Electronic Colloquium on Computational Complexity, Report No. 166 (2019)

Top-down induction of decision trees: rigorous guarantees and inherent limitations

Guy Blanc Jane Lange Li-Yang Tan

Stanford University

November 20, 2019

Abstract

Consider the following heuristic for building a decision tree for a function $f : \{0,1\}^n \to \{\pm 1\}$. Place the *most influential variable* x_i of f at the root, and recurse on the subfunctions $f_{x_i=0}$ and $f_{x_i=1}$ on the left and right subtrees respectively; terminate once the tree is an ε-approximation of f. We analyze the quality of this heuristic, obtaining near-matching upper and lower bounds:

- *Upper bound:* For every f with decision tree size s and every $\varepsilon \in (0, \frac{1}{2})$, this heuristic builds a decision tree of size at most $s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}$.
- *Lower bound:* For every $\varepsilon \in (0, \frac{1}{2})$ and $s \leq 2^{\tilde{O}(\sqrt{n})}$, there is an f with decision tree size s such that this heuristic builds a decision tree of size $s^{\tilde{\Omega}(\log s)}$.

We also obtain upper and lower bounds for monotone functions: $s^{O(\sqrt{\log s}/\varepsilon)}$ and $s^{\tilde{\Omega}(\sqrt[4]{\log s})}$ respectively. The lower bound disproves conjectures of Fiat and Pechyony (2004) and Lee (2009).

Our upper bounds yield new algorithms for properly learning decision trees under the uniform distribution. We show that these algorithms—which are motivated by widely employed and empirically successful top-down decision tree learning heuristics such as ID3, C4.5, and CART—achieve provable guarantees that compare favorably with those of the current fastest algorithm (Ehrenfeucht and Haussler, 1989), and even have certain qualitative advantages. Our lower bounds shed new light on the limitations of these heuristics.

Finally, we revisit the classic work of Ehrenfeucht and Haussler. We extend it to give the first uniform-distribution proper learning algorithm that achieves polynomial sample and memory complexity, while matching its state-of-the-art quasipolynomial runtime.

ISSN 1433-8092

1 Introduction

Consider the problem of constructing a *decision tree* representation of a function $f : \{0,1\}^n \to \{\pm 1\}$, where the goal is to build a decision tree for f that is as small as possible, ideally of size close to the optimal decision tree size of f. Perhaps the simplest and most natural approach is to proceed in a *top-down*, greedy fashion:

1. Choose a "good" variable x_i to query as the root of the decision tree;
2. Build the left and right subtrees by recursing on the subfunctions $f_{x_i=0}$ and $f_{x_i=1}$ respectively.

This reduces the task of building a decision tree to that of choosing the root variable—i.e. determining the *splitting criterion* of this top-down heuristic. Intuitively, a good root variable should be one that is very "relevant" and "important" in terms of determining the value of f; it is reasonable to expect that querying such a variable first would reduce the number of subsequent queries necessary. Our focus in this paper will be on a specific splitting criterion: *influence*.

Definition 1 (Influence)**.** The *influence of the variable x_i on a function $f : \{0,1\}^n \to \{\pm 1\}$* is defined to be

$$\mathrm{Inf}_i(f) \coloneqq \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[f(\boldsymbol{x}) \neq f(\boldsymbol{x}^{\oplus i})],$$

where $\boldsymbol{x}$ is drawn uniformly at random, and $\boldsymbol{x}^{\oplus i}$ denotes $\boldsymbol{x}$ with its i-th coordinate flipped.

Influence is a fundamental and well-studied notion in the analysis of boolean functions [O'D14]. It is the key quantity of interest in many landmark results (e.g. the KKL inequality [KKL88], Friedgut's junta theorem [Fri98], the Invariance Principle [MOO10]) and open problems (e.g. the Gotsman–Linial conjecture [GL89], the Aaronson–Ambainis conjecture [AA14], the Fourier Entropy-Influence conjecture [FK96]) of the field. Beyond the analysis of boolean functions, this notion has been widely employed across both algorithms and complexity theory, where it has indeed proven to be a useful quantitative measure of the relevance and importance of a variable. Most relevant to the algorithmic applications in this paper, influence has been a key enabling ingredient in a large number of results in learning theory [BT96, BBL98, LMN93, Ser04, OS07, OW13, GS10, DHK+10, Kan14a, Kan14b, BCO+15].

1.1 Influence as a splitting criterion

We now give a formal description of the heuristic for constructing decision trees that we study. We define a *bare tree* to be a decision tree with unlabeled leaves, and write T° to denote such trees. We refer to any decision tree T obtained from T° by a labelling of its leaves as a *completion* of T°. Given a bare tree T° and a function f, there is a canonical completion of T° that minimizes the approximation error with respect to f:

Definition 2 (f-completion of a bare tree)**.** Let T° be a bare tree and $f : \{0,1\}^n \to \{\pm 1\}$. Consider the following completion of T°: for every leaf ℓ in T°, label it $\mathrm{sign}(\mathbb{E}[f_\ell(\boldsymbol{x})])$, where f_ℓ is the restriction of f by the path leading to ℓ and $\boldsymbol{x} \sim \{0,1\}^n$ is uniform random. This completion minimizes the approximation error $\Pr[T(\boldsymbol{x}) \neq f(\boldsymbol{x})]$, and we refer to it as the *f-completion of T°*.

In addition to the function f, our heuristic will also take in an error parameter ε, allowing us to construct both *exact* ($\varepsilon = 0$) and *approximate* ($\varepsilon \in (0, \frac{1}{2})$) decision tree representations of f.

BuildTopDownDT(f, ε):

Initialize T° to be the empty tree.

while (f-completion of T° is not an ε-approximation of f) {

1. (Score) For every leaf ℓ in T°, let $x_{i(\ell)}$ denote the most influential variable of the subfunction f_ℓ:
$$\mathrm{Inf}_{i(\ell)}(f_\ell) \geq \mathrm{Inf}_j(f_\ell) \quad \text{for all } j \in [n].$$
Assign ℓ the score:
$$\mathrm{score}(\ell) := \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[\,\boldsymbol{x} \text{ reaches } \ell\,] \cdot \mathrm{Inf}_{i(\ell)}(f_\ell) = 2^{-|\ell|} \cdot \mathrm{Inf}_{i(\ell)}(f_\ell),$$
where $|\ell|$ denotes the depth of ℓ in T°.
2. (Split) Let $\ell^\star$ be the leaf with the highest score. Grow T° by replacing $\ell^\star$ with a query to $x_{i(\ell^\star)}$.

}

Figure 1: Top-down heuristic for building an ε-approximate decision tree representation of f, with influence as the splitting criterion.

