypothesis (OVH). The 3SUM posits that $n^{2-o(1)}$ time is needed to decide if a set of n integers contains three distinct elements whose sum is 0. The OVH posits that $n^{2-o(1)}$ time is needed to decide if a set of n vectors contain two orthogonal vectors. We reduce 3SUM and OVH to two specific variants of our dynamic decision tree problem. In the first variant, we allow examples to be weighted by positive integers. In the second variant, we consider construction rules that use the variance gain (instead of the Gini gain). Our lower bounds show that, unless 3SUM or OVH fail, any dynamic algorithm for maintaining exactly the offline tree has an update or query time at least polynomial in n. Since any $\frac{1}{n+1}$-approximate tree coincides with the offline tree, this implies that maintaining an ϵ-approximate tree requires an update time or query time at least polynomial in ϵ^{-1}. This suggests that our algorithms cannot be improved significantly, and in particular that a polylogarithmic dependence on ϵ^{-1} is not possible.

Section 2 and 3 introduce the necessary notation and definitions. Our main algorithm is presented in Section 4, while Section 5 discusses some applications. Our lower bounds are presented in Section 6. Finally, Section 8 points out interesting directions for future work.

2. Preliminaries

All missing proofs and claims can be found in the Appendix. We assume $X = X_1 \times \ldots \times X_d$ where each X_i is arbitrary (e.g., $\{0,1\}$ or $\mathbb{R}$). We denote a labeled example by $s = (x, y) \in X \times Y$. A dataset is a sequence, or multiset, $S = (s_1, \ldots, s_n) \in (X \times Y)^*$. We always assume $|S| \geq 1$ unless otherwise specified. An *update request* is a pair $u = (s, o)$ where $o \in \{\text{INS}, \text{DEL}\}$. For a dataset S and a sequence of update requests U we denote by $S + U$ the dataset obtained by applying U to S in the obvious way. In particular, the dataset *defined by* U is the dataset $\emptyset + U$. If $U = ((s, \text{INS}))$ then we may just write $S + s$.

Split function and decision trees. Throughout the paper we denote by $\mathcal{S}$ the desired family of split functions (e.g., feature thresholds). For all $x \in X$ and all split functions σ we assume one can compute $\sigma(x)$ in time $O(1)$, so that for a dataset S one can compute $(S_0, S_1) = \sigma(S)$ in time $O(|S|)$. Let T be a decision tree. For $x \in X$ denote by $P(T, x)$ the path followed by x in T. We say x reaches $v \in V(T)$ if $v \in P(T, x)$, and we define $T(x) = \lambda_v(x)$ where v is the leaf in $P(T, x)$. Clearly one can compute $P(T, x)$ and $T(x)$ in time linear in $h(T)$. All these definitions hold for both labeled and unlabeled examples, as well as for update requests, in the obvious way. For $v \in V(T)$ we denote by T_v the subtree of T rooted at v.

All decision trees in this work are maintained explicitly[3]. The trees are pointer-based: every $v \in V(T)$ is represented by a data structure holding pointers to v's parent and children (if any). We denote by D a generic associative array data structure that supports insertion, lookup, and deletion in time $O(\log |D|)$, as well as enumeration in time $O(|D|)$, where $|D|$ is the total number of entries in D; this can be fulfilled by a self-balancing search tree. Each node v of T points to such an array $D(T, v)$ that stores some set of labeled examples by mapping each example to the number of its occurrences. We may keep additional counters or structures at v; for example we may keep $|D(T, v)|$.

Construction rules and greedy algorithms. A construction rule is a map $\rho : (X \times Y)^* \to \mathcal{S} \cup Y$. We say ρ is γ-balanced if it yields γ-balanced splits, i.e., if $\min(|S_0|, |S_1|) \geq \gamma |S|$ when $\rho(S) = \sigma$. If ρ is γ-balanced then $h(T) = O(\gamma^{-1} \log n)$ where $n = |S|$ is the size of the dataset used to construct T. We assume that, if $\rho(S)$ returns a label λ, then λ is a label of maximum frequency in S (or, for regression trees, the average of the labels of S). We denote by $f_\rho : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ an upper bound on the cost of computing $\rho(S)$ as a function of $|S|$. We assume that f_ρ is twice differentiable and $f'_\rho, f''_\rho \geq 0$; this implies f_ρ is superadditive and $f_\rho(n) = \Omega(n)$, so in time $f_\rho(|S|)$

[3] In principle one could maintain the labeling function of the tree without storing the tree's structure.

one can compute both $\rho(S)$ and $\sigma(S)$. Every construction rule ρ immediately defines a greedy algorithm GREEDY_ρ, see Algorithm 2. (For compactness Algorithm 2 is in recursive form, but it is straightforward to make it iterative). GREEDY_ρ is the algorithm we use to reconstruct subtrees dynamically. We assume GREEDY_ρ stores $S(T, v)$ in the dictionary $D(T, v)$, for all $v \in V(T)$.

Algorithm 1 GREEDY_ρ

Input: dataset S
$T =$ a tree on one *unmarked* leaf v
store S in a dictionary $D(T, v)$
compute $\rho(S)$
if $\rho(S)$ is a split function σ **then**
  let $\sigma_v = \sigma$ and let $(S_0, S_1) = \sigma(S)$
  compute $T_0 = \text{GREEDY}_\rho(S_0)$
  compute $T_1 = \text{GREEDY}_\rho(S_1)$
  attach T_0 and T_1 as left and right child of v
else
  let $\lambda_v = \rho(S)$
return T

Gain functions, max-gain rules, and threshold rules. An *impurity* is a function $g : (X \times Y)^* \to \mathbb{R}_{\geq 0}$. Every impurity g gives the *conditional g-gain* G that for every S, σ yields:

$$G(S, \sigma) = g(S) - \left(\frac{|S_0|}{|S|} g(S_0) + \frac{|S_1|}{|S|} g(S_1) \right) \quad (1)$$

where $(S_0, S_1) = \sigma(S)$. In this work we consider only conditional gains. Again, by default we assume g to be the Gini impurity, whose conditional g-gain is the Gini gain; but our results extend to the information gain and the variance (defined in Appendix B). A construction rule ρ is a max-G rule if it always chooses splits of maximum gain, that is, if $\rho(S) = \sigma_S \in \mathcal{S}$ implies:

$$\sigma_S \in \arg\max_{\sigma \in \mathcal{S}} G(S, \sigma) \quad (2)$$

We say ρ has *threshold* $\alpha \geq 0$ if $\rho(S)$ produces a split iff:

$$\max_{\sigma \in \mathcal{S}} G(S, \sigma) \geq \alpha \quad (3)$$

For the gains above, a max-G construction rule with threshold α is γ-balanced for some γ close to α; see Appendix C.

Note. For the sake of the presentation, unless otherwise specified, we assume G is the Gini gain, $\mathcal{S}$ is the family of all feature-threshold splits σ, and ρ is a max-gain rule with threshold α for some fixed $\alpha > 0$. As said above, this implies that $T = \text{GREEDY}_\rho(S)$ has height $O(\log |S|)$. This implies that $\text{GREEDY}_\rho(S)$ runs in time $O(d|S| \log^2 |S|)$: for each of the $O(\log |S|)$ levels it sorts a total of $O(|S|)$ elements, each one of size $O(d)$.

3. Approximate and Feasible Decision Trees

This section starts the technical overview of our algorithms and results. We start by defining the two problems central to our work. Let $u^1, u^2, \ldots$ be a sequence of update requests, and for $i \geq 1$ let S^i be the dataset defined by $u^1, \ldots, u^i$.

Definition 3.1. The *Dynamic Feasible Decision Tree Problem* asks, given $\boldsymbol{\epsilon}$, to maintain for all $i \geq 1$ a tree T^i that is $\boldsymbol{\epsilon}$-feasible with respect to S^i.

Definition 3.2. The *Dynamic Approximate Decision Tree Problem* asks, given $\epsilon \in (0, 1)$ and a construction rule ρ, to maintain for all $i \geq 1$ a tree T^i that is ϵ-approximate with respect to (S^i, ρ).

As a first step, we show how $\boldsymbol{\epsilon}$-feasibility (Definition 1.1) is implied by ϵ-approximation (Definition 1.2). As a result, we can focus on the dynamic approximate decision tree problem; the feasible version comes for free.

Theorem 3.3. *Let $\boldsymbol{\epsilon} = (\alpha, \beta)$, let $\epsilon = \frac{\beta}{100}$, and let ρ be a max-gain construction rule with threshold $\frac{\alpha}{2}$. Then, for all datasets S, every decision tree that is ϵ-approximate w.r.t. (S, ρ) is also $\boldsymbol{\epsilon}$-feasible w.r.t. S.*

The proof of Theorem 3.3 is given in Appendix D.1. The key observation is that $G(S, \sigma)$ is Lipschitz w.r.t. the relative edit distance: that is, $|G(S, \sigma) - G(S', \sigma)| = O(\triangle^*(S, S'))$. This is proven in Appendix C, and was already exploited by (Bressan et al., 2023). Because of this, a tree that is ϵ-approximate w.r.t. (S, ρ) behaves essentially as ρ — it makes almost the same decision at every node, and it has the same gains except for an additive term of $O(\beta)$. In particular, if we choose ρ to have threshold $\frac{\alpha}{2}$ instead of α, we leave enough slack for the ϵ-approximate tree to behave like a tree produced by a construction rule with threshold α, and with the same gains save for an additive β. In other words, the tree is $\boldsymbol{\epsilon}$-feasible.

Moreover, from G being Lipschitz it follows that, if a tree T has gain at least γ at each internal node, then the tree has logarithmic height. Since for a $\boldsymbol{\epsilon}$-feasible tree this holds with $\gamma = \alpha - \beta > 0$, then $\boldsymbol{\epsilon}$-feasible trees have logarithmic depth, too.

Lemma 3.4. *Let $\boldsymbol{\epsilon} = (\alpha, \beta)$. If T is $\boldsymbol{\epsilon}$-feasible w.r.t. S then $h(T) = O\left(\frac{\log |S|}{\alpha - \beta}\right)$.*

Lemma 3.4 follows immediately from the balancedness results of Appendix C.4 (that, again, include bounds also for the information gain and variance gain). Note that, together, Theorem 3.3 and Lemma 3.4 imply that ϵ-approximate trees have logarithmic height, too. These results are used implicitly in the next sections to bound the query time of our algorithms, which are linear in the height of the tree.

4. Dynamic Approximate Decision Trees

We describe FUDY-WC, our algorithm for the dynamic approximate decision tree problem. Our main result is:

Theorem 4.1. *Let ρ be a max-gain construction rule and let $\epsilon > 0$.* FUDY-WC *maintains a decision tree ϵ-approximate w.r.t. ρ with worst-case query time $t_q = O(\epsilon^{-1} \log n)$ and worst-case update time $t_u = O\left(\epsilon^{-3} d \log^4 n\right)$.*

The rest of the section discusses FUDY-WC and gives a detailed walkthrough of the arguments used to prove Theorem 4.1 (for a full proof see Appendix E). Before delving into that, let us remark that we also show that FUDY-WC uses space almost linear in the total size of the dataset. To measure the space usage, we equip the algorithm with a primitive MALLOC, which takes a positive integer k and in time $O(k)$ returns a pointer to a block of k contiguous words; and a primitive FREE, which takes a pointer returned by MALLOC and returns in time $O(1)$. At any point, the *total space* used by the algorithm is the sum of the sizes of the blocks MALLOC-ed so far, minus the sizes of those that have been FREE-ed. In this model we prove (Appendix E.6):

Theorem 4.2. *The total space used by* FUDY-WC *after the first i update requests is in $O\left(\epsilon^{-2}\, d\, |S_i| \log^2 n + t_u\right)$.*

Let us turn to FUDY-WC. In order to build the intuition we start from FUDYADT, the dynamic amortized algorithm of (Bressan et al., 2023). Let S be a dataset, and suppose we start with the decision tree T that is the output of GREEDY_ρ on S. Obviously T is ϵ-approximate w.r.t. (S, ρ). As said above, the Gini gain is Lipschitz, i.e., $|G(S, \sigma) - G(S', \sigma)| = O(\triangle^*(S, S'))$ for any two datasets S'. Thus, if we perform $\epsilon|S|$ updates on S, the Gini gain of any split function over S changes by $O(\epsilon)$. Therefore, the split σ_v at the root v of T can violate ϵ-approximation only after $\Omega(\epsilon|S|)$ updates to S. This applies in fact to every subtree T_v: for σ_v to violate ϵ-approximation, v must be reached by $\Omega(\epsilon|S(T, v)|)$ updates since T_v was constructed using $S(T, v)$. Now, if $n_v = |S(T, v)|$, then rebuilding T_v by running GREEDY_ρ on S_v takes time $O\left(d\, n_v \log^2 n_v\right)$. Then each update pays for $O(\epsilon^{-1} d \log^2 n_v)$ operations. This is the core argument giving the amortized time bound of (Bressan et al., 2023).

Let us try to de-amortize the approach above, and make the bound hold in the worst case. By Definition 1.2, for any given $v \in V(T)$, we could recompute the split of v by ignoring the last ϵn_v update requests that reach v. Thus, in principle, we could use a charging argument to make those ϵn_v updates pay for the reconstruction of T_v. The challenging part is to implement this high-level idea correctly. Consider indeed the following "naive" approach. First, wait until the first $\frac{\epsilon\, n_v}{2}$ update requests reach v. At that point we start constructing a new subtree by running GREEDY_ρ on the updated dataset S'_v at v, and we spread the execution of GREEDY_ρ over the next $\frac{\epsilon\, n_v}{2}$ update requests that reach v. In this way, between any two of those update requests we perform $O(\epsilon^{-1} d \log^2 n_v)$ operations. We do this for every vertex of the tree; every update request will pay for the at most $h(T)$ vertices it reaches, and thus we will get a total update time of $O(h\epsilon^{-1} d \log^2 n_v)$. Unfortunately, this approach does not yield an approximate tree: the new subtree T'_v computed at v is ϵ-approximate w.r.t. $S'(T, v)$; and, by the time T'_v is ready, $S'(T, v)$ is not anymore the dataset at v, because of the additional $\frac{\epsilon\, n_v}{2}$ update requests arrived in the meantime. Therefore we cannot replace T_v with T'_v, because there is no guarantee that T'_v is ϵ-approximate w.r.t. to the current dataset. We shall refer to such a problem as the *ready-too-late* problem. This is not easily fixable: T'_v could be made not approximate by the last request that arrives, in which case we are out of time for rebuilding it.

We need another strategy. The next section shows how to obtain an algorithm by reducing the original dynamic problem to a relaxed version.

4.1. The *Delayed* Approximate Decision Tree Problem

Consider the following problem.

Definition 4.3 (Delayed Approximate Decision Tree)**.** The input is a tuple $(\rho, \epsilon, S, n, U)$ where ρ is a construction rule, $\epsilon \in (0, 1)$, S is a dataset with $|S| = n$, U is a sequence of ϵn updates $u^1, \ldots, u^{\epsilon n}$. The output is a decision tree T that is ϵ-approximate with respect to (S', ρ), and such that $D(T, v)$ stores $S'(T, v)$ for each $v \in V(T)$, where $S' = S + U$.

Note how this is a relaxed version of the original problem: the algorithm only needs to output a single tree that is ϵ-approximate w.r.t. the final dataset S'. We will now show how to reduce the approximate decision tree problem to the delayed variant above. Consider a generic algorithm DELAY-APX for the delayed approximate decision tree problem. Suppose that both S and U are accessible through iterators, so that their elements can be read sequentially in time $O(d)$ per element. Suppose also that DELAY-APX starts by reading S, followed by the elements of U. Define the *fetch time* of DELAY-APX as the maximum time spent by DELAY-APX between any two consecutive elements fetched from U. We claim that, if DELAY-APX has fetch time τ, then we can achieve update time $O(h\,\tau)$ for the original problem, where h is the height of the maintained tree.