In words, BuildTopDownDT builds a bare tree T° in a top-down fashion, starting from the empty tree. In each iteration, we first check if the f-completion of T° is an ε-approximation of f, and if so, we output the completion. Otherwise, we split the leaf $\ell^\star$ with the highest score by querying the most influential variable of $f_{\ell^\star}$, where the score of a leaf ℓ is the influence of the most influential variable of f_ℓ normalized by the depth of ℓ within T°.[1]

1.2 This work

By design, the decision tree returned by BuildTopDownDT(f, ε) is an ε-approximation of f. We write TopDownDTSize(f, ε) to denote the size of this tree, and when $\varepsilon = 0$, we simply write TopDownDTSize(f). The question that motivates our work is:

What guarantees can we make on TopDownDTSize(f, ε) *as a function of the optimal decision tree size of f and ε?*

That is, we would like to understand the quality of BuildTopDownDT as a heuristic for constructing exact and approximate decision tree representations. In addition to being a natural

[1] There are two possibilities for ties in BuildTopDownDT: two variables may have the same influence within a subfunction f_ℓ, and two leaves may have the same score. Our upper bounds hold regardless of how ties are broken, and our lower bounds hold even if ties are broken in the most favorable way.

structural question concerning decision trees, this question also has implications in learning theory. Indeed, BuildTopDownDT is motivated by top-down decision tree learning heuristics such as ID3, C4.5, and CART that are widely employed and empirically successful in machine learning practice. We discuss the learning-theoretic context and applications of our structural results in Section 2.1, and the connection to practical machine learning heuristics in Section 3.1.

To our knowledge, the question above has not been studied in such generality. The most directly relevant prior work is that of Fiat and Pechyony [FP04], who considered the case when f is either a linear threshold function or a read-once DNF formula, and the setting of exact representation ($\varepsilon = 0$). For such functions, they proved that the heuristic builds an exact decision tree representation of optimal size. We give an overview of other related work in Section 3.2.

2 Our results

As our first contribution, we give near-matching upper and lower bounds that provide a fairly complete answer to the question above. Our upper bound is as follows:

Theorem 3 (Upper bound for approximate representation)**.** *For every $\varepsilon \in (0, \frac{1}{2})$ and every size-s decision tree f, we have* $\textsc{TopDownDTSize}(f, \varepsilon) \leq s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}$.

We complement Theorem 3 with lower bounds showing that (a) for exact representation ($\varepsilon = 0$), no non-trivial upper bound can be obtained; and (b) for approximate representation ($\varepsilon \in (0, \frac{1}{2})$), the dependence on s in Theorem 3 is essentially optimal:

Theorem 4 (Lower bounds for exact and approximate representations)**.**

(a) Exact representation: *There is an $f : \{0,1\}^n \to \{\pm 1\}$ with decision tree size $s = \Theta(n)$ such that* $\textsc{TopDownDTSize}(f) \geq 2^{\Omega(s)}$.

(b) Approximate representation: *For every $\varepsilon \in (0, \frac{1}{2})$ and function $s(n) \leq 2^{\tilde{O}(\sqrt{n})}$, there is an $f : \{0,1\}^n \to \{\pm 1\}$ with decision tree size s such that* $\textsc{TopDownDTSize}(f, \varepsilon) \geq s^{\tilde{\Omega}(\log s)}$.

Prior to our work, it was not known whether an upper bound of $\textsc{TopDownDTSize}(f, \varepsilon) \leq \mathrm{poly}(s, 1/\varepsilon)$ holds for all size-s decision trees f and $\varepsilon \in (0, \frac{1}{2})$; Theorem 4(b) provides a strong negative answer. Indeed, such an upper bound had been conjectured to hold for the class of *monotone* functions [Lee09]. We now discuss our results on monotone functions, which disprove this conjecture, along with a stronger variant of it for exact representation [FP04].

Monotone functions. A monotone boolean function $f : \{0,1\}^n \to \{\pm 1\}$ is one that satisfies $f(x) \leq f(y)$ for all $x \preceq y$ (where $x \preceq y$ iff $x_i \leq y_i$ for all $i \in [n]$). An elementary and useful fact about monotone functions is that the influence of a variable on a monotone function f is equivalent to its *correlation* with f:

Fact 2.1 (Influence $\equiv$ correlation for monotone functions)**.** *For all monotone functions $f : \{0,1\}^n \to \{\pm 1\}$ and $i \in [n]$, we have* $\mathrm{Inf}_i[f] = 2\,\mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_i] - \mathbb{E}[f(\boldsymbol{x})]$.[2]

[2]The equivalence between influence and correlation for monotone functions is more transparent if one works with $\{\pm 1\}^n$ instead of $\{0,1\}^n$ as the domain: for monotone functions $f : \{\pm 1\}^n \to \{\pm 1\}$, we have $\mathrm{Inf}_i[f] = \mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_i]$.

Therefore, for monotone functions, splitting on the most influential variable of a subfunction is equivalent to splitting on the variable that has the *highest correlation* with the subfunction.[3]

Our proof of Theorem 3 extends in a straightforward manner to give a different upper bound under the assumption of monotonicity, where the dependence on s is significantly better. We refer to a size-s decision tree computing a monotone function as a size-s monotone decision tree.

Theorem 5 (Upper bound for approximate representation of monotone functions.)**.** *For every $\varepsilon \in (0, \frac{1}{2})$ and every size-s monotone decision tree f, we have* $\textsc{TopDownDTSize}(f, \varepsilon) \leq s^{O(\sqrt{\log s}/\varepsilon)}$.

In analogy with Theorem 4, we also obtain lower bounds for exact and approximate representations of monotone functions:

Theorem 6 (Lower bounds for exact and approximate representations of monotone functions)**.**

(a) Exact representation: *There is a monotone $f : \{0,1\}^n \to \{\pm 1\}$ with decision tree size $s = \Theta(n)$ such that* $\textsc{TopDownDTSize}(f) \geq 2^{\Omega(s)}$.

(b) Approximate representation: *For every $\varepsilon \in (0, \frac{1}{2})$ and function $s(n) \leq 2^{\tilde{O}(n^{4/5})}$, there is an $f : \{0,1\}^n \to \{\pm 1\}$ with decision tree size s such that* $\textsc{TopDownDTSize}(f, \varepsilon) \geq s^{\tilde{\Omega}(\sqrt[4]{\log s})}$.

Although we have stated Theorem 6 in terms of the specific heuristic BuildTopDownDT that we study, the actual lower bounds that we establish are significantly stronger: they apply to all "impurity-based top-down heuristics". This is a broad class that captures a wide variety of decision tree learning heuristics used in machine learning practice, including ID3, C4.5, and CART; see Section 3.1 for details.

Theorem 7 (Stengthening of Theorem 6(b))**.** *For every $\varepsilon \in (0, \frac{1}{2})$ and function $s(n) \leq 2^{\tilde{O}(n^{4/5})}$, there is a size-$s$ monotone decision tree f such that the ε-approximator built by any impurity-based top-down heuristic must have size $s^{\tilde{\Omega}(\sqrt[4]{\log s})}$.*

Disproving conjectures of Fiat–Pechyony and Lee. Motivated by applications in learning theory (discussed next in Section 2.1), Fiat and Pechyony [FP04] and Lee [Lee09] also considered the quality of BuildTopDownDT as a heuristic for building decision trees for monotone functions.

[FP04] conjectured that for all monotone functions f, even in the case of exact representation ($\varepsilon = 0$), BuildTopDownDT returns a tree of minimal depth and size "not far from minimal." Theorem 6(a) provides a counterexample to the conjectured bound on size, and the function in Theorem 6(b) disproves the conjecture about depth; see Remark 22.[4]

Stated in the notation of our paper, [Lee09] raised the possibility that $\textsc{TopDownDTSize}(f, \varepsilon) \leq \mathrm{poly}(s, 1/\varepsilon)$ for all size-s monotone decision trees f and $\varepsilon \in (0, \frac{1}{2})$. The author further remarked that "showing $\textsc{TopDownDTSize}(f, \varepsilon) \leq \mathrm{poly}(s)$, even only for constant accuracy ε,[5] would be a huge advance". Theorem 6(b) rules this out.

[3] We observe that for general *non-monotone* functions, correlation can in general be a very poor splitting criterion, in the sense of building a decision tree that is much larger than the optimal decision tree. Consider $f : \{0,1\}^n \to \{\pm 1\}$ where $f(x) = x_j \oplus x_k$, the parity of two variables. The optimal decision tree size of f is 4, but since $\mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_i] = 0$ for all $i \in [n]$, the top-down heuristic using correlation as its splitting criterion may build a tree of size $\Omega(2^n)$ before achieving any non-trivial accuracy $\varepsilon < \frac{1}{2}$. (On the other hand, the top-down heuristic using influence as its splitting criterion would build the optimal tree of size 4.) We revisit this observation in Section 3.1.

[4] For clarity of exposition, throughout this overview we discuss our results with decision tree *size* as the complexity measure. There are analogues of all of our results, both upper and lower bounds, for decision tree *depth* as the complexity measure.

[5] That is, a bound of the form $\textsc{TopDownDTSize}(f, \varepsilon) \leq s^{O_\varepsilon(1)}$.

2.1 Algorithmic applications: Properly learning decision trees

Learning decision trees has been a touchstone problem in uniform-distribution PAC learning for more than thirty years. It sits right at the boundary of our understanding of efficient learnability, and continues to be the subject of intensive research. The seminal work of Ehrenfeucht and Haussler [EH89] gave a $\mathrm{poly}(n^{\log s}, 1/\varepsilon)$-time algorithm for learning decision trees using random examples (see also [Blu92] for an alternative proof based on Rivest's algorithm for learning decision lists [Riv87]);[6] subsequently, Linial, Mansour, and Nisan [LMN93] gave an algorithm that also runs in quasipolynomial time, but achieves polynomial sample complexity; Kusilevitz and Mansour [KM93], leveraging a novel connection to cryptography [GL89], gave a polynomial-time algorithm using membership queries; Gopalan, Kalai, and Klivans [GKK08] obtained an *agnostic* analogue of [KM93]'s algorithm, extending it to tolerate adversarial noise; O'Donnell and Servedio [OS07] gave a polynomial-time algorithm for learning *monotone* decision trees from random examples; recent work of Hazan, Klivans, and Yuan [HKY18] gives an algorithm agnostically learning decision trees with *polynomial sample complexity*; even more recent work of Chen and Moitra [CM19] gives an algorithm for learning *stochastic* decision trees.

Properly learning decision trees. When learning decision trees, it is natural to seek a hypothesis that is itself a decision tree. Indeed, it may be natural to seek a decision tree hypothesis even when learning other concept classes. The simple structure of decision trees makes them desirable both in terms of interpretability and explanatory power, which is why they are ubiquitous in empirical machine learning. A further advantage of decision tree hypotheses is that they are very fast to evaluate: evaluating a depth-d decision tree on a given input takes time $O(d)$,[7] whereas evaluating say a degree-d polynomial—another canonical and ubiquitous representation class in learning theory—can take time $\Theta(n^d)$, the number of monomials in the polynomial.

In learning theory, algorithms that return a hypothesis belonging to the concept class are known as *proper*. Understanding the complexity of proper learning (vis-à-vis improper learning) is an important research direction in learning theory [Fel16]; proper learning also has deep connections to proof complexity [ABF+09] and property testing [GGR98].

2.1.1 New proper learning algorithms

Among the decision tree learning algorithms discussed at the beginning of this subsection, the only one that is proper is the one of Ehrenfeucht and Haussler [EH89]. Our upper bounds on TopDownDTSize yield new algorithms for properly learning decision trees under the uniform distribution:

Theorem 8 (Algorithmic consequence of Theorem 3)**.** *Size-s decision trees can be properly learned under the uniform distribution in time* $\mathrm{poly}(n, s^{\log(s/\varepsilon)\log(1/\varepsilon)})$ *using membership queries.*[8]

[6] In fact, the algorithm of [EH89] learns decision trees in the more challenging setting of *distribution-free* PAC learning. All other results in this discussed in this section, including ours, are specific to uniform-distribution learning, and we focus our exposition on this setting.

[7] Every size-s decision tree is well-approximated by a decision tree of depth $O(\log s)$.

[8] We remark that our algorithm only requires fairly "mild" use of membership queries. Our algorithm only requires *random edge samples* (Definition 24), and hence falls within both the random walk model of Bshouty et al. [BMOS05] and the local membership queries model of Awasthi et al. [AFK13]. These (incomparable) models are natural relaxations of the standard model of learning from random examples, and do not allow the learning algorithm unrestricted membership query access to the target function.

Analogously, Theorem 5 yields a new algorithm for learning monotone decision trees using only random examples. The learnability of monotone functions with respect to various complexity measures has been the subject of intensive study in uniform-distribution learning [HM91, KV94, KLV94, Bsh95, BT96, BBL98, Ver98, SM00, Ser04, OS07, Sel08, DSLM+09, Lee09, JLSW11, OW13].

Theorem 9 (Algorithmic consequence of Theorems 3 and 5)**.** *Size-s monotone decision trees can be properly learned under the uniform distribution in time* $\mathrm{poly}(n, \min(s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}, s^{O(\sqrt{\log s}/\varepsilon)}))$ *using only random examples.*

We now compare our results with the prior state of the art for properly learning decision trees.