The reduction that turns the fetch time τ into an update time $O(h\,\tau)$ is performed by our algorithm FUDY-WC (Algorithm 2), which maintains a decision tree T as follows. First, for every $v \in V(T)$ FUDY-WC keeps an associative array $D(T, v)$ storing the dataset S_v from which the current subtree T_v was built by GREEDY_ρ. Second, for every $v \in V(T)$ FUDY-WC simulates an instance $\mathcal{I}(v)$ of DELAY-APX with input $\left(\rho, \frac{\epsilon}{2}, |S_v|, S_v, U_v\right)$. Now, the execution of

Algorithm 2 FUDY-WC

1: **Input:** construction rule ρ, approximation $\epsilon \in (0,1)$, and a sequence of update requests $\{(s_i, o_i)\}_{i \geq 1}$
2: $T =$ a tree on one leaf v with $\lambda_v = 1$ and $\mathcal{I}(v) =$ NIL
3: **for** each $i = 1, 2, \ldots$ **do**
4: **for** each v in $P(T, s_i)$ **do**
5: **if** $\mathcal{I}(v) =$ NIL **then**
6: initialize an empty iterator $U(v)$
7: start an instance $\mathcal{I}(v)$ of DELAY-APX$\big(\rho, \frac{\epsilon}{2}, |D(T,v)|, D(T,v), U(v)\big)$
8: append (s_i, o_i) to $U(v)$
9: **run** $\mathcal{I}(v)$ until it fetches an element from $U(v)$
10: **if** $\mathcal{I}(v)$ returns a pointer to a tree T'_v **then**
11: set $\mathcal{I}(v) =$ NIL and replace T_v with T'_v
12: **break** the loop of line 4

$\mathcal{I}(v)$ is allowed to progress only when an update request (s, o) from the input reaches v. When that happens, FUDY-WC appends (s, o) to U_v, and makes $\mathcal{I}(v)$ progress until it fetches that update. Thus, while $\mathcal{I}(v)$ sees U_v as a sequence of ϵn elements, FUDY-WC is actually creating that sequence on the fly. In this way, as soon as $\frac{\epsilon}{2}|S_v|$ updates reach v, by construction $\mathcal{I}(v)$ returns a new tree T'_v that is $\frac{\epsilon}{2}$-approximate w.r.t. $S'_v := S_v + U_v$, with S'_v being precisely the *current* dataset at v. Thus, all the nodes of T'_v satisfy the definition of $\frac{\epsilon}{2}$-approximation. Now FUDY-WC can replace T_v with T'_v, which boils down to just updating the pointer at v's parent (and free-ing the memory used by T_v). Thereafter, T'_v remains ϵ-approximate w.r.t. S'_v until the next $\frac{\epsilon}{2}$-approximation arrive; but, by that time, a new replacement for T'_v will have been built.

In this way the tree T maintained by FUDY-WC is always ϵ-approximate. Moreover, by our assumption on the fetch time of DELAY-APX, every update request makes $\mathcal{I}(v)$ perform at most τ operations. Therefore, as every update request entails running at most $h(T)$ instances of DELAY-APX, the update time of FUDY-WC is at most $O(\tau \cdot h(T))$. Formally, we prove (Appendix E):

Lemma 4.4. *Let* DELAY-APX *be an algorithm for the delayed approximate decision tree problem with fetch time τ. Then the tree T^i maintained by* FUDY-WC *is ϵ-approximate w.r.t. (S^i, ρ) for all $i \geq 0$. Moreover,* FUDY-WC *has update time $O(\tau \cdot h)$ where $h = \max_{i \geq 1} h(T^i)$.*

Next, we show an efficient algorithm for DELAY-APX. Together with Lemma 4.4, this will yield Theorem 4.1.

4.2. An Algorithm for the Delayed Problem

Theorem 4.5. *There is an algorithm for the delayed approximate decision tree problem,* DELAY-APX *(Algorithm 3), with fetch time $O\big(\epsilon^{-2} d \log^3 n\big)$.*

Theorem 4.5 together with Lemma 4.4 yields Theorem 4.1. The complete proof is nontrivial and requires keeping track carefully of several invariants. It is given in Appendix E.3, in a more general form that allows for general construction rules. The rest of the section gives a sketch.

Simplifying a little, DELAY-APX works as follows. First, we construct T from S, ignoring U. Now T may not be ϵ-approximate w.r.t. (S', ρ), but we show that its root does satisfy the ϵ-approximation constraint w.r.t. (S', ρ). This means that we do not need to recompute the split at the root anymore. Then, for $i = 1, 2, \ldots$, we feed T the next $|U|/2^i$ requests from U and we reconstruct any subtree whose root is not ϵ-approximate w.r.t. S plus all the updates seen so far. We can show that this reconstruction involves subtrees that hold, overall, roughly $|U|/\epsilon 2^i$ examples; thus every request "pays" for the work done on $\frac{1}{\epsilon}$ of those examples. Note that during the reconstruction T may be in an inconsistent state, that is, not ϵ-approximate w.r.t. any subsequence of S'. But eventually the process catches up, and T becomes ϵ-approximate w.r.t. the final sequence S'. This is where the "delayed" nature of the problem helps.

Algorithm 3 DELAY-APX

1: **Input:** construction rule ρ, $\epsilon \in (0,1)$, $n \in \mathbb{N}$, set S of examples, sequence U of $\epsilon|S|$ update requests
2: let $\ell = \log_2(\epsilon n) + 1$ and let τ be as in Theorem E.3
3: compute $T = \text{GREEDY}_\rho(S)$ using at most τ operations for each request fetched from U_0^1
4: **for** $i = 1, \ldots, \ell$: using at most τ operations for every request fetched from U_i^{i+1}, **do**
5: initialize a set $R_i = \emptyset$
6: **for** each $(s, o) \in U_{i-1}^i$ **do**
7: compute the path $P(T, s)$ of s in T
8: **for** each vertex $v \in P(T, s)$ **do**
9: update $D(T, v)$ with (s, o)
10: increment $\Delta_i(v)$
11: **if** $\Delta_i(v) \geq \frac{\epsilon}{4 \log_2 n} n_{i-1}(v)$ **then**
12: add v to R_i
13: $R_i^* = \{v \in R_i : \nexists$ no proper ancestor of $v \in R_i\}$
14: **for** every $v \in R_i^*$ **do**
15: $T_v = \text{GREEDY}_\rho(D(T, v))$
16: **return** a pointer to T

Let us describe DELAY-APX in more detail. Without loss of generality we may assume $\epsilon n = 2^{\ell - 1}$ for some $\ell \in \mathbb{N}$; otherwise we can just replace ϵ with an appropriate $\epsilon' \in (\epsilon/2, \epsilon)$. Define the following times:

$$t_i = \begin{cases} \epsilon n (1 - 2^{-i}) & i = 0, \ldots, \ell - 1 \\ \epsilon n & i = \ell \end{cases} \tag{4}$$

For all $j, i \geq 0$ let $U_j^i = u^{t_j + 1}, \ldots, u^{t_i}$, and to simplify the notation let $U_j = U_j^\ell$ and $U^i = U_0^i$. We also let $U_\ell^{\ell+1} =$

$U^\ell_{\ell-1}$. Therefore, the subsequence U^i contains all but the last $2^{-i}|U|$ elements of U, for all $i \neq \ell$, while $U^\ell = U$. Finally, for all $i \geq 0$ let $S^i = S + U^i$. Thus, S^i is the dataset obtained by applying to S all but the last $2^{-i}|U|$ requests of U. In particular, $S^0 = S$ and $S^\ell = S + U$.

Let us go back to the algorithm. To begin, while the requests in U^1 are fetched, DELAY-APX computes T^0 by running $\text{GREEDY}_\rho(S)$, which can be easily seen to imply a total work of $O(d\,h\,n\log n)$. Since $|U^1| = \frac{\epsilon n}{2}$, by spreading the computation of T^0 over the elements of U^1 one can achieve fetch time $O(\epsilon^{-1}dh\log n)$. Now suppose that, for some $i \geq 1$, before fetching any request of U^{i+1} we have $T^{i-1} = \text{GREEDY}_\rho(S^{i-1})$. For $i = 1$ this holds by construction. Recall that every $v \in V(T^{i-1})$ holds an associative array $D(T^{i-1}, v)$ that stores $S^{i-1}(T^{i-1}, v)$. While the requests of U^{i+1}_i are being fetched, DELAY-APX updates the associative arrays of the nodes of T^{i-1} using the requests of U^i_{i-1}. Moreover, it marks every vertex v of T^{i-1} that is reached by more than $\epsilon|D(T^{i-1}, v)|$ of those requests. These are the vertices that may violate ϵ-approximation. Therefore, DELAY-APX takes all the marked vertices and recomputes their subtrees on the updated datasets using GREEDY_ρ. The resulting tree is defined to be T^i.

Now let us bound the fetch time. As we did above for T^0, we shall bound the total work and spread it evenly over the elements of U^{i+1}_i. It is not hard to see that the total work is dominated by the time spent to recompute the trees of the marked vertices, so we shall bound that work. First, each request in U^i_{i-1} reaches at most h vertices. Second, the dataset of each marked vertex has size at most $\frac{1}{\epsilon}$ times the number of requests that reached that vertex. Thus, the total size of the sets at marked vertices is at most:

$$\frac{h\,|U^i_{i-1}|}{\epsilon} = hn2^{-i} \tag{5}$$

Thus, the total work to recompute the subtrees at marked vertices is $O(d\,h^2\,2^{-i}n\log n)$. Distributing this work over the elements fetched from U^{i+1}_i, and recalling that $|U^{i+1}_i| = \epsilon n 2^{-(i+1)}$, yields a fetch time of:

$$O\big(d\,\epsilon^{-1}\,h^2\,\log n\big) \tag{6}$$

This argument is almost but not quite correct. The reason is that we assumed that $T^{i-1} = \text{GREEDY}_\rho(S^{i-1})$, but then we computed a tree T^i which is *not* in general equal to $\text{GREEDY}_\rho(S^i)$. Thus the argument does not correctly propagate the invariant $T^i = \text{GREEDY}_\rho(S^i)$. To fix this issue we perform two changes. First, we slightly tighten the approximation parameter from ϵ to $\frac{\epsilon}{\log_2 n}$. Second, we slightly loosen the invariant from $T^i = \text{GREEDY}_\rho(S^i)$ to T^i being ϵ-approximate w.r.t. (S^i, ρ). At this point the argument carries over, but yields an update time bound slightly worse than that of Theorem 4.5; however, this can be fixed by performing a more refined analysis.

5. Applications

Dynamic ϵ-feasible trees. The first application of our results is, as promised, maintaining ϵ-feasible decision trees in the fully-dynamic model:

Theorem 5.1. FUDY-WC *maintains an ϵ-feasible decision tree with worst-case query time $t_q = O(\beta^{-1}\log n)$ and worst-case update time $t_u = O(\beta^{-3}d\log^4 n)$.*

The result follows by coupling Theorem 4.1 and Theorem 3.3, see Appendix F.1.

Dynamic boosted trees. Let $Y = \{0, 1\}$ and let $c : X \to Y$. We see c as a target function (or concept) to be approximated by our trees. Let P be a distribution over X, and T be a decision tree. The loss of T is:

$$L_P(T, c) = \Pr_{x\sim P}(T(x) \neq c(x))$$

For a split function σ we denote by T_σ its associated decision stump, that is, the tree of height 1 whose sole internal node uses σ. Now let $\mathcal{S}$ be a family of split functions. We say $\mathcal{S}$ satisfies the *weak learning assumption* if there exists $\gamma > 0$ such that, for every distribution P over X, there exists $\sigma \in \mathcal{S}$ such that $L_P(\sigma, c) \leq \frac{1}{2} - \gamma$. It is well known that, if the weak learning assumption holds, then through a simple greedy algorithm one can construct trees based on splits from $\mathcal{S}$ that achieve arbitrarily small loss. More precisely, for every $\delta > 0$ one can construct a tree T with height $h(T) = O(\log\frac{1}{\delta})$ such that $L_P(T, c) \leq \delta$, where the O notation hides constants depending only on γ. This is known as tree boosting (Kearns & Mansour, 1999; Takimoto & Maruoka, 1998). All definitions and results above can be stated for a dataset S by letting P be the uniform distribution over S; in particular $L_S(T, c)$ is the fraction of examples in a dataset S for which T mispredicts the label.

We prove that, under the weak learning assumption, our algorithm FUDY-WC maintains a boosted tree. To this end, suppose for each $i \geq 1$ there is a concept $c_i : X \to Y$ such that the labels of S_i are consistent with c_i (that is, $y = c_i(x)$ for all $(x, y) \in S_i$). Moreover suppose that the weak learning assumption is satisfied w.r.t. c_i. Then we say the weak learning assumption holds for all $i \geq 1$. We show:

Theorem 5.2. *Suppose the weak learning assumption holds for all $i \geq 1$. For any $\delta > 0$* FUDY-WC *can maintain a tree of height $O(\ln\frac{1}{\delta})$ so that, for all $i \geq 1$, the loss over the current dataset satisfies $L_{S_i}(T_i, c) \leq \delta$. Moreover* FUDY-WC *does so in worst-case query time $O(\log\frac{1}{\delta})$ and worst-case update time $O(\delta^{-3}d\log^4 n)$.*

The proof of Theorem 5.2 is in Appendix F.2. The idea is that, for small enough $\epsilon = \Theta(\gamma\delta)$, a tree that is ϵ-approximate has essentially the same gains of the boosted tree. Proving this requires some care, as the construction rule used by the boosted tree (and thus, by FUDY-WC) *is not*

a max-gain rule; it only ensures that the gain is at least $\gamma\delta$ whenever the impurity is at least δ. However, we can show that the ϵ-approximate tree of FUDY-WC has gain $\Omega(\gamma\delta)$, too. The height and the loss are bounded using the arguments of (Kearns & Mansour, 1999; Takimoto & Maruoka, 1998), and the update/query times using our results of Section 3.

6. Lower Bounds

We prove conditional lower bounds based on widely accepted conjectures from computational complexity: the 3SUM and the OV conjectures. Our bounds show that maintaining *exactly* the offline tree requires update/query time *polynomial* in n. Since for $\epsilon = \frac{1}{n+1}$ the only ϵ-approximate tree is the offline tree itself, this shows that, in Theorem 4.1, the polynomial dependence on $\frac{1}{\epsilon}$ of the update time for maintaining an ϵ-approximate tree cannot be avoided, unless 3SUM or OV fail. The same holds for the polynomial dependence on $\frac{1}{\beta}$ of the update time for maintaining a $\boldsymbol{\epsilon}$-feasible tree, see Theorem 5.1.

Our first bound is for dynamic decision trees but in the more general setting of *weighted* examples. More precisely, every example (x, y) appearing in an update request is replaced by a triple (x, y, w) where $w \in \mathbb{N}$; the impurity, gain, and so on are then defined as before, but on the dataset where every example (x, y, w) is replaced by w copies of (x, y). (The unweighted case thus corresponds to having $w = 1$ in all examples). In this weighted setting we prove:

Theorem 6.1. *Let $X = \{0,1\}^d$, $Y = \{0,1\}$, $c \in (0, \frac{1}{3})$. Let ρ be a max-G construction rule with threshold $\alpha = 0$ that labels using the mean or the label distribution[4], where G can be the Gini, the information, or the variance gain. Unless OVH fails, maintaining exactly the decision tree of GREEDY_ρ under weighted examples requires amortized time $\Omega(n^c)$ for some $c > 0$, even if the weights are in $O(n^{3c})$.*

Our second bound is for the unweighted version of the problem, but only for maintaining trees based on the variance gain which label using majority vote. This is nonstandard, as the tree combines the gain used for regression and the labeling rule used for classification.