- **Polynomial-time algorithms for superlogarithmic size.** Theorems 8 and 9 give the first polynomial-time algorithms for properly learning decision trees of size $\omega(\log n)$ to constant accuracy. To see this, we first note that [EH89]'s runtime of $\mathrm{poly}(n^{\log s}, 1/\varepsilon)$ is superpolynomial time for any $s = \omega(1)$. Alternatively, functions depending on $k \ll n$ variables ("k-juntas") can be properly learned in time $\mathrm{poly}(n, 2^k)$, using random examples for monotone juntas, and membership queries otherwise [BL97, MOS04]. Since every size-s decision tree certainly depends on at most $k \leq s$ variables, this runtime is polynomial for decision trees of size $s = O(\log n)$, but becomes superpolynomial once $s = \omega(\log n)$. In contrast, the runtimes of our algorithms in Theorems 8 and 9 remain polynomial for $s = 2^{\Omega(\sqrt{\log n})}$ and $s = 2^{\Omega((\log n)^{2/3})}$ respectively.

- **Dimension-independent hypothesis size.** Related to the above, the sizes of the hypotheses returned by the algorithms of Theorems 8 and 9 are $s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}$ and $s^{O(\sqrt{\log s}/\varepsilon)}$ respectively, independent of n, whereas the size of the hypotheses returned by [EH89]'s algorithm can be as large as $n^{\Omega(\log s)}$. This is gap can be exponential or even larger for small values of s.

- **Average depth as the complexity measure.** Our algorithms and analyses extend easily to accommodate *average depth* as the complexity measure. The average depth of a decision tree, $\triangle(T)$, is the number of queries T makes on a uniform random input. Average depth is a stronger complexity measure than size since $\triangle(T) \leq \log(\mathrm{size}(T))$.[9]

  Theorem 10 (Learning trees with small average depth)**.** *Decision trees of average depth $\triangle$ can be properly learned under the uniform distribution in time* $\mathrm{poly}(n, 2^{\triangle^2/\varepsilon})$ *using membership queries, and monotone decision trees of average depth $\triangle$ can be properly learned in time* $\mathrm{poly}(n, 2^{\triangle^{3/2}/\varepsilon})$ *using random examples.*

  To our knowledge, these represent the first polynomial-time algorithms for properly learning decision trees of superconstant average depth, $\triangle = \omega(1)$. Prior to our work, the fastest algorithm ran in time $\mathrm{poly}(n^{\triangle/\varepsilon})$; this algorithm, which uses random examples, follows implicitly from the results of Mehta and Raghavan [MR02].

2.2 Proper learning with polynomial sample and memory complexity

For our final contribution, we revisit the classic algorithm of Ehrenfeucht and Haussler [EH89]. As discussed above, this remains the fastest algorithm for properly learning decision trees. We extend

[9] Furthermore, it is easy to construct examples of decision trees T with the largest possible gap between these measures: $\triangle(T) = O(1)$ and $\log(\mathrm{size}(T)) = \Omega(n)$.

it to give the first uniform-distribution proper algorithm that achieves polynomial sample and memory complexity, while matching its state-of-the-art quasipolynomial runtime (Theorem 29).

Reference	Running time	Sample complexity	Memory complexity	Proper?
[EH89]	$\mathrm{poly}(n^{\log s}, 1/\varepsilon)$	$\mathrm{poly}(n^{\log s}, 1/\varepsilon)$	$\mathrm{poly}(n^{\log s}, 1/\varepsilon)$	✓
[LMN93]	$\mathrm{poly}(n^{\log(s/\varepsilon)})$	$\mathrm{poly}(s, 1/\varepsilon) \cdot \log n$	$\mathrm{poly}(n, s, 1/\varepsilon)$	×
[MR02]	$\mathrm{poly}(n^{\log(s/\varepsilon)})$	$\mathrm{poly}(s, 1/\varepsilon) \cdot \log n$	$\mathrm{poly}(n^{\log(s/\varepsilon)})$	✓
This work	$\mathrm{poly}(n^{\log s}, 1/\varepsilon)$	$\mathrm{poly}(s, 1/\varepsilon) \cdot \log n$	$\mathrm{poly}(n, s, 1/\varepsilon)$	✓

Table 1: Algorithms for learning size-s decision trees from random examples under the uniform distribution

Ehrenfeucht and Haussler had posed (as the first open problem of their paper) the question of achieving polynomial sample complexity. Such algorithms were subsequently obtained by Linial, Mansour, and Nisan [LMN93] and Mehta and Raghavan [MR02]. Interestingly, these two algorithms are very different from each other and from [EH89]: the algorithm of [LMN93], being Fourier-based, is non-proper, whereas the algorithm of [MR02], which uses dynamic programming, has a large memory footprint. Furthermore, both algorithms have a quasipolynomial dependence on $1/\varepsilon$ in their runtimes, rather than [EH89]'s polynomial dependence.

This state of affairs raises the question of whether there is a *single* algorithm that achieves "the best of [EH89], [LMN93], and [MR02]" in each of the four metrics discussed above; see Table 1. We give such an algorithm in this work (Theorem 29). Our algorithm is a surprisingly simple modification of [EH89]'s algorithm, but our analysis is more involved. At a high level, the idea is to terminate [EH89]'s algorithm early to achieve our improved sample and memory complexity. However, incorporating this plan with the inherently bottom-up nature of [EH89]'s algorithm necessitates a delicate error analysis. (In particular, [EH89]'s algorithm is an Occam algorithm, whereas ours is not.)[10][11]

We remark that there is an ongoing flurry of research activity on the memory complexity of learning basic concept classes under the uniform distribution, with a specific focus on tradeoffs between memory and sample complexity [Sha14, SVW16, Raz17, KRT17, MM17, Raz18, MM18, BOGY18, GRT18, GRT19].

[10] Although our algorithm, like the others in Table 1, only uses random examples, to our knowledge there are no known membership query algorithms that achieves our guarantees.

[11] We note that it is possible to combine the ideas in [EH89] and [MR02] to give an algorithm that runs in $\mathrm{poly}(n^{\log(s/\varepsilon)})$ time and has sample and memory complexity $\mathrm{poly}(s, 1/\varepsilon) \cdot \log n$ and $\mathrm{poly}(n, s, 1/\varepsilon)$ respectively. We do not provide the details in this paper since our main result (Theorem 29) achieves strictly better guarantees.

3 Discussion and related work

3.1 Relationship to practical machine learning heuristics

Our work is motivated in part by the tremendous popularity and empirical success of top-down decision tree learning heuristics in machine learning practice, such as ID3 [Qui86], its successor C4.5 [Qui93], and CART [Bre17]. The data mining textbook [WFHP16] describes C4.5 as "a landmark decision tree program that is probably the machine learning workhorse most widely used in practice to date". In a similar vein, quoting Kearns and Mansour [KM99], "In experimental and applied machine learning work, it is hard to exaggerate the influence of top-down heuristics for building a decision tree from labeled sample data [...] Dozens of papers describing experiments and applications involving top-down decision tree learning algorithms appear in the machine learning literature each year".

We give a high-level description of how these heuristics work, using the framework of uniform-distribution learning. As we will soon see, they serve as motivation for the heuristic that we study, BuildTopDownDT (Figure 1). These heuristics grow a bare tree T° for a function $f : \{0,1\}^n \to \{0,1\}$ as follows. Consider the progress measure

$$\mathcal{H}(T^\circ) := \sum_{\ell \in \text{leaves}(T^\circ)} \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[\,\boldsymbol{x} \text{ reaches } \ell\,] \cdot \mathscr{G}(\mathbb{E}[f_\ell]),$$

where $\mathscr{G} : [0,1] \to [0,1]$ is known as the *impurity function*, and encapsulates the splitting criterion of the heuristic. This carefully chosen function is restricted to be concave, symmetric around $\frac{1}{2}$, and to satisfy $\mathscr{G}(0) = \mathscr{G}(1) = 0$ and $\mathscr{G}(\frac{1}{2}) = 1$. For example, $\mathscr{G}$ is the binary entropy function in ID3 and C4.5; CART uses $\mathscr{G}(p) = 4p(1-p)$, known as the *Gini criterion*; [KM99] studies the variant $\mathscr{G}(p) = 2\sqrt{p(1-p)}$.[12] Writing $T^\circ_{\ell,i}$ to denote T° with its leaf ℓ replaced with a query to the variable x_i, these heuristics, in a single iteration, grow T° to $T^\circ_{\ell^\star,i^\star}$, where

$$(\ell^\star, i^\star) \text{ is the leaf-variable pair that maximizes } \mathcal{H}(T^\circ) - \mathcal{H}(T^\circ_{\ell^\star,i^\star}). \tag{1}$$

We refer to any such top-down heuristic as an *impurity-based* heuristic, and the progress measure $\mathcal{H}(T^\circ) - \mathcal{H}(T^\circ_{\ell^\star,i^\star})$ as the *purity gain*.

Inherent limitations of impurity-based heuristics. It is easy to see (and has been well known [Kea96]) that impurity-based heuristics can, in general, fare very badly, in the sense of building a decision tree that is much larger than the optimal decision tree. For example, consider $f(x) = x_j \oplus x_k$ for $j, k \in [n]$, the parity of two variables. For such a target function, *regardless of the choice of the impurity function* $\mathscr{G}$, splitting on any of the n variables results in zero purity gain. This is because $\mathbb{E}[f] = \mathbb{E}[f_{x_i=b}]$ for all $i \in [n]$ and $b \in \{0,1\}$. Therefore, *any* impurity-based heuristic may build a tree of size $\Omega(2^n)$ before achieving any non-trivial error $\varepsilon < \frac{1}{2}$, whereas the size of the optimal tree of f is only 4.

One could exclude such "parity-like" examples by considering only *monotone* functions. Monotonicity is a ubiquitous condition in machine learning since many data sets are naturally monotone in their attributes. In the case of monotone functions, it can be shown that for *any impurity function* $\mathscr{G}$, the variable split that results in the most progress in the sense of (1), i.e. the variable x_i

[12] The work of Dietterich, Kearns, and Mansour [DKM96] gives a detailed experimental comparison of various impurity functions.

that maximizes the purity gain

$$\mathscr{G}(\mathbb{E}[f]) - \tfrac{1}{2}(\mathscr{G}(\mathbb{E}[f_{x_i=0}]) + \mathscr{G}(\mathbb{E}[f_{x_i=1}])),$$

is precisely the most influential variable of f (we prove this in Section 7; see Proposition 7.7). In other words, in the case of monotone functions, BuildTopDownDT closely models impurity-based heuristics. The works of Fiat and Pechyony [FP04] and Lee [Lee09] (recall our discussion following Theorem 7) were explicitly motivated by this observation, as are our results on monotone functions (Theorems 5 to 7 and 9).

As we will show, our monotone lower bounds for BuildTopDownDT actually apply to all impurity-based heuristics (Theorem 7), regardless of the choice of the impurity function $\mathscr{G}$ (hence including ID3, C4.5, and CART).[13] Since one could argue that real-world data sets are unlikely to be "parity-like", we view our monotone lower bounds as providing more robust (albeit still only theoretical) evidence of the limitations and potential shortcomings of the impurity-based top-down heuristics used in practice.

Top-down versus bottom-up: from practice to theory? We find it especially intriguing that the algorithm of Ehrenfeucht and Haussler [EH89]—which as discussed, remains the fastest algorithm for properly learning decision trees with provable runtime guarantees—builds its hypothesis tree *bottom up*, in exactly the *opposite* order from the top-down heuristics used in practice. It is natural to ask if top-down heuristics can serve as inspiration for the design and analyses of fundamentally different algorithms for properly learning decision trees.

Our algorithmic upper bounds for BuildTopDownDT (Theorems 8 and 9) provide affirmative answers, and as discussed above, these new algorithms even have certain qualitative advantages over [EH89]. Our lower bounds (Theorems 4 and 6), on the other hand, establish their inherent limitations. They imply that BuildTopDownDT is provably *not* a polynomial-time algorithm for properly learning decision trees using membership queries, or a polynomial-time algorithm for properly learning monotone decision trees using random examples. Either of these results would constitute a major advance in learning theory, and BuildTopDownDT—and other impurity-based variants of it—had been a natural candidate for obtaining them. Indeed, the results of [Lee09] were explicitly motivated by the goal of showing that BuildTopDownDT *is* a polynomial-time algorithm for properly learning monotone decision trees. This is now ruled out by Theorems 6 and 7.[14]

3.2 Related work

Fiat and Pechyony [FP04] considered linear threshold functions and read-once DNF formulas, and showed that BuildTopDownDT, when run on such functions, returns a decision tree of optimal size computing them exactly. (Stated in the notation of Theorem 3, $\textsc{TopDownDTSize}(f) = s$ for such functions.)

Kearns and Mansour [KM99] (see also [Kea96, DKM96]) showed that impurity-based heuristics are *boosting algorithms*, where one views the functions labeling internal nodes of the tree (single variables in our case) as weak learners. At a high level, the proofs of our upper bounds (Theorems 3

[13] Different impurity functions $\mathscr{G}$ lead to different *orderings* of leaves to split, and hence result in different trees.