Theorem 6.2. *Let $X = \mathbb{N}$, $Y = \mathbb{R}$, and G be the variance gain. Let ρ be any max-G threshold construction rule that labels using majority labels. Unless the 3SUM conjecture fails, maintaining exactly the decision tree of GREEDY_ρ requires update and/or query time $\Omega(n^c)$ for some $c > 0$.*

The proofs of Theorem 6.1 and 6.2 are in Appendix H.

[4]This means that, when queried on an unlabeled example x, T returns the distribution of the labels at the leaf v reached by x.

7. Extensions

We conclude by discussing some generalizations of our results that can be found in the Appendix.

Additional stopping conditions. The original definition of $\boldsymbol{\epsilon}$-feasibility (Bressan et al., 2023) constrains v to be a leaf also when $|S(T, v)| \leq k^*$, or when v has depth h^*, where $k^*, h^* \in \mathbb{N}$ are given in input. These stopping conditions are often crucial in practice for decision trees of good quality. We can include these constraints without altering the bounds of Theorem 4.1. For $|S(T, v)| \leq k^*$, define ρ so that $\sigma \in \mathcal{S}$ if and only if $G(S(T, v)) \geq \alpha$ *and* $|S(T, v)| > k^*$. For the the depth of v, define an "enriched" decision function $\hat{\rho}$ that takes in input a pair (S, ζ) where $\zeta \in \mathbb{N}$. One then lets $\hat{\rho}(S, \zeta) = \rho(S)$ if $\zeta > 0$, and $\hat{\rho}(S, \zeta)$ be the majority label of S if $\zeta = 0$. Then, $\text{GREEDY}_{\hat{\rho}}$ at v computes $\hat{\rho}(S(T, v), \zeta)$ where ζ equals h^* minus the depth of v.

General construction rules, gains, etc. Our full bounds are stated in terms of:

- For a construction rule ρ, the maximum time $f_\rho(n)$ needed to compute $\rho(S)$ when $|S| \leq n$. For the rules considered above $f_\rho(n) = O(dn \log n)$, but one could have $f_\rho(n) = O(dn \log^3 n)$ and this would be just reflected in our bounds. See Theorem E.2.
- For a construction rule ρ, the balance of the splits it produces, possibly as a function of the dataset's size. The rules considered above produce $\Theta(\alpha)$-balanced splits, where α is the gain threshold, but the information gain for instance yields $\Theta(\frac{\alpha}{\log n})$-balanced splits. Again, this is taken into account by our full bounds. See Theorem E.2.
- The sensitivity of the gain to changes in the dataset. As said, in the Appendix we do this for three significant gains (Gini gain, information gain, and variance gain). Moreover, as long as the impurity measure is an upper bound (even just asymptotic) on the expected loss, then our bounds for boosted trees will hold, too. This is true for instance for $\sqrt{g}$, used by (Kearns & Mansour, 1999), where g is the Gini impurity. See Theorem F.1.

8. Conclusions and Open Problems

We present the first fully-dynamic algorithm for maintaining a decision tree, with strong guarantees on both the quality of the tree and its worst-case update time. The update time is optimal up to polylogarithmic factors, and our conditional lower bounds show that, in some cases of interest, any dynamic algorithm maintaining exactly the offline tree has an update time polynomial in n. Interesting open problems include the study of other successful decision-tree based algorithms, such as *random forests* and *gradient-boosted* trees, in the fully-dynamic settings. Moreover, we conjecture that there is no efficient algorithm for maintaining exactly the offline tree, even when all examples have unit weights.

Acknowledgements

Marco Bressan is supported in part by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence), and by the EU Horizon CL4-2022-HUMAN-02 research and innovation action under grant agreement 101120237, project ELIAS (European Lighthouse of AI for Sustainability). Mauro Sozio is partially supported by the French National Agency (ANR) under project APY (ANR-20-CE38-0011).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Audenaert, K. M. R. A sharp continuity estimate for the von Neumann entropy. *Journal of Physics A: Mathematical and Theoretical*, 40(28):8127, jun 2007. doi: 10.1088/1751-8113/40/28/S18.

Bressan, M., Damay, G., and Sozio, M. Fully-dynamic decision trees. *Proc. of AAAI*, 2023. Preprint at https://arxiv.org/abs/2212.00778.

Cohen-Addad, V., Hjuler, N., Parotsidis, N., Saulpic, D., and Schwiegelshohn, C. Fully dynamic consistent facility location. In *Proc. of NeurIPS*, 2019.

Das, A., Wang, J., Gandhi, S. M., Lee, J., Wang, W., and Zaniolo, C. Learn smart with less: Building better online decision trees with fewer training examples. In *Proc. of IJCAI*, pp. 2209–2215, 7 2019. doi: 10.24963/ijcai.2019/306.

Domingos, P. and Hulten, G. Mining high-speed data streams. In *Proc. of ACM KDD*, pp. 7180, 2000. ISBN 1581132336. doi: 10.1145/347090.347107. URL https://doi.org/10.1145/347090.347107.

Duetting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Fully dynamic submodular maximization over matroids. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8821–8835, 2023.

Gama, J. a., Rocha, R., and Medas, P. Accurate decision trees for mining high-speed data streams. In *Proc. of ACM KDD*, pp. 523528, 2003. ISBN 1581137370. doi: 10.1145/956750.956813. URL https://doi.org/10.1145/956750.956813.

Hanauer, K., Henzinger, M., and Schulz, C. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, 2022.

Haug, J., Broelemann, K., and Kasneci, G. Dynamic model tree for interpretable data stream learning, 2022. URL https://arxiv.org/abs/2203.16181.

Henzinger, M. and Kale, S. Fully-dynamic coresets. In Grandoni, F., Herman, G., and Sanders, P. (eds.), *Proc. of ESA*, volume 173 of *LIPIcs*, pp. 57:1–57:21, 2020.

Hulten, G., Spencer, L., and Domingos, P. Mining time-changing data streams. In *Proc. of ACM KDD*, pp. 97106, 2001. ISBN 158113391X. doi: 10.1145/502512.502529. URL https://doi.org/10.1145/502512.502529.

Jin, R. and Agrawal, G. Efficient decision tree construction on streaming data. In *Proc. of ACM KDD*, pp. 571576, 2003. ISBN 1581137370. doi: 10.1145/956750.956821. URL https://doi.org/10.1145/956750.956821.

Kearns, M. J. and Mansour, Y. On the boosting ability of top-down decision tree learning algorithms. *J. Comput. Syst. Sci.*, 58(1):109–128, 1999. doi: 10.1006/JCSS.1997.1543. URL https://doi.org/10.1006/jcss.1997.1543.

Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., and Zadimoghaddam, M. Fully dynamic algorithm for constrained submodular optimization. In *Proc. of NeurIPS*, 2020.

Li, R.-H. and Belford, G. G. Instability of decision tree classification algorithms. In *Proc. of ACM KDD*, KDD '02, pp. 570575, 2002. ISBN 158113567X. doi: 10.1145/775047.775131. URL https://doi.org/10.1145/775047.775131.

Manapragada, C., Webb, G. I., and Salehi, M. Extremely fast decision tree. In *Proc. of ACM KDD*, pp. 19531962, 2018. doi: 10.1145/3219819.3220005. URL https://doi.org/10.1145/3219819.3220005.

Patrascu, M. Towards polynomial lower bounds for dynamic problems. In *Proceedings of ACM STOC*, pp. 603610, 2010. ISBN 9781450300506. doi: 10.1145/1806689.1806772. URL https://doi.org/10.1145/1806689.1806772.

Quinlan, J. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Elsevier Science, 1993. ISBN 9781558602380. URL https://books.google.it/books?id=HExncpjbYroC.

Quinlan, J. R. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

Rutkowski, L., Pietruczuk, L., Duda, P., and Jaworski, M. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279, 2013. doi: 10.1109/TKDE.2012.66.

Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014. ISBN 1107057132.

Sun, J., Jia, H., Hu, B., Huang, X., Zhang, H., Wan, H., and Zhao, X. Speeding up very fast decision tree with low computational cost. In *Proc. of IJCAI*, pp. 1272–1278, 7 2020. doi: 10.24963/ijcai.2020/177.

Takimoto, E. and Maruoka, A. On the boosting algorithm for multiclass functions based on information-theoretic criterion for approximation. In *Discovery Science*, pp. 256–267. Springer, 1998. doi: 10.1007/3-540-49292-5_23. URL https://doi.org/10.1007/3-540-49292-5_23.

Topse, F. Bounds for entropy and divergence for distributions over a two-element set. *Journal of Inequalities in Pure & Applied Mathematics*, 2(2):Paper No. 25, 13 p.–Paper No. 25, 13 p., 2001. URL http://eudml.org/doc/122035.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., and Steinberg, D. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008. doi: 10.1007/s10115-007-0114-2. URL https://doi.org/10.1007/s10115-007-0114-2.

A. Bounds for GREEDY$_\rho$

Lemma A.1. GREEDY$_\rho(S)$ *runs in time*

$$O\Big(\hat{h} \cdot \Big(f_\rho(|S|) + d\,|S| \log |S|\Big)\Big)$$

where $\hat{h}$ is the height of the returned tree.

Proof. Recall the pseudocode from Algorithm 2. For each $v \in V(T)$, GREEDY$_\rho$ spends time $f_\rho(S(T,v)) + O(d|S(T,v)| \log |S(T,v)|)$ to compute $\rho(S(T,v))$ and populate $D(T,v)$. If v is internal then GREEDY$_\rho$ also spends time $O(d|S(T,v)|)$ to compute $(S(T,v_1), S(T,v_2)) = \sigma(S(T,v))$ where v_1, v_2 are the children of v; this is dominated by the bound above. The claim follows by the assumptions on f_ρ and since $\sum_{v \in V_i} |S(T,v)| \le |S|$ where V_i by the subset of V at depth i. □

B. Gain measures

We recall the definitions of Gini gain, information gain and variance gain.

B.1. Gini gain

Let $Y = \{1, \dots, k\}$. The *Gini Impurity* of a dataset S is:

$$I_{\text{gini}}(S) = 1 - \sum_{i=1}^{k} p_i^2(S) \tag{7}$$

where $p_i(S)$ is the fraction of examples of S having label i. The *Gini gain* $G_{\text{gini}} : (X \times Y)^* \times \mathcal{S}$ is the conditional I_{gini}-gain.

B.2. Information gain

Let $X \in \mathbb{N}$ be a random variable. The entropy of X is:

$$H(X) = \sum_x \Pr(X = x) \log \frac{1}{\Pr(X = x)} \tag{8}$$

If X takes on at most k distinct values with positive probability then $H(X) \le \log k$. For $p \in [0,1]$ let $H(p)$ be the entropy of a Bernoulli random variable of parameter p. Let $Y, Z \in \mathbb{N}$ be two random variables defined on the same space of events. The conditional entropy of Y given Z is:

$$H(Y|Z) = \sum_z \Pr(Z = z) H(Y|Z = z) \tag{9}$$

The *mutual information* or *information gain* between Y and Z is:

$$\begin{aligned} G_{\text{info}}(Y,Z) &= H(Y) - H(Y|Z) && (10)\\ &= H(Z) - H(Z|Y) && (11)\\ &= H(Y) + H(Z) - H((Y,Z)) && (12) \end{aligned}$$

Now let S be a dataset. The entropy of S and the information gain of a split rule σ on S are defined as follows: letting (X,Y) be a random uniform element of S,

$$\begin{aligned} H(S) &= H(Y) && (13)\\ G_{\text{info}}(S,\sigma) &= G_{\text{info}}(Y, \sigma(X)) && (14) \end{aligned}$$

One can see that $G_{\text{info}} : (X \times Y)^* \times \mathcal{S}$ is the conditional H-gain.

B.3. Variance gain

Let $Y = \mathbb{R}$. The variance of the labels of a set S is:

$$\text{var}(S) = \frac{1}{|S|^2} \sum_{s,s' \in S} (y - y')^2 \tag{15}$$

where $s = (x,y)$ and $s' = (x',y')$. The *variance gain* $G_{\text{var}} : (X \times Y)^* \times \mathcal{S}$ is the conditional var-gain.

C. Smoothness, approximation, and balancedness of gains

In this section we prove that $G \in \{G_{\text{gini}}, G_{\text{info}}, G_{\text{var}}\}$ is smooth w.r.t. the relative edit distance; that is, that for any split rule σ we can bound $|G(S,\sigma) - G(S',\sigma)|$ in term of $\triangle^*(S,S')$. As a consequence we also prove that an ϵ-approximate tree also guarantees a good approximation in terms of G, and that max-G α-threshold construction rules are γ-balanced where α depends on γ. We first prove some ancillary results on the functions g for which G is a conditional g-gain, and then move on to prove the rest.

C.1. Ancillary results

Lemma C.1. *Let $g : (X \times Y)^* \to \mathbb{R}_{\ge 0}$ and let $f : \mathbb{N} \to \mathbb{R}$ be nondecreasing and such that for all datasets S, S':*

$$\begin{aligned} g(S) &\le f(|S|) && (16)\\ |g(S) - g(S')| &\le \frac{f(\max(|S|,|S'|))}{\max(|S|,|S'|)} && (17) \end{aligned}$$

Then for all datasets S, S':

$$|g(S) - g(S')| \le 3\triangle^*(S,S') f(\max(|S|,|S'|)) \tag{18}$$

Proof. For simplicity let $k = \triangle(S,S')$. If $k = 0$ then $g(S) = g(S')$ and the bound is trivial, so assume $k \ge 1$ and $|S'| \ge |S|$. Suppose first $\triangle^*(S,S') \ge \frac{1}{3}$. Then:

$$\begin{aligned} |g(S) - g(S')| &\le \max(g(S), g(S')) && (19)\\ &\le f(\max(|S|,|S'|)) && (20)\\ &\le 3\triangle^*(S,S') f(\max(|S|,|S'|)) && (21) \end{aligned}$$

Now suppose instead $\triangle^*(S,S') < \frac{1}{3}$. Observe that this implies $|S'| \le \frac{3}{2}|S|$ and $|S| \ge 2$, and thus $|S| - 1 \ge \frac{1}{3}|S'|$.