[14] Blum et al. [BFJ+94] gave an information-theoretic lower bound showing that no "statistical query" algorithm can learn decision trees in polynomial time. However, this lower bound does not apply when membership queries are allowed or when the function is assumed to be monotone.

and 5) are similar in spirit to their analysis, in the sense that they are all incremental in nature, showing that each split contributes to the accuracy of the decision tree hypothesis. However, our results and analyses are incomparable—for example, [KM99] does not relate the size of the resulting hypothesis to the size of the optimal decision tree; [KM99]'s analysis assumes the existence of weak learners for all filtered-and-rebalanced versions of the target distribution, whereas we carry out the entirety of our analyses with respect to the uniform distribution.[15]

Recent work of Brutzkus, Daniely, and Malach [BDM19b] studies a variant of ID3 proposed by [KM99], focusing on learning conjunctions and read-once DNF formulas under product distributions. They provide theoretical and empirical evidence showing that for such functions, the size-t tree grown by [KM99]'s variant of ID3 achieves optimal or near-optimal error among all trees of size t. Concurrent work by the same authors [BDM19a] shows that ID3 efficiently learns $(\log n)$-juntas in the setting of smoothed analysis.

4 Preliminaries

Throughout this paper, we use bold font (e.g. $\boldsymbol{x}$ and $\boldsymbol{S}$) to denote random variables; all probabilities and expectations are with respect to the uniform distribution unless otherwise stated.

For any decision tree T, we say the *size* of T is the number of leaves in T, and the *depth* of T is length of the longest path between the root and a leaf. If a tree has size 1, then it contains a single leaf, computes either the constant $+1$ or constant -1 function, and has depth 0. For a function $f : \{0,1\}^n \to \{\pm 1\}$, the *optimal decision tree size* of f is the smallest s for which there exists a decision tree of size s that exactly computes f, and we write $\mathrm{size}(f)$ to denote this quantity. If T is a decision tree that computes f, then we will often use T interchangeably with f.

Choose any $f, g : \{0,1\}^n \to \{\pm 1\}$. Then, the *error* is defined as

$$\mathrm{error}(f,g) = \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})].$$

We say that f is an ε-approximation of g if $\mathrm{error}(f,g) \leq \varepsilon$. If T° is a bare tree, then $\mathrm{error}(T^\circ, f)$ is shorthand for $\mathrm{error}(T,f)$ where T is the f-completion of T°. We also use the following shorthand.

$$\mathrm{error}(f, \pm 1) = \min(\mathrm{error}(f,-1), \mathrm{error}(f,1)).$$

The *variance* of $f : \{0,1\}^n \to \{\pm 1\}$, denoted $\mathrm{Var}(f)$, is

$$\mathrm{Var}(f) = 4 \cdot \Pr[f(\boldsymbol{x}) = -1] \cdot \Pr[f(\boldsymbol{x}) = 1].$$

The *total influence* of f, denoted $\mathrm{Inf}(f)$, is

$$\mathrm{Inf}(f) = \sum_{i=1}^{n} \mathrm{Inf}_i(f).$$

It is easy to see that for any decision tree $T : \{0,1\}^n \to \{\pm 1\}$,

$$\mathrm{error}(T, \pm 1) \leq \mathrm{Inf}(T)$$

and

$$\frac{\mathrm{Var}(T)}{2} \leq \mathrm{error}(T, \pm 1) \leq \mathrm{Var}(T)$$

always hold.

[15] Indeed, [KM99]'s results concern impurity-based heuristics, and as discussed above, statements like Theorem 3 that apply to all functions cannot hold for such heuristics because of parity-like functions.

5 Upper bounds on TopDownDTSize: Proofs of Theorems 3 and 5

Recall that BuildTopDownDT(f, ε) continually grows a bare tree, T°, until the f-completion of T° is an ε-approximation of f. At a high level, the proofs of our upper bounds on TopDownDTSize proceed as follows.

Section 5.1 We define a progress metric, the "cost" of T°, which upper bounds the error of the f-completion of T° with respect to f. Hence, when the "cost" drops below ε, BuildTopDownDT can terminate. We show that whenever BuildTopDownDT grows T°, the "cost" of T° decreases by exactly the score of the leaf selected.

Section 5.2 We lower bound the score of the leaf that BuildTopDownDT selects.

Section 5.3 We put the above together to prove upper bounds on TopDownDTSize. At each step, the "cost" of T° must decrease by at least the lower bounds in Section 5.2, which allows us to upper bound the number of steps until the "cost" falls below ε. This is sufficient since the size of the tree that BuildTopDownDT produces is exactly one more than the number of steps it takes.

5.1 Definition and properties of "Cost"

Definition 11 (Cost of a bare tree)**.** Let $f : \{0,1\}^n \to \{\pm 1\}$ be a function and T° be a bare tree. Then the *cost of T° relative to f* is defined as

$$\mathrm{cost}_f(T^\circ) = \sum_{\text{leaf } \ell \in T^\circ} 2^{-|\ell|} \cdot \mathrm{Inf}(f_\ell).$$

This cost function is useful to track because it naturally decreases during BuildTopDownDT and upper bounds the error of the completion.

Lemma 5.1 (Properties of cost of a bare tree)**.** *For any $f : \{0,1\}^n \to \{\pm 1\}$ and bare tree T°, the following hold:*

1. $\mathrm{error}(T^\circ, f) \le \mathrm{cost}_f(T^\circ)$.
2. *Choose any leaf ℓ of T° and variable x_i. Let $(T^\circ)'$ be the bare tree that results from replacing ℓ in T° with a query to x_i. Then,*

$$\mathrm{cost}_f((T^\circ)') = \mathrm{cost}_f(T^\circ) - 2^{-|\ell|} \cdot \mathrm{Inf}_i(f_\ell).$$

At each step, BuildTopDownDT splits the leaf with the largest score, resulting in the cost decreasing by exactly the score selected. Once the cost decreases to below ε, we know the completion of T° is an ε-approximation of f, meaning BuildTopDownDT can terminate.

Proof. The proof of (1) is a simple application of the fact that $\mathrm{error}(g, \pm 1) \le \mathrm{Inf}(g)$ for any boolean function g:

$$\begin{aligned}
\mathrm{error}(T^\circ, f) &= \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[(\text{Completion of } T^\circ)(\boldsymbol{x}) \neq f(\boldsymbol{x})] \\
&= \sum_{\text{leaf } \ell \in T^\circ} \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[\boldsymbol{x} \text{ reaches } \ell] \cdot \mathrm{error}(f_\ell, , \pm 1) \\
&\le \sum_{\text{leaf } \ell \in T^\circ} 2^{-|\ell|} \cdot \mathrm{Inf}_i(f_\ell) = \mathrm{cost}_f(T^\circ).
\end{aligned}$$