By definition of $\triangle$ there exist $S_0, \ldots, S_k \in (X \times Y)^*$ with $S_0 = S, S_k = S'$, and such that $\triangle(S_i, S_{i+1}) = 1$ for all $i = 0, \ldots, k-1$. Note that in particular there exists such a set where $|S| - 1 \leq |S_i| \leq |S'|$ for all i. By the properties of f this implies:

$$|g(S) - g(S')| \leq \sum_{i=0}^{k-1} |g(S_i) - g(S_{i+1})| \tag{22}$$

$$\leq \sum_{i=0}^{k-1} \frac{f(\max(|S_i|, |S_{i+1}|))}{\max(|S_i|, |S_{i+1}|)} \tag{23}$$

$$\leq \sum_{i=0}^{k-1} \frac{f(|S'|)}{|S| - 1} \tag{24}$$

$$= k \frac{f(|S'|)}{|S| - 1} \tag{25}$$

$$\leq 3k \frac{f(\max(|S|, |S'|))}{\max(|S|, |S'|)} \tag{26}$$

which equals $3\triangle^*(S, S') f(\max(|S|, |S'|))$. □

C.1.1. GINI IMPURITY

Lemma C.2. *If $\triangle(S, S') \leq 1$ then $|I_{\text{gini}}(S) - I_{\text{gini}}(S')| \leq \frac{4}{\max(|S|, |S'|)}$.*

Proof. The claim is trivial if $S = S'$, so assume $S' = S + s$. Without loss of generality we may assume $s = (x, y)$ where $y = 1$. Clearly $I_{\text{gini}}(S) - I_{\text{gini}}(S') = \sum_{i=1}^{k} (p_i^2(S') - p_i^2(S))$, and since $p_i^2(S') - p_i^2(S) = (p_i(S') + p_i(S))(p_i(S') - p_i(S))$ and $(p_i(S') + p_i(S)) \leq 2$, then:

$$|I_{\text{gini}}(S) - I_{\text{gini}}(S')| \tag{27}$$

$$= \left| p_1^2(S') - p_1^2(S) + \sum_{i=2}^{k} (p_i^2(S') - p_i^2(S)) \right| \tag{28}$$

$$\leq 2|p_1(S') - p_1(S)| + 2 \left| \sum_{i=2}^{k} (p_i(S') - p_i(S)) \right| \tag{29}$$

Standard calculations give:

$$p_i(S') - p_i(S) = \begin{cases} \frac{1 - p_i(S)}{n+1} & i = 1 \\ -\frac{p_i(S)}{n+1} & i \geq 2 \end{cases} \tag{30}$$

Thus

$$2|p_1(S') - p_1(S)| + 2 \left| \sum_{i=2}^{k} (p_i(S') - p_i(S)) \right| \tag{31}$$

$$= 2\frac{1 - p_1(S)}{n+1} + 2 \sum_{i=2}^{k} \frac{p_i(S)}{n+1} \tag{32}$$

$$= 4\frac{1 - p_1(S)}{n+1} \tag{33}$$

which is at most $\frac{4}{n+1} = \frac{4}{\max(|S|, |S'|)}$, as claimed. □

C.1.2. ENTROPY

Claim 1. *$H(p) \leq 3p \log \frac{1}{p}$ for all $p \in [0, 1/2]$.*

Proof. Theorem 1.1 of (Topse, 2001) and easy manipulations yield:

$$H(p) \leq \frac{\log p \, \log(1 - p)}{\log 2} \tag{34}$$

$$= \frac{\log \frac{1}{p} \log\left(1 + \frac{p}{1-p}\right)}{\log 2} \tag{35}$$

$$\leq \frac{\log \frac{1}{p}}{\log 2} \frac{p}{1 - p} \tag{36}$$

$$\leq 3p \log \frac{1}{p} \tag{37}$$

□

Claim 2. *For any two random variables X, X' defined on the same space of events and taking on at most n distinct values:*

$$|H(X) - H(X')| \tag{38}$$

$$\leq \text{tvd}(X, X') \log_2(n - 1) + H(\text{tvd}(X, X')) \tag{39}$$

Proof. This is a special case of the FannesAudenaert inequality (Audenaert, 2007) for diagonal matrices. □

Lemma C.3. *If $\triangle(S, S') \leq 1$ then $|H(S) - H(S')| < \frac{5 \log n}{n}$ where $n = \max(|S|, |S'|)$.*

Proof. The claim is trivial if $S = S'$, hence assume $S' = S + s$, and let (X, Y) be a uniform random element of S and (X', Y') a uniform random element of S'. Clearly $\text{tvd}(Y, Y') = \frac{1}{n}$, hence by the definition of $H(S), H(S')$, and by Claim 2 and Claim 1, and since $n \geq 2$ and so $\frac{1}{n} \leq \frac{1}{2}$:

$$|H(S) - H(S')| \leq \frac{1}{n} \log_2(n - 1) + H\left(\frac{1}{n}\right) \tag{40}$$

$$< \frac{2}{n} \log n + \frac{3}{n} \log n \tag{41}$$

concluding the proof. □

C.1.3. VARIANCE

Let $c = \sum_{y \in Y} \frac{c}{2}$ for some $c \in \mathbb{R}_{\geq 0}$. Clearly this implies $\text{var}(S) \leq c^2$ for all S.

Lemma C.4. *If $\triangle(S, S') \leq 1$ then $|\text{var}(S) - \text{var}(S')| \leq \frac{3c^2}{\max(|S|, |S'|)}$.*

Proof. The claim is trivial if $S = S'$, so assume $S' = S + s$ for some $s = (x_s, y_s)$ and let $n = |S'|$. Standard calculations show that:

$$\operatorname{var}(S+s) \tag{42}$$
$$= \operatorname{var}(S)\frac{(n-1)^2}{n^2} + \frac{1}{n^2}\sum_{(x,y)\in S}(y-y_s)^2 \tag{43}$$
$$= \operatorname{var}(S) - \frac{2n-1}{n^2}\operatorname{var}(S) + \frac{1}{n^2}\sum_{(x,y)\in S}(y-y_s)^2 \tag{44}$$

Thus:

$$|\operatorname{var}(S+s) - \operatorname{var}(S)| \tag{45}$$
$$\leq \frac{2n-1}{n^2}\operatorname{var}(S) + \frac{1}{n^2}\sum_{(x,y)\in S}(y-y_s)^2 \tag{46}$$
$$\leq \frac{2}{n}c^2 + \frac{1}{n^2}|S|c^2 \tag{47}$$

which is at most $\frac{3}{n}c^2$, concluding the proof. □

C.2. Smoothness results

Lemma C.5. *Let G be a conditional g-gain where g satisfies the hypotheses of Lemma C.1. Then for all S, S' with $\triangle(S, S') \leq 1$:*

$$|G(S,\sigma) - G(S',\sigma)| \leq 4\frac{f(n)}{n} \tag{48}$$

where $n = \max(|S|, |S'|)$ and f is as in Lemma C.1.

Proof. Let $(S_0, S_1) = \sigma(S)$ and $(S'_0, S'_1) = \sigma(S')$. Without loss of generality let $S' = S + s$ and $S'_0 = S_0$, and let $n = |S'| = \max(|S|, |S'|)$. Standard calculations show that:

$$|G(S,\sigma) - G(S',\sigma)| \tag{49}$$
$$\leq |g(S) - g(S')| + \frac{g(S_0)|S_0|}{n(n-1)} + \left| g(S'_1)\frac{|S'_1|}{n} - g(S_1)\frac{|S_1|}{n-1}\right| \tag{50}$$

By the hypotheses of Lemma C.1 the first term is bounded by $\frac{f(n)}{n}$ and the second term is bounded by $\frac{f(|S_0|)|S_0|}{n(n-1)} \leq \frac{f(|S_0|)}{n} \leq \frac{f(n)}{n}$ too. For the third term, since $\frac{|S'_1|}{n} > \frac{|S_1|}{n-1}$ and again by the hypotheses of Lemma C.1,

$$\left| g(S'_1)\frac{|S'_1|}{n} - g(S_1)\frac{|S_1|}{n-1}\right| \tag{51}$$
$$\leq \left(g(S_1) + \frac{f(|S'_1|)}{|S'_1|}\right)\frac{|S'_1|}{n} - g(S_1)\frac{|S_1|}{n-1} \tag{52}$$
$$= g(S_1)\left(\frac{|S'_1|}{n} - \frac{|S_1|}{n-1}\right) + \frac{f(|S'_1|)}{n} \tag{53}$$
$$= g(S_1)\frac{n-|S'_1|}{n(n-1)} + \frac{f(|S'_1|)}{n} \tag{54}$$
$$\leq f(|S_1|)\frac{n-|S'_1|}{n(n-1)} + \frac{f(|S'_1|)}{n} \tag{55}$$

which, since f is nondecreasing and $|S'_1| \geq 1$, is bounded from above by $2\frac{f(n)}{n}$. We conclude that $|G(S,\sigma) - G(S',\sigma)| \leq 4\frac{f(n)}{n}$, as claimed. □

Lemma C.6. *Let G be a conditional g-gain where g satisfies the hypotheses of Lemma C.1. Then for all S, S':*

$$|G(S,\sigma) - G(S',\sigma)| \leq 12\triangle^*(S,S')f(\max(|S|,|S'|)) \tag{56}$$

with f is as in Lemma C.1.

Proof. Fix σ and let $G(\cdot) = G(\cdot,\sigma)$. By the form of G, we have that:

$$G(S) \leq g(S) \leq f(|S|) \tag{57}$$

Moreover by Lemma C.5 we have that for all S, S' with $\triangle(S, S') \leq 1$:

$$|G(S) - G(S')| \leq 4\frac{f(n)}{n} \tag{58}$$

where $n = \max(|S|, |S'|)$. Therefore $g_G = G$ satisfies the hypotheses of Lemma C.1 with $f_G = 4f$, yielding:

$$|g_G(S) - g_G(S')| \leq 3\triangle^*(S,S')f_G(\max(|S|,|S'|)) \tag{59}$$

that is, $|G(S) - G(S')| \leq 12\triangle^*(S,S')f(\max(|S|,|S'|))$. □

Theorem C.7. *Let σ be any split rule. For all datasets S, S' the distance $|G(S,\sigma) - G(S',\sigma)|$ is at most:*

$$\begin{array}{ll} \triangle^*(S,S')\cdot 48 & G = G_{\text{gini}} \\ \triangle^*(S,S')\cdot 60\,\log\max(|S|,|S'|) & G = G_{\text{info}} \\ \triangle^*(S,S')\cdot 36\,c^2 & G = G_{\text{var}} \end{array} \tag{60}$$

where $c = \sup_{y\in Y}|y|/2$.

Proof. By Lemma C.2, $G = G_{\text{gini}}$ satisfies the hypotheses of Lemma C.6 with $g = I_{\text{gini}}$ and $f = 4$, hence $|G_{\text{gini}}(S,\sigma) - G_{\text{gini}}(S',\sigma)| \leq \triangle^*(S,S') \cdot 48$. By Lemma C.3, $G = G_{\text{info}}$ satisfies the hypotheses of Lemma C.6 with $g = H$ and $f(n) = 5\log n$, hence $|G_{\text{info}}(S,\sigma) - G_{\text{info}}(S',\sigma)| \leq \triangle^*(S,S') \cdot 60\log\max(|S|,|S'|)$. By Lemma C.4, $G = G_{\text{var}}$ satisfies the hypotheses of Lemma C.6 with $g = \text{var}$ and $f(n) = 3c^2$, hence $|G_{\text{var}}(S,\sigma) - G_{\text{var}}(S',\sigma)| \leq \triangle^*(S,S') \cdot 36c^2$. □

C.3. Approximation of maximum gain

The next result says that, for maximum-gain construction rules, an ϵ-approximate guarantees split rules whose gain is close to the maximum possible.

Lemma C.8. *Let ρ be a max-G construction rule with $G \in \{G_{\text{gini}}, G_{\text{info}}, G_{\text{var}}\}$. If a decision tree T is ϵ-approximate w.r.t. (S,ρ), then for every internal vertex $v \in V(T)$ the difference $G(S(T,v),\sigma_v^*) - G(S(T,v),\sigma_v)$ is at most*

$$\begin{cases} \epsilon \cdot 96 & \text{if } G = G_{\text{gini}} \\ \epsilon \cdot 120\,\log|S(T,v)| & \text{if } G = G_{\text{info}} \\ \epsilon \cdot 72c^2 & \text{if } G = G_{\text{var}} \end{cases} \tag{61}$$

where $\sigma_v^* = \rho(S(T,v))$.

Proof. Suppose $G = G_{\text{gini}}$; the proof for $G = G_{\text{info}}$ and $G = G_{\text{var}}$ is similar. Let T be ϵ-approximate w.r.t. (S,ρ) and let $v \in V(T)$ be any internal vertex of T. By Definition 1.2 there exists $S_v \in (X \times Y)^*$ such that $\sigma_v = \rho(S_v)$ and $\triangle^*(S(T,v),S_v) \leq \epsilon$. Then by a double application of Theorem C.7, and since $G(S_v,\sigma_v) \geq G(S_v,\sigma)$ for every $\sigma \in \mathcal{S}$:

$$\begin{aligned} & G(S(T,v),\sigma_v) && (62)\\ &\geq G(S_v,\sigma_v) - 48\,\triangle^*(S(T,v),S_v) && (63)\\ &\geq G(S_v,\sigma_v^*) - 48\,\triangle^*(S(T,v),S_v) && (64)\\ &\geq G(S(T,v),\sigma_v^*) - 96\,\triangle^*(S(T,v),S_v) && (65)\end{aligned}$$

which proves the claim since $\triangle^*(S(T,v),S_v) \leq \epsilon$. □

C.4. Balancedness of threshold construction rules

Lemma C.9. *Let G be a conditional g-gain where g satisfies the hypotheses of Lemma C.1. Then for every $S \in (X \times Y)^*$:*

$$\frac{\min(|S_0|,|S_1|)}{|S|} \geq \frac{G(S,\sigma)}{4f(|S|)} \tag{66}$$

where $(S_0,S_1) = \sigma(S)$ and f is as in Lemma C.1.

Proof. Without loss of generality assume $|S_0| \leq |S_1|$. Let $\eta = \triangle^*(S,S_1)$ and note that $\eta = \frac{|S_0|}{|S|}$ and $1-\eta = \frac{|S_1|}{|S|}$. Then:

$$\begin{aligned} & \frac{|S_0|}{|S|}g(S_0) + \frac{|S_1|}{|S|}g(S_1) && (67)\\ &\geq (1-\eta)g(S_1) && (68)\\ &\geq (1-\eta)(g(S) - 3\eta f(|S|)) && \text{Lemma C.1} \quad (69)\\ &\geq g(S) - \eta(g(S) + 3f(|S|)) && (70)\\ &\geq g(S) - 4\eta f(|S|) && (71)\end{aligned}$$

We conclude that $G(S,\sigma) \leq 4\eta f(n)$ and therefore

$$\frac{\min(|S_0|,|S_1|)}{|S|} = \frac{|S_0|}{|S|} = \eta \geq \frac{G(S,\sigma)}{4f(n)} \tag{72}$$

concluding the proof. □

Theorem C.10. *Let $\mathcal{S}$ be any family of split rules and let $G \in \{G_{\text{gini}}, G_{\text{info}}, G_{\text{var}}\}$. Then any max-$G$ construction rule $\rho : (X \times Y)^* \to \mathcal{S} \cup Y$ with threshold α is γ-balanced, where:*

$$\gamma(|S|) = \begin{cases} \frac{\alpha}{16} & \text{if } G = G_{\text{gini}} \\ \frac{\alpha}{20\log|S|} & \text{if } G = G_{\text{info}} \\ \frac{\alpha}{12c^2} & \text{if } G = G_{\text{var}} \end{cases} \tag{73}$$

where $c = \sup_{y\in Y}|y|/2$.

Proof. Let:

$$\begin{cases} g = I_{\text{gini}} \text{ and } f(n) = 4 & \text{if } G = G_{\text{gini}} \\ g = H \text{ and } f(n) = 5\log n & \text{if } G = G_{\text{info}} \\ g = \text{var} \text{ and } f(n) = 3c^2 & \text{if } G = G_{\text{var}} \end{cases} \tag{74}$$

Then the claim follows by Lemma C.9 by noting that, when $\sigma = \rho(S) \in \mathcal{S}$, by definition of rule with threshold α we have $G(S,\sigma) \geq \alpha$. □

D. Proofs for Section 3

D.1. Proof of Theorem 3.3

We prove the following version that encompasses more general gains.

Theorem D.1. *Let $\mathcal{S}$ be a set of split rules and let $\boldsymbol{\epsilon} = (\alpha,\beta) \in (0,1]^2$. Let $\rho : (X \times Y)^* \to \mathcal{D} \cup Y$ be a max-G construction rule with threshold $\frac{\alpha}{2}$ that assigns majority/average labels, and define:*

$$\epsilon = \begin{cases} \frac{\min(\alpha,\beta)}{100} & G = G_{\text{gini}} \\ \frac{\min(\alpha,\beta)}{130\log n} & G = G_{\text{info}} \\ \frac{\min(\alpha,\beta)}{80c^2} & G = G_{\text{var}},\ c = \sup_{y\in Y}|y|/2 \end{cases}$$

Then, for all datasets S, every decision tree that is ϵ-approximate w.r.t. (S,ρ) is $\boldsymbol{\epsilon}$-feasible w.r.t. (S,ρ).

Suppose first $G = G_{\text{gini}}$. Let $v \in V(T)$. By definition of ϵ-approximation (Definition 1.2) there exists S_v such that $\triangle^*(S(T,v), S_v) \leq \epsilon$ and that the decision taken at v is $\rho(S_v) \in \mathcal{S}$. Let S_v be any such set. Let σ_v and σ_v^* be split rules with maximum gain on respectively S_v and $S(T,v)$:

$$\sigma_v = \arg\max_{\sigma \in \mathcal{S}} G(S_v, \sigma) \tag{75}$$

$$\sigma_v^* = \arg\max_{\sigma \in \mathcal{S}} G(S(T,v), \sigma) \tag{76}$$

Suppose $G(S(T,v), \sigma_v^*) \geq \alpha$. By Theorem C.7, and by the choice of ϵ and the definition of σ_v and σ_v^*:

$$\begin{aligned} G(S_v, \sigma_v) &\geq G(S_v, \sigma_v^*) && (77)\\ &\geq G(S(T,v), \sigma_v^*) - 48\epsilon && (78)\\ &> G(S(T,v), \sigma_v^*) - \alpha/2 && (79)\\ &\geq \alpha/2 && (80) \end{aligned}$$

Similarly, if $G(S(T,v), \sigma_v^*) = 0$ then:

$$\begin{aligned} G(S_v, \sigma_v) &\leq G(S(T,v), \sigma_v) + 48\epsilon && (81)\\ &< G(S(T,v), \sigma_v) + \alpha/2 && (82)\\ &\leq G(S(T,v), \sigma_v^*) + \alpha/2 && (83) \end{aligned}$$

and the rightmost expression equals $\frac{\alpha}{2}$. Since ρ has threshold $\frac{\alpha}{2}$, this proves that v is internal if $G(S(T,v), \sigma_v^*) \geq \alpha$ and v is a leaf if $G(S(T,v), \sigma_v^*) = 0$. Finally, if v is internal then by Lemma C.8:

$$\begin{aligned} G(S(T,v), \sigma_v) &\geq G(S(T,v), \sigma_v^*) - 96\epsilon && (84)\\ &> G(S(T,v), \sigma_v^*) - \beta && (85) \end{aligned}$$

Since the facts above hold for any choice of S_v, we conclude that T is $\boldsymbol{\epsilon}$-feasible as desired.

The proof for $G = G_{\text{info}}$ is similar. If $G(S(T,v), \sigma_v^*) \geq \alpha$ then again by Theorem C.7 and the definition of σ_v and σ_v^*:

$$\begin{aligned} G(S_v, \sigma_v) &\geq G(S_v, \sigma_v^*) && (86)\\ &\geq G(S(T,v), \sigma_v^*) - 60\epsilon \log n && (87)\\ &> G(S(T,v), \sigma_v^*) - \alpha/2 && (88)\\ &\geq \alpha/2 && (89) \end{aligned}$$

Similarly, if $G(S(T,v), \sigma_v^*) = 0$ then:

$$\begin{aligned} G(S_v, \sigma_v) &\leq G(S(T,v), \sigma_v) + 60\epsilon \log n && (90)\\ &< G(S(T,v), \sigma_v) + \alpha/2 && (91)\\ &\leq G(S(T,v), \sigma_v) + \alpha/2 && (92) \end{aligned}$$

Finally, if v is internal then by Lemma C.8:

$$\begin{aligned} G(S(T,v), \sigma_v) &\geq G(S(T,v), \sigma_v^*) - \epsilon \cdot 120 \log n && (93)\\ &> G(S(T,v), \sigma_v^*) - \beta && (94) \end{aligned}$$

The proof for $G = G_{\text{var}}$ is completely analogous.

E. Proofs for Section 4

E.1. Proof of Lemma 4.4

First, we prove that the arrays $D(T,v)$ are updated correctly. For each $v \in V(T^i)$ let $i_v \leq i$ be the most recent iteration where v is in a subtree created by DELAY-APX, or $i_v = 0$ if no such iteration exists. We claim that, for all $i \geq 1$ and all $z \in V(T^i)$, $D(T^i, z)$ stores precisely $S^{i_z}(T^i, z)$. This is trivial for $i = 0$, so suppose the claim holds for T^{i-1} and let $z \in V(T^i)$. If $i_z < i$ then the claim holds for z since $D(T^i, z) = D(T^{i-1}, z)$. Otherwise T_z^i is a subtree of a tree T_v^i that has been returned by DELAY-APX at iteration i, and by induction DELAY-APX has been given in input $D(T^{i-1}, v)$ and all subsequent updates that reached v. By construction of DELAY-APX this implies that $D(T^i, z)$ stores precisely $S^{i_z}(T^i, z)$.

Now consider a generic iteration of FUDY-WC where line 2 is executed. By the claim above and by construction of FUDY-WC, $\mathcal{I}(v)$ was given in input a set of examples and a sequence of updates that define precisely $S^i(T^i, v)$. Thus T_v^i is $\frac{\epsilon}{2}$-approximate w.r.t. $(S^i(T^i, v), \rho)$. Now let $j > i$. If less than $\frac{\epsilon}{2}|S^i(T^i, v)|$ requests have reached v between iteration $i+1$ and the end of iteration j, then T_v^j is ϵ-approximate w.r.t. $(S^j(T^j, v), \rho)$; and before the $\frac{\epsilon}{2}|S^i(T^i, v)|$-th such request reaches v, the subtree T_v^i will be replaced, making it $\frac{\epsilon}{2}$-approximate again.

The claim on the update time follows straightforwardly from the algorithm.

E.2. Proof of Theorem 4.1

We prove a more general result. First we need a lemma. A tree T is η-balanced if every $v, w \in V(T)$ with w child of v satisfy $|S(T,w)| \geq \eta |S(T,v)|$.

Lemma E.1. *If ρ is a γ-balanced construction rule and T is ϵ-approximate w.r.t. (S, ρ), then T is $(\gamma - 2\epsilon)$-balanced.*

Proof. Let S_v be the set for which v satisfies the definition of ϵ-approximate tree. Then:

$$\begin{aligned} |S(T,w)| &\geq |S_v(T,w)| - \triangle(S_v, S(T,v)) && (95)\\ &\geq |S_v(T,w)| - \epsilon |S(T,v)| && (96)\\ &\geq \gamma |S_v| - \epsilon |S(T,v)| && (97)\\ &\geq \gamma(1-\epsilon)|S(T,v)| - \epsilon |S(T,v)| && (98)\\ &\geq (\gamma - 2\epsilon)|S(T,v)| && (99) \end{aligned}$$

The proof is complete. □

Now, for $\epsilon > 0$ and $n \in \mathbb{N}$, let $h_\rho(\epsilon, n)$ be the maximum height, over all S with $|S| \leq n$, of any decision tree ϵ-approximate w.r.t. (S, ρ).

Theorem E.2. *There is a deterministic algorithm for the dynamic approximate decision tree problem,* FUDY-WC*, that for all $i \geq 1$ has query time*

$$O(h(T_i)) = O(h_\rho(\epsilon, n))$$

and update time

$$O\left(h_\rho(\epsilon,n)\,\log n \cdot \left(h_\rho(\epsilon,n) + \frac{d\log n}{\epsilon} + \frac{f_\rho(n)}{\epsilon n}\right)\right)$$

Proof. By Theorem E.3, DELAY-APX is correct and has fetch time in

$$O\left(h_\rho(\epsilon,n)^2\,\log n \cdot \left(h_\rho(\epsilon,n) + \frac{d\log n}{\epsilon} + \frac{f_\rho(n)}{\epsilon n}\right)\right)$$

where n is the maximum size of the dataset in the original problem. Combining this bound with Lemma 4.4 concludes the proof. □

Theorem 4.1 follows from Theorem E.2 by noting that $h_\rho(\epsilon, n) = O\left(\frac{\log n}{\gamma - 2\epsilon}\right)$ because of Lemma E.1, and that $f_\rho(n) = O(dn \log n)$ for max-gain construction rules based on threshold splits because of Lemma G.1.

E.3. Proof of Theorem 4.5

Once again we prove a more general result:

Theorem E.3. *There is an algorithm for the delayed approximate decision tree problem,* DELAY-APX*, that has fetch time*

$$O\left(h\,\log n \cdot \left(h + \frac{d\log n}{\epsilon} + \frac{f_\rho(m)}{\epsilon n}\right)\right)$$

where $m \leq n(1+\epsilon)$ is the maximum size of the dataset obtained from applying any prefix of U to S and $h \leq h_\rho(\epsilon, m)$ is the maximum height of the tree held by the algorithm.

The next subsections prove the following two results, which form the two parts of Theorem E.3 (correctness of DELAY-APX and bound on the fetch time):

Theorem E.4. *T^i satisfies the constraints of Definition 4.3 w.r.t. (S^i, ρ) for all $i \in \{0, \ldots, \ell\}$. In particular, the tree returned by* DELAY-APX *is ϵ-approximate w.r.t. $(S + U, \rho)$.*

Theorem E.5. DELAY-APX *can be implemented to have fetch time $O\left(h\,\log n \cdot \left(h + \frac{d\log n}{\epsilon} + \frac{f_\rho(m)}{\epsilon n}\right)\right)$ where $h = \max_{i=0,\ldots,\ell} h(T_i)$ and $m = \max_{i=0,\ldots,\ell} |S^i|$.*

The correctness is given directly by Theorem E.4. For the fetch time, Theorem E.5 yields a bound of $O\left(h\,\log n \cdot \left(h + \frac{d\log n}{\epsilon} + \frac{f_\rho(m)}{\epsilon n}\right)\right)$ where $h = \max_{i=0,\ldots,\ell} h(T_i)$ and $m = \max_{i=0,\ldots,\ell} |S^i|$. Since by Theorem E.4 every T_i is ϵ-approximate w.r.t. (S^i, ρ) and $|S^i| \leq m$, then by definition $h(T_i) \leq h_\rho(\epsilon, m)$ for all i. Moreover clearly $m \leq |S| + |U| = (1+\epsilon)$.

For convenience let us recall here some definitions and notation. Let $i \in \{0, \ldots, \ell\}$. For every $v \in V(T^i)$:

$$n_i(v) = |S^i(T^i, v)| \tag{100}$$

$$\Delta_i(v) = \left|\{(s,o) \in U^i_{i-1} \;:\; v \in P(T^i, x)\}\right| \tag{101}$$

$$c_i(v) = \begin{cases} 0 \text{ if } i = 0 \text{ or } R^*_i \cap A(T,v) \neq \emptyset \\ c_{i-1}(v) + \Delta_i(v) \text{ otherwise} \end{cases} \tag{102}$$

where $A(T, v)$ denotes the set of all nodes on the path from the root of T to v. Note that:

$$c_i(v) - c_{i-1}(v) \leq \Delta_i(v), \qquad \forall i \geq 1 \tag{103}$$

Our proofs use the following facts.

Lemma E.6. *If $v \in R^*_i$ then $S^i(T^i, v) = S^i(T^{i-1}, v)$.*

Proof. By construction R^*_i contains no proper ancestor of v, hence all such ancestors are untouched by the loop of line 3. □

Lemma E.7. *For every $i = 1, \ldots, \ell$ every $v \in R^*_i$ satisfies $n_i(v) \leq n_{i-1}(v) + \Delta_i(v)$.*

Proof. By Lemma E.6 $|S^i(T^i, v)| = |S^i(T^{i-1}, v)|$, hence:

$$\begin{aligned} n_i(v) &= |S^i(T^i, v)| && (104)\\ &= |S^i(T^{i-1}, v)| && (105)\\ &\leq |S^{i-1}(T^{i-1}, v)| + \Delta_i(v) && (106)\\ &= n_{i-1}(v) + \Delta_i(v) && (107) \end{aligned}$$

where (106) is straightforward and (104),(107) use the definition of n_i. □

E.4. Proof of Theorem E.4

First, we show that T^i is ϵ-approximate w.r.t. (S^i, ρ) for all $i \in \{0, \ldots, \ell\}$. For $i = 0$ the claim is trivial since $T^0 = \text{GREEDY}_\rho(S^0)$. Let then $i \geq 1$, let $v \in V(T^i)$, and let $i_v \in \{0, \ldots, i\}$ be the last round where v was created (i.e., where v or some ancestor of v was in $R^*_{i_v}$). By construction, the split rule at v in T^i is $\rho(S^{i_v}(T^{i_v}, v))$; and by Lemma E.8 below, $c_i(v) \leq \epsilon\,|S^{i_v}(T^{i_v}, v)|$, so $\triangle^*(S^i(T^i, v), S^{i_v}(T^{i_v}, v)) \leq \epsilon$. Thus T^i is ϵ-approximate w.r.t. (S^i, ρ), as claimed.

Next, we show that for every $v \in V(T^i)$ the array $D(T^i, v)$ stores exactly $S^i(T^i, v)$. This is true for $i = 0$ by definition of GREEDY_ρ. Now let $i \geq 1$ and suppose the claim is true for $i - 1$. Because of line 3, at the end of the loop of line 3 each $v \in V(T^{i-1})$ satisfies that $D(T^{i-1}, v)$ stores

$S^i(T^{i-1}, v)$. By definition of GREEDY$_\rho$, then, after the rebuilds at line 3 $D(T^i, v)$ stores $S^i(T^i, v)$ for all $v \in V(T^i)$, as claimed.

For the second claim just note that $S + U = S^\ell$ and DELAY-APX returns T^ℓ.