The proof of (2) follows from the fact that if T is a tree with x_i at the root, T_0 as its 0-subtree, and T_1 as its 1-subtree, then $\mathrm{Inf}(T) - \mathrm{Inf}_i(T) = \frac{1}{2}(\mathrm{Inf}(T_0) + \mathrm{Inf}(T_1))$. This fact is true because

$$\begin{aligned}
\mathrm{Inf}(T) - \mathrm{Inf}_i(T) &= \sum_{j \neq i} \mathrm{Inf}_j(T) \\
&= \sum_{j \neq i} \frac{1}{2}\mathrm{Inf}_j(T_0) + \frac{1}{2}\mathrm{Inf}_j(T_1) \\
&= \frac{1}{2}\left(\sum_{j=1}^{n} \mathrm{Inf}_j(T_0) + \sum_{j=1}^{n} \mathrm{Inf}_j(T_1)\right) \\
&= \frac{1}{2}(\mathrm{Inf}(T_0) + \mathrm{Inf}(T_1)).
\end{aligned}$$

□

5.2 Lower bounds on the score of the leaf BuildTopDownDT selects

We give two different lower bounds. These lower bounds are incomparable, so when proving Theorems 3 and 5, we use whichever is better. Both of these lower bounds rely on a powerful inequality from the analysis of boolean functions due to O'Donnell, Saks, Schramm, and Servedio [OSSS05], which we restate in the form most convenient for us.

Theorem 12 (Corollary of Theorem 1.1 from [OSSS05])**.** *Let f be a size-s decision tree. Then,*

$$\max_i \left(\mathrm{Inf}_i(f)\right) \geq \frac{\mathrm{Var}(f)}{\log s}.$$

We prove our first lower bound on the score of the leaf selected.

Lemma 5.2. *Let f be a size s decision tree. At step j,* BuildTopDownDT(f, ε) *selects a leaf, ℓ^* with score at least*

$$\mathrm{score}(\ell^*) \geq \frac{\varepsilon}{(j+1)\log(s)}.$$

Proof. If BuildTopDownDT has not terminated at step j, then, the completion of T° is not an ε-approximation of f. Equivalently,

$$\sum_{\text{leaf } \ell \in T^\circ} 2^{-|\ell|} \cdot \mathrm{error}(f_\ell, \pm 1) > \varepsilon$$

At step j, there are exactly $j+1$ leaves in T°, so there must be at least one leaf, ℓ, where

$$2^{-|\ell|} \cdot \mathrm{error}(f_\ell, \pm 1) > \frac{\varepsilon}{j+1}.$$

Since $\mathrm{Var}(f_\ell) \geq \mathrm{error}(f_\ell, \pm 1)$, we also know

$$2^{-|\ell|} \cdot \mathrm{Var}(f_\ell) > \frac{\varepsilon}{j+1}.$$

By Theorem 12, we know that there is some variable x_i such that $\mathrm{Inf}_i(f_\ell) \geq \mathrm{Var}(f_\ell)/\log(\mathrm{size}(f_\ell))$. The optimal size of any restriction of f is certainly at most the optimal size of f itself, so

$$2^{-|\ell|} \cdot \mathrm{Inf}_i(f_\ell) > \frac{\varepsilon}{(j+1)\log(s)}.$$

Since BuildTopDownDT picks a leaf with maximum score, and ℓ has a score at least $\varepsilon/(j+1)\log(s)$, it must pick a leaf with at least that score. □

A standard fact from the analysis of boolean functions gives a $\log s$ upper bound on the total influence of a size-s decision tree (see e.g. [OS07]). In order to prove a second lower bound on the score of the leaf that BuildTopDownDT selects, we will need a refinement of this bound that takes into account the variance of the function. The following lemma is a slight variant of a related (though incomparable) result in [BT15], which upper bounds the total influence of an s-term DNF formula by $2\mu\log(s/\mu)$, where $\mu \coloneqq \Pr[f(\boldsymbol{x}) = 1]$.

Lemma 5.3 (Total influence of size-s DTs)**.** *Let $f : \{0,1\}^n \to \{\pm 1\}$ be computed by a size-s decision tree T. Then*

$$\mathrm{Inf}(f) \leq \mathrm{Var}(f)\log(4s/\mathrm{Var}(f)).$$

Proof. We may assume without loss of generality that $\mu \coloneqq \Pr[f(\boldsymbol{x}) = 1] \leq \frac{1}{2}$, since $\mathrm{Inf}(f) = \mathrm{Inf}(\neg f)$ and if f is a size-s decision tree then so is its negation $\neg f$. Since

$$\begin{aligned}
\mathrm{Inf}(f) &= \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\{0,1\}^n}[\mathrm{sens}_f(\boldsymbol{x})] && (\text{where } \mathrm{sens}_f(x) \coloneqq |\{i \in [n] \colon f(x) \neq f(x^{\oplus i})\}|)\\
&= 2 \cdot \mathbb{E}\left[\mathrm{sens}_f(\boldsymbol{x})\mathbb{1}[f(\boldsymbol{x}) = 1]\right]\\
&\leq 2 \sum_{\text{1-leaves } \ell \in T} 2^{-|\ell|} \cdot |\ell| && (\mathrm{sens}_f(x) \leq |\ell| \text{ for every } x \text{ that reaches } \ell)\\
&\leq 2\mu\log(s/\mu) && (\text{Concavity of } t \mapsto t\log(1/t), \text{ and } \mathrm{size}(T) \leq s)\\
&\leq \mathrm{Var}(f)\log(4s/\mathrm{Var}(f)), && (\mathrm{Var}(f) = 4\mu(1-\mu), \text{ and our assumption that } \mu \leq \tfrac{1}{2})
\end{aligned}$$

the lemma follows. □

We now provide a second lower bound on the score of the leaf BuildTopDownDT selects. The lower bound provided below in Lemma 5.4 is better than the bound provided by Lemma 5.2 when $\mathrm{cost}_f(T^\circ)$ is large.

Lemma 5.4. *Let f be a size s decision tree. Suppose, at step j, that* BuildTopDownDT(f, ε) *has already constructed the bare tree T° and that $\mathrm{cost}_f(T^\circ) \geq \varepsilon\log(4s/\varepsilon)$ Then, the next leaf, ℓ^*, that* BuildTopDownDT *picks has score at least*

$$\mathrm{score}(\ell^*) \geq \frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(4s/\varepsilon)\log(s)}.$$

Proof. We will show that when $\mathrm{cost}_f(T^\circ)$ is large, there is some leaf with high total influence, which means it must have high variance, and finally a variable with high influence.