Lemma E.8. *Let $i \in \{0, \dots, \ell\}$. Then every $v \in V(T^i)$ satisfies $c_i(v) \leq \epsilon\, n_{i_v}(v)$ where $i_v \in \{0, \dots, i\}$ is the last round where v was created.*

Proof. The proof is trivial for $i = 0$ since $c_0(v) = 0$. Let then $i \geq 1$. Since v was built at round i_v then $c_{i_v}(v) = 0$; using a telescoping sum and applying (103),

$$c_i(v) = c_i(v) - c_{i_v}(v) \tag{108}$$

$$= \sum_{j=i_v+1}^{i} (c_j(v) - c_{j-1}(v)) \tag{109}$$

$$\leq \sum_{j=i_v+1}^{i} \Delta_j(v) \tag{110}$$

We shall then bound $\Delta_j(v)$. By definition of i_v, for all $j \in \{i_v + 1, \dots, i\}$ at round j the condition of line 3 fails, hence:

$$\Delta_j(v) < \frac{\epsilon}{4 \log_2 n} n_{j-1}(v) \tag{111}$$

In this case, since $n_j(v) \leq n_{j-1}(v) + \Delta_j(v)$ by Lemma E.7, we have:

$$n_j(v) \leq n_{j-1}(v) \left(1 + \frac{\epsilon}{4 \log_2 n}\right) \tag{112}$$

By iterating (112) we conclude that for every $j = i_v + 1, \dots, i$:

$$n_j(v) \leq \left(1 + \frac{\epsilon}{4 \log_2 n}\right)^{j - i_v} n_{i_v}(v) \tag{113}$$

and (111) then implies for every $j = i_v + 1, \dots, i$:

$$\Delta_j(v) \leq n_{i_v}(v) \cdot \frac{\epsilon}{4 \log_2 n} \left(1 + \frac{\epsilon}{4 \log_2 n}\right)^{j - 1 - i_v} \tag{114}$$

Plugging this bound in (110) and noting that $i \leq 2 \log_2(\epsilon n) \leq 2 \log_2 n$, we obtain:

$$c_i(v) \leq n_{i_v}(v) \cdot \frac{\epsilon}{4 \log_2 n} \sum_{j=i_v+1}^{i} \left(1 + \frac{\epsilon}{4 \log_2 n}\right)^{j - 1 - i_v} \tag{115}$$

$$\leq n_{i_v}(v) \cdot \frac{\epsilon i}{4 \log_2 n} \left(1 + \frac{\epsilon}{4 \log_2 n}\right)^{i} \tag{116}$$

$$\leq n_{i_v}(v) \cdot \frac{\epsilon}{2} \left(1 + \frac{\epsilon}{4 \log_2 n}\right)^{2 \log_2 n} \tag{117}$$

$$< n_{i_v}(v) \cdot \epsilon\, \frac{e^{1/2}}{2} \tag{118}$$

which is at most $\epsilon \cdot n_{i_v}(v)$, as claimed. □

E.5. Proof of Theorem E.5

By Lemma A.1 and by definition of h, GREEDY$_\rho(S)$ runs in time:

$$O\Big(h \cdot \Big(f_\rho(n) + d\, n \log n\Big)\Big) \tag{119}$$

Thus, GREEDY$_\rho(S)$ can be ran by using for each request in U_0^1 a number of operations in:

$$O\left(\frac{h}{\epsilon} \cdot \left(\frac{f_\rho(n)}{n} + d\, \log n\right)\right) \tag{120}$$

Now consider round i. By Lemma E.9 below, the total number of operations taken by the loop at line 3 together with line 3 is in $O(t_i\, dh \log n + |R_i| h \log n)$. As each iteration of that loop inserts at most h elements in R_i then $|R_i| \leq t_i\, h$, so the bound above is in $O(t_i \cdot (d + h) h \log n)$. Hence, excluding line 3, the i-th round can be ran in fetch time $O((d + h) h \log n)$. It remains to bound the time taken by the loop at line 3, which is dominated by the total time of the invocations of GREEDY$_\rho$. Let then $v \in R_i^*$ and consider $D(T, v)$. By construction, $D(T, v)$ stores $S^i(T^{i-1}(v))$, which by Lemma E.6 equals $S^i(T^i, v)$. Thus, by Lemma A.1, by definition of h, and by the assumptions on f_ρ the total time of the invocations of GREEDY$_\rho$ is bounded asymptotically by:

$$\sum_{v \in R_i^*} h \cdot \Big(f_\rho(n_i(v)) + d\, n_i(v) \log n_i(v)\Big) \tag{121}$$

$$= h\, f_\rho\left(\sum_{v \in R_i^*} n_i(v)\right) + dh \log n \sum_{v \in R_i^*} n_i(v) \tag{122}$$

Thus, we shall bound $\sum_{v \in R_i^*} n_i(v)$. Fix any $v \in R_i^*$. By line 3:

$$n_{i-1}(v) \leq \frac{4 \log_2 n}{\epsilon} \Delta_i(v) \tag{123}$$

Moreover $n_i(v) \leq n_{i-1}(v) + \Delta_i(v)$ by Lemma E.7, hence:

$$n_i(v) \leq \left(1 + \frac{4 \log_2 n}{\epsilon}\right) \Delta_i(v) \leq \frac{5 \log_2 n}{\epsilon} \Delta_i(v) \tag{124}$$

Since no two vertices in R_i^* are in an ancestor-descendant relationship, every $(s, o) \in U_{i-1}^i$ reaches at most one vertex in R_i^*. Therefore:

$$\sum_{v \in R_i^*} \Delta_i(v) \leq |U_{i-1}^i| = t_{i-1} - t_i = t_i \tag{125}$$

We conclude that:

$$\sum_{v \in R_i^*} n_i(v) \leq \frac{5 \log_2 n}{\epsilon} \cdot t_i \tag{126}$$

We can now bound the two terms of (122). For the first term, we consider two cases. If $\frac{t_i \, 5 \log_2 n}{\epsilon n} > 1$ then we use again the fact that no two vertices in R_i^* are in an ancestor-descendant relationship to obtain $\sum_{v \in R_i^*} n_i(v) \leq |S^i| \leq m$, which yields:

$$f_\rho \left(\sum_{v \in R_i^*} n_i(v) \right) \leq f_\rho(m) < \frac{f_\rho(m) \, t_i \, 5 \log_2 n}{\epsilon n} \tag{127}$$

If instead $t_i \leq \frac{\epsilon n}{5 \log_2 n}$ then observe that, by its assumptions, f_ρ satisfies $f_\rho(x) \leq \frac{f_\rho(cx)}{c}$ for all $x \in \mathbb{R}_{\geq 0}$ and all $c \geq 1$. Using (126) and choosing $c = \frac{\epsilon n}{t_i 5 \log_2 n} < 1$, we obtain:

$$f_\rho \left(\sum_{v \in R_i^*} n_i(v) \right) \leq f_\rho \left(\frac{5 \log_2 n}{\epsilon} \cdot t_i \right) \tag{128}$$

$$\leq \frac{f_\rho(n) \, t_i \, 5 \log_2 n}{\epsilon n} \tag{129}$$

$$\leq \frac{f_\rho(m) \, t_i \, 5 \log_2 n}{\epsilon n} \tag{130}$$

Therefore the first term of (122) is bounded by:

$$O\left(t_i \cdot \frac{h \, f_\rho(m) \, \log_2 n}{\epsilon n} \right) \tag{131}$$

For the second term of (122), again by (126) we obtain:

$$dh \log n \sum_{v \in R_i^*} n_i(v) = O\left(t_i \cdot \frac{h \, d \log^2 n}{\epsilon} \right) \tag{132}$$

Thus the total number of operations performed by GREEDY$_\rho$ in the loop of line 3 is in:

$$O\left(t_i \cdot \frac{h \, \log n}{\epsilon} \left(\frac{f_\rho(m)}{n} + d \log n \right) \right) \tag{133}$$

Summing all bounds and dividing by t_i, one obtains the following bound on the fetch time for each round $i = 1, \ldots, \ell$:

$$O\left((d+h) h \log n + \frac{h \, \log n}{\epsilon} \left(\frac{f_\rho(m)}{n} + d \log n \right) \right) \tag{134}$$

Since (134) dominates (120), then it bounds the fetch time of DELAY-APX. By rearranging terms, we obtain that (134) is bounded by:

$$O\left(h \, \log n \cdot \left(h + \frac{d \log n}{\epsilon} + \frac{f_\rho(m)}{\epsilon n} \right) \right) \tag{135}$$

which concludes the proof.

Lemma E.9. DELAY-APX *can be implemented so that, at every round $i \geq 1$:*

1. *each iteration of the loop at line 3 takes $O(d \, h \log n)$ operations*
2. *line 3 takes $O(|R_i| \, h \log n)$ operations*
3. *line 3 can enumerate R_i^* in $O(1)$ per element*

Proof. 1. By definition $|P(T,s)| \leq h$, hence computing $P(T,s)$ takes time $O(h)$. For each $v \in P(T,s)$, updating $D(T,v)$ takes time $O(d \log n)$ by assumption. To increment $\Delta_i(v)$ in time $O(1)$, create it as a new variable associated to v the first time v is processed by the loop of line 3 and set it to 1, then mark v as "alive" so that subsequent updates increment that variable. Assuming we can access $n_{i-1}(v)$ in time $O(1)$, checking the condition at line 3 takes time $O(1)$. Finally, updating R_i takes time $O(\log |V(T)|) = O(\log n)$ using an associative array with logarithmic update time. It remains to show how line 3 can access $n_{i-1}(v)$ in $O(1)$ operations.

Consider again $D(T,v)$. Just before round $i+1$ starts, $D(T,v)$ stores $S^i(T^i, v)$. This is true for $i = 0$ since GREEDY$_\rho$ stores explicitly $S^0(T^0, v)$ in $D(T,v)$; and it remains true for $i \geq 1$ since either v is in a subtree rebuilt at round i, and the argument above applies, or $D(T,v)$ is updated by line 3. Thus, for each $i \geq 1$, at the beginning of round i we can access $n_{i-1}(v)$ in time $O(1)$ by querying $|D(T,v)|$. To make it available throughout all the round, right before executing line 3 query $|D(T,v)|$ and store it in a new variable $\hat{n}_{i-1}(v)$, then mark v as "done" so that $\hat{n}_{i-1}(v)$ does not get overwritten. From this point onward, for any $v \in V(T)$ one can retrieve $n_{i-1}(v)$ in time $O(1)$ by using $\hat{n}_{i-1}(v)$ if it exists, and using $|D(T,v)|$ otherwise.

2,3. Initialise an empty linked list $R_i^* = \emptyset$. For every $v \in R_i$, list all the ancestors of v in T — this takes time $O(h)$ as every vertex of T keeps a pointer to its parent — and if none of them is in R_i then append v to R_i^*. □

E.6. Proof of Theorem 4.2

We assume memory can be managed via two primitives:

- MALLOC, which takes as a parameter a positive integer k, and returns a pointer to a block of n contiguous words all set to zero. The running time is $O(k)$.
- FREE, which takes as a parameter a pointer returned by a previous call to MALLOC. The running time is $O(1)$.

The algorithm has access to $O(1)$ zeroed words of memory (the registers); any additional space must be allocated with MALLOC. At any point in time, the *total space* used by the algorithm is the sum of the sizes of all the blocks MALLOC-ed so far, minus the sizes of those that have been FREE-ed. (Blocks whose allocation is undergoing count as completely

allocated).

Recall FUDY-WC. Call *round* i the sequence of operations that start with the i-th update request and terminate just before the $(i+1)$-th request. Let τ_i be the update time at round i, let $n_i = |S_i|$ be the size of the dataset defined by the first i requests, and let $n_i^* = \max_{i \le j} n_j$.

Adapting the algorithm. It is immediate to adapt FUDY-WC and its subroutines so that it uses MALLOC and it accesses only memory that has bees MALLOC-ed before, without increasing the asymptotic worst-case running time (the time spent on accessing memory already bounds asymptotically the time taken by a MALLOC to allocate that memory). In the remainder we describe how FUDY-WC uses FREE and bound the total space used at any time.

We equip FUDY-WC with a *garbage queue*: a linked list that stores pointers to memory blocks (or, more generally, to structures such as decision trees, binary search trees, linked lists, ...). The queue is initially empty, and FUDY-WC can push a pointer to the back the queue at any time. A *collector* algorithm then takes an element from the front of the list (if any) and invokes FREE on it. If the pointer is the node of a decision or search tree, or a node of a linked list, then the garbage collector push to the back of the queue all successors of that pointer (the pointers to the subtrees, or the pointer to the next element in the list); this takes time $O(1)$ since the position of those successors is known by design. Thus, after every $O(1)$ operations the collector frees space $\Omega(1)$.

Finally, we adapt FUDY-WC and its subroutines to use the garbage collector, as follows:

- GREEDY$_\rho$ pushes all pointers to the blocks of memory used temporarily (i.e., not part of the returned tree) into the garbage queue right before return time. This clearly leaves the asymptotic running time unchanged.
- for DELAY-APX, just after line 3 we push the old tree T_v in the garbage queue. Moreover, at the end of each iteration of the loop of line 3 we push R_i, R_i^*, as well as the blocks used for the counters into the garbage queue. Note that this leaves the asymptotic running time per update unchanged.
- for FUDY-WC, after 2 we push the old tree T_v as well as $U(v)$ into the garbage queue. Note again that this leaves the asymptotic running time per update unchanged.

We can proceed to bound the space used by FUDY-WC. At any point in time, the *active space* is the total space minus the total size of the blocks reachable by the pointers in the garbage queue. An analysis of FUDY-WC and its subroutines shows:

Claim 3. *For some constant $C > 0$, at any point in time during round i the active space used by* FUDY-WC *is at most $C\, h_\rho(\epsilon, n_i)^2\, d\, n_i$.*

Just before round $i+1$, run the collector until the garbage queue becomes empty or until it frees space at least:

$$\tau_i + D \cdot h_\rho(\epsilon, n_i^*)^2\, d \tag{136}$$

for some constant D to be fixed later. We then have:

Claim 4. *For some constant $B > 0$, for all $i \ge 1$ the total space used by* FUDY-WC *at the end of round i is at most $B\, h_\rho(\epsilon, n_i^*)^2\, d\, n_i$.*

Proof. The claim holds for $i = 1$ for B large enough (recall that we always assume $n_i \ge 1$). Now suppose the claim holds for some $i - 1 \ge 1$; we show it holds for i, too. We consider two cases:

1. at the end of round i the garbage queue is empty. In this case the total space at the end of round i is just the active space, which as shown above is bounded by $C\, h_\rho(\epsilon, n_i)^2\, d\, n_i$; the claim then holds for every $B \ge C$.

2. at the end of round i the garbage queue is not empty. In this case the garbage collector has freed space at least $\tau_i + D \cdot h_\rho(\epsilon, n_i^*)^2\, d$. By inductive hypothesis, then, the total space used at the end of round i is at most:

$$\begin{aligned} & B\, h_\rho(\epsilon, n_{i-1}^*)^2\, d\, n_{i-1} + \tau_i && (137)\\ & \qquad - \left(\tau_i + D \cdot h_\rho(\epsilon, n_i^*)^2\, d\right) \\ & \le B\, h_\rho(\epsilon, n_i^*)^2\, d\, (n_i + 1) + \tau_i && (138)\\ & \qquad - \left(\tau_i + D \cdot h_\rho(\epsilon, n_i^*)^2\, d\right) \\ & = B\, h_\rho(\epsilon, n_i^*)^2\, d\, \left(n_i + 1 - \frac{D}{B}\right) && (139) \end{aligned}$$

   which for $D \ge B$ is at most $B\, h_\rho(\epsilon, n_i^*)^2\, d\, n_i$ and thus proves the claim in this case too.

The proof is complete. □

To prove Theorem 4.2, observe how the facts above imply that for all $i \ge 1$:

1. the update time at round i increases by an additive $O(\tau_i + h_\rho(\epsilon, n_i^*)^2\, d)$. The update time at round i was originally τ_i by definition, and moreover the bound of Theorem E.2 dominates $h_\rho(\epsilon, n_i^*)^2\, d$ from above.
2. the total space used at any point during round i is at most $\tau_i + O(h_\rho(\epsilon, n_i^*)^2\, d\, n_i)$.

F. Proofs for Section 5

F.1. Proof of Theorem 5.1

We prove a slightly different statement, from which Theorem 5.1 follows by substituting ϵ.

Theorem F.1. *Let $\mathcal{S}$ be the set of split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$ or $\sigma(x) = \mathbb{1}_{x_j=t}$, let $G \in \{G_{\text{gini}}, G_{\text{info}}, G_{\text{var}}\}$, and let $\boldsymbol{\epsilon} = (\alpha, \beta) \in (0,1]^2$. One can maintain an $\boldsymbol{\epsilon}$-feasible decision tree with update time $O\left(\frac{d \log^4 n}{\epsilon^3}\right)$, where*

$$\epsilon = \begin{cases} \frac{\min(\alpha,\beta)}{100} & G = G_{\text{gini}} \\ \frac{\min(\alpha,\beta)}{130 \log n} & G = G_{\text{info}} \\ \frac{\min(\alpha,\beta)}{80c^2} & G = G_{\text{var}},\ c = \sup_{y \in Y} |y|/2 \end{cases}$$

Proof. Let us first consider the case $G = G_{\text{gini}}$. Define ρ to be a max-gain construction rule with threshold $\frac{\alpha}{2}$ that assigns majority/average labels. By Theorem C.10, ρ is γ-balanced for $\gamma = \frac{\alpha}{32}$. By Theorem G.1, $f_\rho(n) = O(dn \log n)$. Let $\epsilon = \frac{\min(\alpha,\beta)}{100}$; clearly $\epsilon < \frac{\gamma}{2}$ and $\gamma - 2\epsilon = \Theta(\epsilon)$. By Theorem 4.1, there is an ϵ-approximate decision tree algorithm with update time $O\left(\frac{d \log^4 n}{\epsilon^3}\right)$. It is straightforward to keep at every leaf v of T a priority queue that maps every label to its count and, in constant time, returns a majority label or average label of $S(T, v)$ without altering the running time. To show that the tree maintained by the algorithm is $\boldsymbol{\epsilon}$-feasible, consider the current dataset S^i and the current tree T^i and apply Theorem D.1.

The proof for $G = G_{\text{info}}$ is similar. Define ρ as above. By Theorem C.10, ρ is γ-balanced where $\gamma(n) = \frac{\alpha}{40 \log n}$, by Theorem G.1 $f_\rho(n) = O(dn \log n)$, and letting $\epsilon = \frac{\min(\alpha,\beta)}{130 \log n}$ ensures $\epsilon < \frac{\gamma}{2}$ and $\gamma - 2\epsilon = \Theta(\epsilon)$. The $O\left(\frac{d \log^4 n}{\epsilon^3}\right)$ bound and the $\boldsymbol{\epsilon}$-feasibility follows like above.

The proof for $G = G_{\text{var}}$ is completely analogous. □

F.2. Proof of Theorem 5.2

Let $Y = \{0, 1\}$ and let $c \subseteq X$ be a target concept. We extend the definition of gain measure G from datasets to distributions P over X. To ease the notation, for a split σ we also denote by σ the set of examples x that satisfy $\sigma(x) = 1$, and by $\overline{\sigma}$ its complement $X \setminus \sigma$. Moreover, for every distribution P let $P|\sigma$ be the conditional distribution over σ. Finally, for every distribution P we let $g(P) = P(c)(1 - P(c))$. Define:

$$G(P, \sigma) = g(P) - (P(\sigma)g(P|\sigma) + P(\overline{\sigma})g(P|\overline{\sigma}))$$

Let $\mathcal{S}$ be a family of split function. We say $\mathcal{S}$ satisfies the *weak learning assumption* if there exists $\gamma > 0$ such that, for every distribution P over X, there exists $\sigma \in \mathcal{S}$ such that $G(P, \sigma) \geq \gamma\, G(P, \sigma)$. This is the basic assumption under many boosting techniques, and in particular under the techniques for boosting trees of (Kearns & Mansour, 1999) and (Takimoto & Maruoka, 1998).[5] Now, a boosted tree T with accuracy (i.e., expected binary loss) at most $\delta > 0$ can be obtained by the following construction rule ρ. Let v be a leaf in T, and let P_v be the distribution of the examples that reach v. The *loss* at v is $L(P_v) = \min(P_v(c), 1 - P_v(c))$, i.e., the error rate of the majority label of P_v. Note that $L(P_v) \leq g(P_v)$. If $g(P_v) \leq \delta$ then v remains as is it, and ℓ_v is the majority label of P_v, so $L(P_v) \leq \delta$. If instead $g(P_v) > \delta$, then let P'_v be the *balanced* version of P_v, that is, the one defined as

$$P'_v(x) = \begin{cases} \frac{P_v(x)}{2P_v(c)} & x \in c \\ \frac{P_v(x)}{2(1-P_v(c))} & x \notin c \end{cases} \tag{140}$$

so that $P'_v(c) = 1 - P'_v(c) = \frac{1}{2}$. The construction rule then chooses for v a split rule $\sigma_v \in \mathcal{S}$ such that

$$G(P'_v, \sigma_v) \geq \gamma \cdot g(P'_v) \tag{141}$$

Note that σ_v exists by the weak learning assumption. By Proposition 5 of (Takimoto & Maruoka, 1998), the same inequality on the original distribution P_v:

$$G(P_v, \sigma_v) \geq \gamma \cdot g(P_v) \geq \gamma \cdot \delta \tag{142}$$

(Note that they use G for g, and therefore our $G(P, \sigma)$ is their $G_P(c) - G_P(c|\sigma)$). As shown by (Takimoto & Maruoka, 1998), for every $\delta > 0$ the construction rule ρ produces a tree T with height $h(T) = O(\log \frac{1}{\delta})$, whose expected loss over P is at most δ. (Note that γ is fixed and independent of γ; the $O()$ notation in $h(T)$ thus hides universal constants depending only on γ but not on δ).

Now let $\epsilon = \frac{\gamma \cdot \delta}{96} = \Theta(\delta)$. Let S be a dataset, and let T be a tree that is ϵ-approximate w.r.t. (S, ρ). (All arguments above apply by letting each distribution be the uniform over the relevant dataset; e.g., P_v will be the uniform distribution over $S(T, v)$, and so on). We want to show that T has loss at most 2δ over S. Let $v \in V(T)$. If v is a leaf, then by definition of ϵ-approximation there exists S_v with $\triangle^*(S(T, v)S_v) \leq \epsilon$ such that $\rho(S_v)$ outputs a label. By definition of ρ this means that $g(S_v) \leq \delta$. By Lemma C.2, and Lemma C.1 with $f = 4$,

$$g(S(T, v)) \leq g(S_v) + 12\epsilon < 2\delta \tag{143}$$

Suppose instead that v is internal with split σ_v. Again by definition of ϵ-approximation there exists S_v with $\triangle^*(S(T, v)S_v) \leq \epsilon$ such that $\rho(S_v)$ outputs σ_v. By definition of ρ this means $g(S_v) > \delta$, and by the arguments above,

$$G(S_v, \sigma_v) \geq \gamma \cdot g(P_v) \geq \gamma \cdot \delta \tag{144}$$

[5] To be precise, the assumption usually says that there exists a decision stump with *loss* at most $\frac{1}{2} - \gamma$. (Takimoto & Maruoka, 1998) however show that, for the balanced distributions that are used in the proof, one can equivalently use the gain.

Applying Lemma C.6, again with $f = 4$, we obtain:

$$G(S(T, v), \sigma_v) \geq G(S_v, \sigma_v) - 48\epsilon \geq \frac{\gamma \cdot \delta}{2} \tag{145}$$

Again by the arguments of (Takimoto & Maruoka, 1998), the fraction of examples of S that T mispredicts is at most 2δ, and moreover T has height $O(\log \frac{1}{\delta}) = O(\log \frac{1}{\epsilon})$.

Finally, observe that the bounds of Theorem G.1 on the cost f_ρ of computing ρ apply to *our* ρ, too. Indeed, one just needs to compute the relative frequency of each label, and then (implicitly) rebalance the distribution by weighing each example appropriately when computing G.

G. Complexity of construction rules

Recall that f_ρ is the complexity of computing ρ as a function of the length of the input. In this section we bound f_ρ for some common construction rules, proving:

Theorem G.1. *Let $\mathcal{S}$ be the set of all split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$ or $\sigma(x) = \mathbb{1}_{x_j=t}$, let $G \in \{G_{\text{gini}}, G_{\text{info}}, G_{\text{var}}\}$, and let $\rho : (X \times Y) \to \mathcal{S} \cup Y$ be a max-G threshold construction rule that assigns majority or average labels. Then $f_\rho(n) \in O(d\, n \log n)$.*

As a majority/average label can be computed in time $O(|S| \log |S|)$, to prove Theorem G.1 it is sufficient to show that $\arg\max_{\sigma \in \mathcal{S}} G(S, \sigma)$ can be computed in time $O(d|S| \log |S|)$, which we do in Lemma G.2, Lemma G.3 and Lemma G.4. Note that split rules in the form $\mathbb{1}_{x_j \leq t}, \mathbb{1}_{x_j \geq t}, \mathbb{1}_{x_j > t}$ are captured by Theorem G.1 by replacing x_j with $-x_j$ and/or σ with $1 - \sigma$.

Lemma G.2. *Let $\mathcal{S}_j$ be the set of all split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$ or $\sigma(x) = \mathbb{1}_{x_j=t}$. Then $\arg\max_{\sigma \in \mathcal{S}_j} G_{\text{gini}}(S, \sigma)$ can be computed in time $O(|S| \log |S|)$.*

Proof. Suppose $\mathcal{S}_j$ is the set of all split rules in the form $\mathbb{1}_{x_j=t}$. Let $S \in (X \times Y)^*$, and for every value t in the domain of the j-th feature let σ_t be the rule defined by $\sigma_t(x) = \mathbb{1}_{x_j=t}$. Let $\mathrm{C}, \mathrm{C_L}, \mathrm{L}$ be associative arrays with logarithmic access/update time and linear enumeration time. First, in time $O(|S| \log |S|)$, go through every $(x, y) \in S$ and increase $\mathrm{C}[x_j]$, $\mathrm{C_L}[x_j][y]$, $\mathrm{L}[y]$, and compute $|S|$ and $Q = \sum_{y \in L} (\mathrm{L}[y])^2$. Then in time $O(1)$ compute:

$$I_{\text{gini}}(S) = 1 - \frac{Q}{|S|^2} \tag{146}$$

Now let $t \in \mathrm{C}$. Observe that, if $(S_{t,0}, S_{t,1}) = \sigma_t(S)$, then:

$$I_{\text{gini}}(S_{t,0}) \tag{147}$$

$$= 1 - \frac{Q - \sum_{y \in \mathrm{C_L}[t]} \left((L[y])^2 - (L[y] - \mathrm{C_L}[t][y])^2 \right)}{(|S| - \mathrm{C}[t])^2} \tag{148}$$

and:

$$I_{\text{gini}}(S_{t,1}) = 1 - \sum_{y \in \mathrm{C_L}[t]} \frac{(\mathrm{C_L}[t][y])^2}{(\mathrm{C}[t])^2} \tag{149}$$

Note that $I_{\text{gini}}(S_{t,0})$ and $I_{\text{gini}}(S_{t,1})$ can be computed in time $O(|\mathrm{C_L}[t]| \log |S|)$ by iterating on $\mathrm{C_L}[t]$. Since $|\mathrm{C_L}[t]| \leq \mathrm{C}[t]$ and $\sum_t \mathrm{C}[t] = |S|$, then in time $O(|S| \log |S|)$ one can compute $I_{\text{gini}}(S_{t,0})$ and $I_{\text{gini}}(S_{t,1})$ for all $t \in \mathrm{C}$ and therefore (using $I_{\text{gini}}(S)$ and $\mathrm{C}[t]$) also $G_{\text{gini}}(S, \sigma_t)$. In time $O(|S|)$ one then finds and returns $t^* = \arg\max_{t \in \mathrm{C}} G_{\text{gini}}(S, \sigma_t)$.

For the case $\mathbb{1}_{x_j<t}$, sort the distinct keys of C by increasing value in time $O(|S| \log |S|)$; let them be $t_1 < \ldots < t_k$. For $i = 1, \ldots, k$ we keep track of cumulative versions of C and $\mathrm{C_L}$ that store:

$$\mathrm{C_L^{\leq}}[t_i][y] = \sum_{j=1}^{i} \mathrm{C_L}[t_j][y] \tag{150}$$

$$\mathrm{C^{\leq}}[t_i] = \sum_{j=1}^{i} \mathrm{C}[t_j] \tag{151}$$

Let $N_{0,0} = 0$ and $N_{0,1} = 1$, and for all $i = 1, \ldots, k$ define:

$$N_{i,0} = \sum_{y \in L} \left(\sum_{j=1}^{i} \mathrm{C_L}[t_j][y] \right)^2 \tag{152}$$

$$= \sum_{y \in L} \left(\mathrm{C_L^{\leq}}[t_i][y] \right)^2 \tag{153}$$

$$N_{i,1} = \sum_{y \in L} \left(L[y] - \sum_{j=1}^{i} \mathrm{C_L}[t_j][y] \right)^2 \tag{154}$$

$$= \sum_{y \in L} \left(L[y] - \mathrm{C_L^{\leq}}[t_i][y] \right)^2 \tag{155}$$

Note that, if $(S_{t_i,0}, S_{t_i,1}) = \sigma_{t_i}(S)$, then:

$$I_{\text{gini}}(S_{t_i,0}) = 1 - \frac{N_{i,0}}{\left(|S| - \mathrm{C_L^{\leq}}[t_i] \right)^2} \tag{156}$$

$$I_{\text{gini}}(S_{t_i,1}) = 1 - \frac{N_{i,1}}{\left(\mathrm{C_L^{\leq}}[t_i] \right)^2} \tag{157}$$

It is not hard to compute $\mathrm{C_L^{\leq}}[t_{i+1}][y]$ and $\mathrm{C^{\leq}}[t_{i+1}]$ from $\mathrm{C_L^{\leq}}[t_i][y]$ and $\mathrm{C^{\leq}}[t_i]$ in time $O(|\mathrm{C_L}[t_{i+1}]|)$, and therefore to compute $N_{i,0}$ and $N_{i,1}$ and thus $I_{\text{gini}}(S_{t_i,0})$ and $I_{\text{gini}}(S_{t_i,1})$ for all i in total time $O(|S| \log |S|)$. This implies the claim in the same way as in the previous case. □

Lemma G.3. *Let $\mathcal{S}_j$ be the set of all split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$ or $\sigma(x) = \mathbb{1}_{x_j=t}$. Then $\arg\max_{\sigma \in \mathcal{S}_j} G_{\text{info}}(S, \sigma)$ can be computed in time $O(|S| \log |S|)$.*

Proof. Suppose $\mathcal{S}_j$ is the set of all split rules in the form $\mathbb{1}_{x_j=t}$. Using the same notation of Lemma G.2, note that:

$$H(S) = \sum_{y\in \mathrm{L}} \frac{\mathrm{L}[y]}{|S|} \log \frac{|S|}{\mathrm{L}[y]} \tag{158}$$

Moreover, for every $t \in \mathrm{C}$:

$$H(S_{t,1}) = \sum_{y\in \mathrm{C_L}[t]} \frac{\mathrm{C_L}[t][y]}{\mathrm{C}[t]} \log \frac{\mathrm{C}[t]}{\mathrm{C_L}[t][y]} \tag{159}$$

Thus we can compute $H(S_{t,1})$ in time $O(|\mathrm{C_L}[t]| \log |S|)$. To show that the same holds for $H(S_{t,0})$ we need some more manipulations. Note that:

$$H(S_{t,0}) \tag{160}$$

$$= \sum_{y\in \mathrm{L}} \frac{\mathrm{L}[y] - \mathrm{C_L}[t][y]}{|S| - \mathrm{C}[t]} \log \frac{|S| - \mathrm{C}[t]}{\mathrm{L}[y] - \mathrm{C_L}[t][y]} \tag{161}$$

$$= \sum_{y\in \mathrm{L}} \frac{\mathrm{L}[y]}{|S| - \mathrm{C}[t]} \log \frac{|S| - \mathrm{C}[t]}{\mathrm{L}[y]} \tag{162}$$

$$+ \sum_{y\in \mathrm{C_L}[t]} \left(\frac{\mathrm{L}[y] - \mathrm{C_L}[t][y]}{|S| - \mathrm{C}[t]} \log \frac{|S| - \mathrm{C}[t]}{\mathrm{L}[y] - \mathrm{C_L}[t][y]} - \frac{\mathrm{L}[y]}{|S| - \mathrm{C}[t]} \log \frac{|S| - \mathrm{C}[t]}{\mathrm{L}[y]} \right)$$

The second summation can clearly be computed in time $O(|\mathrm{C_L}[t]| \log |S|)$. The first summation can instead be written as:

$$\frac{\log(|S| - \mathrm{C}[t])}{|S| - \mathrm{C}[t]} \sum_{y\in \mathrm{L}} \mathrm{L}[y] - \frac{1}{|S| - \mathrm{C}[t]} \sum_{y\in \mathrm{L}} \mathrm{L}[y] \log \mathrm{L}[y] \tag{163}$$

which can be computed in time $O(\log |S|)$ if we precompute $\sum_{y\in \mathrm{L}} \mathrm{L}[y] \log \mathrm{L}[y]$ — note that $\sum_{y\in \mathrm{L}} \mathrm{L}[y] = |S|$. We conclude that $G_{\mathrm{info}}(S, \sigma_t)$ can be computed in time $O(|\mathrm{C_L}[t]| \log |S|)$. Since $\sum_t |\mathrm{C_L}[t]| \leq |S|$, one can compute $G_{\mathrm{info}}(S, \sigma_t)$ for all $t \in \mathrm{C}$ in time $O(|S| \log |S|)$. In time $O(|S|)$ one then finds and returns $t^* = \arg\max_t G_{\mathrm{info}}(S, \sigma_t)$.

The case $\mathbb{1}_{x_j<t}$ is similar, see the proof of Lemma G.2. □

Lemma G.4. *Let $\mathcal{S}_j$ be the set of split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$ or $\sigma(x) = \mathbb{1}_{x_j=t}$. Then $\sigma^*_{\mathcal{S}_j}(S, G_{\mathrm{var}})$ can be computed in time $O(|S| \log |S|)$.*

Proof. The proof is similar to that of Lemma G.2. □

H. Proofs for Section 6

H.1. Proof of Theorem 6.1

We prove a more general statement for several gain measures. First, let us introduce the relevant definitions and notation in more detail.

The Orthogonal Vectors problem (OV) asks to decide whether a set $V \subseteq \{0,1\}^d$ contains two orthogonal vectors. The Orthogonal Vector Conjecture OVH states that the complexity of OV is at least $n^{2-o(1)}$ for all $d = \omega(\log n)$, where $n = |V|$. We show that, unless OVH fails, every fully dynamic algorithm for a certain *weighted* version of the dynamic exact approximate tree problem has amortized cost polynomial in n. In this weighted version, each example (x, y) is associated with a positive weight $w((x, y))$. When the weights are integers, an example (x, y) with weight $w((x, y))$ counts as $w((x, y))$ copies of (x, y). Our bound holds for common gain measures such as variance gain, Gini gain, information gain, and for labelling using the mean and/or returning the label distribution at leaves. The bound holds even when $X = Y = \{0, 1\}$ and the maximum weight of any example is in $O(n^\epsilon)$ for any sufficiently small $\epsilon > 0$.

Theorem H.1. *Let $X = \{0,1\}^d$ and $Y = \{0,1\}$ and $\epsilon \in (0, \frac{1}{3})$, and let $G \in \{G_{\mathrm{gini}}, G_{\mathrm{info}}, G_{\mathrm{var}}\}$. Let ρ be any construction rule based on a max-G rule with threshold 0 that labels by using the mean or the label distribution at the leaves. Unless OVH fails, no algorithm with amortized cost $O(n^\epsilon)$ maintains the tree of* GREEDY$_\rho$ *under weighted examples. This holds even in the case when all features and the class are binary and the maximum weight of any example is in $O(n^{3\epsilon})$.*

Proof. For the sake of presentation we consider the case when the maximum weight of any example is $n^{1+\epsilon}$. The maximum weight can be reduced to $n^{3\epsilon}$ by inserting $n^{1-2\epsilon}$ copies of the same example, each one with weight $n^{3\epsilon}$. Let $d = \Theta(\log^2 n)$, let $V \subseteq \{0,1\}^d$ be the OV instance, and let $n = |V|$. For every $u \in V$ let $\bar{u}$ be its complement (i.e. $\bar{u}_i = 1 - u_i$) and $A_u \subseteq [d]$ be the support of u. Moreover let $a = |A_u|$ and $N = n^{1+\epsilon}$. We let (x, y, w) denote a labeled example (x, y) with weight w. For every $u \in V$ and every $i \in A_u$, let $z^i \in \{0,1\}^d$ be the vector defined by $z^i_j = \mathbb{1}_{j\in A_u \setminus \{i\}}$ for all $j \in [d]$. Define the following (multi)sets of weighted labeled examples:

$$S = \{(\bar{u}, 0, 1), (\bar{u}, 1, 1) \,:\, u \in V\} \tag{164}$$

$$Q_u = \{(u, 1, N), \ldots, (u, 1, N)\} \cup \left\{(z^i, 0, N) \,:\, i \in A_u\right\}, \qquad \forall u \in V \tag{165}$$

where the example $(u, 1, N)$ appears $a = |A_u|$ times. Observe that $|Q_u| = 2|A_u|$. For any $A \subseteq [d]$ let

$$S(A) = \{(x, y) \in S : x_i = 1, \forall i \in A\} \tag{166}$$

$$S(A)_j = \{(x, y) \in S : x_i = 1, \forall i \in A, y = j\} \tag{167}$$

Observe that $|S(A)_0| = |S(A)_1|$ by construction of S.

Fix any $u \in V$, let $S' = S + Q_u$, and let $T = \mathrm{GREEDY}_\rho(S')$. We will show that T contains a leaf v such that $S'(T, v) =$

$S(A_u) \cup Q_u$. Note that $S(A_u) \neq 0$ if and only if V contains a vector orthogonal to u, and moreover $S(A_u) \neq 0$ if and only if the label histogram at v gives non-zero weight to 0. It is also immediate to see that v is the leaf reached by u in T. Therefore by querying T for u we will learn whether V contains a vector orthogonal to u.

To show the existence of the leaf v, we show that the gain of any feature in A_u is always strictly larger than that of any other feature in $[d]$. We will assume the Gini gain, but similar calculations work for the information gain and the variance gain, too. The idea is that the total weight of the "gadget" Q_u dominates the total weight of S by a factor of n^ϵ, and therefore the max-gain split will be the split that yields maximum gain over Q_u alone.

Let $S' = S \cup Q_u$ and let $B \subseteq A_u$. Let $k = |S'(B)_1| - |S'(B)_0|$. For the Gini gain by splitting on feature $i \in A_u \setminus B$, given that we have split on all features in B i.e. $G(S'(B), i)$ we obtain

$$
\begin{aligned}
&G(S'(B), i) && (168)\\
&\geq g(S'(B)) && (169)\\
&- \frac{N+n}{2aN-kN} \cdot \left(1 - \left(\frac{N+\frac{n}{2}}{N+n}\right)^2 - \left(\frac{\frac{n}{2}}{N+n}\right)^2\right)\\
&- \frac{2aN-(k+1)N+n}{2aN-kN}\left(1 - \left(\frac{aN+\frac{n}{2}}{2aN-(k+1)N+n}\right)^2\right.\\
&\qquad\left. - \left(\frac{aN-(k+1)N+\frac{n}{2}}{2aN-(k+1)N+n}\right)^2\right)\\
&\geq g(S'(B)) - o(1) && (170)\\
&- \frac{2aN-(k+1)N+n}{2aN-kN}\left(1 - \left(\frac{aN+\frac{n}{2}}{2aN-(k+1)N+n}\right)^2\right.\\
&\qquad\left. - \left(\frac{aN-(k+1)N+\frac{n}{2}}{2aN-(k+1)N+n}\right)^2\right)
\end{aligned}
$$

as $|S'(B \cup \{i\})_1| - |S'(B \cup \{i\})_0| = k+1$ and there is a fraction of $\frac{aN}{2aN-(k+1)N}$ examples in $Q_u(B \cup \{i\})$ labeled 1 and a fraction of $\frac{aN-(k+1)N}{2aN-(k+1)N}$ examples in $Q_u(B) \cup \{i\})$ labeled 0.

When splitting on feature $j \in [d] \setminus A_u$, we obtain:

$$
\begin{aligned}
&G(S'(B), j) && (171)\\
&\leq g(S'(B)) - \left(1 - \left(\frac{aN}{2aN-kN}\right)^2 - \left(\frac{aN-kN}{2aN-kN}\right)^2\right) && (172)\\
&= g(S'(B)) - \left(1 - \left(\frac{a}{2a-k}\right)^2 - \left(\frac{a-k}{2a-k}\right)^2\right) && (173)
\end{aligned}
$$

as $|S'(B \cup \{j\})_1| - |S'(B \cup \{j\})_0| = |S'(B)_1| - |S'(B)_0| = k$ and there is a fraction of $\frac{aN}{2aN-kN}$ examples in $Q_u(B \cup \{j\})$ labeled 1 and a fraction of $\frac{aN-kN}{2aN-kN}$ examples in $Q_u(B \cup \{j\})$ labeled 0.

Observe that the following inequality holds for any $0 < k \leq a$, $a > 1$,

$$
\begin{aligned}
&\frac{2a-(k+1)}{2a-k} \cdot \left(1 - \left(\frac{a}{2a-(k+1)}\right)^2 - \left(\frac{a-(k+1)}{2a-(k+1)}\right)^2\right)\\
&< 1 - \left(\frac{a}{2a-k}\right)^2 - \left(\frac{a-k}{2a-k}\right)^2. && (174)
\end{aligned}
$$

Hence, $G(S'(B), i) > G(S'(B), j)$ for n sufficiently large, for every $i \in A_u \setminus B$, $j \in [d] \setminus A_u$.

We can conclude our reduction. For every $\ell \in [d]$ we construct a decision tree T_ℓ of maximum height ℓ by inserting every element of S. Then, for every u in V and every $\ell \in [d]$:

- we insert all of Q_u in T_ℓ
- we query for $T_\ell(u)$; if the answer is a label distribution where 0 has positive weight, then we return "yes"
- we remove all of Q_u from T_ℓ.

If no "yes" was returned, then we return "no". By the observations above, this correctly solves OV over V.

Suppose every tree T_ℓ can be maintained in amortized time τ. As $|Q_u| = \tilde{O}(n^{1-2\epsilon})$, $|S| \in O(n)$, and $d \in O(\log^2 n)$, the total running time of the procedure above is in:

$$
\begin{aligned}
&O(n\tau \log^2 n) + O(n|Q_u|\tau \log^2 n) && (175)\\
&= O(n^{2-2\epsilon}\tau \log^2 n) && (176)\\
&= O(n^{2-2\epsilon+o(1)}\tau) && (177)
\end{aligned}
$$

Thus $\tau = O(n^\epsilon)$ implies OV can be solved in time $O(n^{2-\epsilon+o(1)})$ thereby falsifying OVH.

□

H.2. Proof of Theorem 6.2

The 3SUM conjecture states that the 3SUM problem (deciding if an array A of n integers contains three distinct elements that sum to 0) cannot be solved in time $n^{2-\Omega(1)}$. We reduce from the *range mode query* problem, which asks to maintain an n-entry vector of integers A and to answer queries in the form (i, j) with the most frequent element in the interval $A[i \dots j]$. It is known that, unless the 3SUM conjecture fails, there exists $c > 0$ such that any algorithm for the range mode query problem has update and/or query time $\Omega(n^c)$ (Patrascu, 2010).

Let $X = Y = \mathbb{R}$. Let $h \in \{2, 3, 4\}$ to be fixed later, let $G = G_{\text{var}}$, and let λ be the function that labels using a

mode (i.e., any most-frequent label). Let $\mathcal{S}$ be the family of all split rules in the form $\sigma(x) = \mathbb{1}_{x_j<t}$. Let T be the tree produced by GREEDY$_\rho$ where ρ is the construction rule that returns any split that maximizes G if such a maximum is positive and the depth of the current node is less than h, and that returns the label decided by λ otherwise.

The reduction is as follows. The tree T is initially empty. Upon receiving an update request $A[i] := v$, we first delete any previous example whose feature has value i from the current dataset (this can be done in time $O(\log n)$ by keeping a self-balancing tree that stores the current dataset using the feature value as a key). We then create the labeled example $(i, v) \in X \times Y$ and insert it into T. Moreover, we keep track of the maximum value of v in any update seen so far; let it be v^*. Upon receiving a query (i, j), we insert in T the labeled examples $s_i := (i - .5, C^2)$ and $s_j := (j + .5, C)$, where C is a sufficiently large power of n (say, n^{10}), which can be represented using $O(\log n)$ bits. Let us now analyze the decision taken by ρ at the root of T at this point.

Let S be the current dataset. We assume that all elements of S except for s_i have label 0; our conclusions hold even without this assumption, since what we actually need is just that C^2 dominates all other labels by sufficiently large $\mathrm{poly}(n)$ factors. Since h is positive, then ρ returns a split rule; let S_0, S_1 be the corresponding subsets and assume $s_i \in S_0$ (a symmetric argument applies otherwise). The conditional variance given by the split rule on S can then be easily calculated recalling that $\mathrm{var}(B(p)) = p(1-p)$ and that $\mathrm{var}(cX) = c^2 \mathrm{var}(X)$ for any random variable X:

$$\frac{|S_0|}{|S|} \cdot \frac{1}{|S_0|}\left(1 - \frac{1}{|S_0|}\right) C^4 + \frac{|S_1|}{|S|} \cdot 0 \tag{178}$$

$$= |S|^{-1}\left(1 - \frac{1}{|S_0|}\right) C^4 \tag{179}$$

The expression above is minimized when $|S_0|$ is minimized. Therefore S_0 must consist precisely of all elements in S with label at most $i + .5$, and S_1 of all elements of S with label at least $i+1$. If instead $s_i \in S_1$, a symmetric argument shows that S_0 consists precisely of all elements in S with label at most $i - 1$, and S_1 of all elements of S with label at least $i + .5$. In any case there exists a child v of the root of T such that $S(T, v)$ contains all examples of S with label at least i, including s_j (and possibly s_i) but no example of S with label smaller than i. One can prove that the argument holds even removing the assumption that all labels except that of s_i are 0: doing so would change (178) by at most $O(C^2/n^2)$, which leaves the optimal split unchanged.

Applying the same argument above to $S(T, v)$ and noting that $h \geq 2$ shows that v must have in turn a child w whose split rule isolates precisely s_i, if $s_i \in S(T, v)$, and which otherwise separates all examples with label at least $j+1$ (and possibly s_j, too). We conclude that, for some $h \in \{2, 3, 4\}$, T has a leaf z such that $S(T, z)$ consists precisely of the points with label between i and j. Querying the tree for the label of i then returns precisely a mode of $A[i \dots j]$, as desired. As we do not know the right h in advance, we maintain in parallel three distinct decision trees for $h = 2$, $h = 3$, and $h = 4$. It is easy to see that, at query time, one can choose the correct tree to be queried based solely on the values of i, j and n (and therefore in polylogarithmic time). After answering the query, we delete s_i and s_j from all trees. This implies that maintaining GREEDY$_\rho$ requires time n^c for some $c > 0$ unless 3SUM fails.