We define:

$$h_s : [0,1] \to \mathbb{R} \text{ where } h_s(t) = t\log\left(\frac{4s}{t}\right) \text{ and } h_s(0) = 0.$$

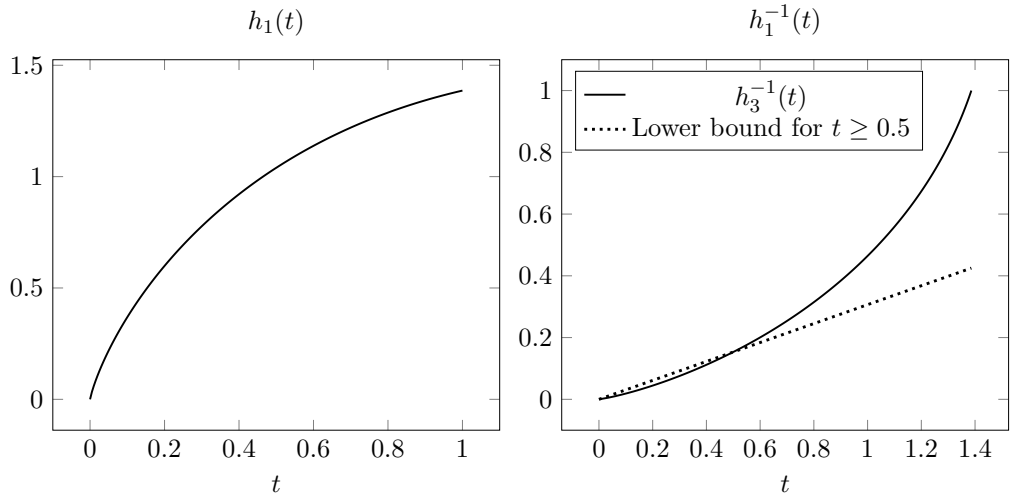

Figure 2: Graphs of the function $h_1(t) = t \cdot \log(\frac{4}{t})$ on the left, and of its inverse, $h_1^{-1}(t)$ on the right. Since the inverse is convex, we can use a linear lower bound as the dotted line in the right plot shows.

Then, for any tree T of size at most s, we have that

$$\mathrm{Inf}(T) \leq h_s(\mathrm{Var}(T)).$$

As long as $s \geq 1$, h_s is an increasing concave function. This means it has a convex inverse, h_s^{-1}, and that for any tree T of size at most s, the following lower bounds the variance.

$$\mathrm{Var}(T) \geq h_s^{-1}(\mathrm{Inf}(T)). \tag{2}$$

Since h_s^{-1} is convex and $h_s^{-1}(0) = 0$, we can lower bound it as follows. Choose arbitrary $a \in \mathbb{R}$. Then, for $t \geq a$ we have that $h_s^{-1}(t) \geq t \cdot \frac{h_s^{-1}(a)}{a}$. Choosing $a = \varepsilon \log(4s/\varepsilon)$, we have that,

$$h_s^{-1}(t) \geq \frac{t}{\log(4s/\varepsilon)} \text{ for all } t \geq \varepsilon \log\left(\frac{4s}{\varepsilon}\right).$$

Consider the bare tree, T°, at step j. By definition, it has cost

$$\sum_{\text{leaf } \ell \in T^\circ} 2^{-|\ell|} \cdot \mathrm{Inf}(f_\ell) = \mathrm{cost}_f(T^\circ).$$

We next apply Jensen's inequality.

$$\sum_{\text{leaf } \ell \in T^\circ} 2^{-|\ell|} \cdot h_s^{-1}(\mathrm{Inf}(f_\ell)) \geq h_s^{-1}(\mathrm{cost}_f(T^\circ)).$$

Since, at step j, there are $j+1$ leaves in T°, for at least one of the leaves, ℓ,

$$2^{-|\ell|} \cdot h_s^{-1}(\mathrm{Inf}(f_\ell)) \geq \frac{h_s^{-1}(\mathrm{cost}_f(T^\circ))}{j+1} \geq \frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(\frac{4s}{\varepsilon})}.$$

By Equation (2), we can lower bound the variance of f_ℓ:

$$2^{-|\ell^*|} \cdot \mathrm{Var}(f_\ell) \geq 2^{-|\ell|} \cdot h_s^{-1}(\mathrm{Inf}(f_\ell)) \geq \frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(\frac{4s}{\varepsilon})}.$$

Then, using Theorem 12 and the fact that if f is exactly computed by a size s tree, then f_ℓ is exactly computed by a tree of size at most s.

$$2^{-|\ell|} \cdot \max_i \left(\mathrm{Inf}_i[f_\ell]\right) \geq \frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(\frac{4s}{\varepsilon})\log(s)}.$$

Recall that BuildTopDownDT picks the leaf with largest score, so it will pick a leaf with score at least $\frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(4s/\varepsilon)\log(s)}$. □

5.3 Proofs of Theorems 3 and 5

Armed with the above Lemmas, we are now ready to prove our upper bounds on the size of the tree that BuildTopDownDT produces.

Theorem 3 (Upper bound for approximate representation)**.** *For every $\varepsilon \in (0, \frac{1}{2})$ and every size-s decision tree f, we have* $\textsc{TopDownDTSize}(f, \varepsilon) \leq s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}$.

Proof. We use C_j to refer to $\mathrm{cost}_f(T^\circ)$ after j steps of BuildTopDownDT. The size of the tree returned is one more than the number of steps BuildTopDownDT takes. Furthermore, if $C_j \leq \varepsilon$, then T° has error at most ε at step j, so BuildTopDownDT will return a tree of size at most $j+1$.

Our analysis proceeds in two phases:

Phase 1: We will show that the larger C_j is, the faster it must decrease at each step. This multiplicative reduction of C_j will allow us to conclude that after at most $k = s^{\log(4s/\varepsilon)\log(1/\varepsilon)}$ steps, that $C_k \leq \varepsilon \log(\frac{4s}{\varepsilon})$.

Phase 2: We will argue that C_j makes additive progress towards 0 once it is less than $\varepsilon \log(\frac{4s}{\varepsilon})$, showing that after $m = s^{2\log(4s/\varepsilon)\log(1/\varepsilon)}$ steps, that $C_m \leq \varepsilon$.

Once $C_m \leq \varepsilon$, the algorithm must terminate.

Phase 1: Based on Lemma 5.4, we know that during phase 1, BuildTopDownDT will select a leaf with influence at least $\frac{\mathrm{cost}_f(T^\circ)}{(j+1)\log(4s/\varepsilon)\log s}$ at each step j. From Lemma 5.1, we know that:

$$\begin{aligned} C_j &\leq C_{j-1} - \frac{C_j}{j\log(4s/\varepsilon)\log s} \\ &= C_{j-1} \cdot \left(1 - \frac{1}{j\log(4s/\varepsilon)\log s}\right). \end{aligned}$$

We can use this to bound C_k, the cost after some (k) number of steps, in terms of C_0.

$$\begin{aligned} C_k &\leq C_0 \prod_{j=1}^{k} \left(1 - \frac{1}{j \log(4s/\varepsilon) \log s}\right) \\ &= C_0 \exp\left(\sum_{j=1}^{k} \log\left(1 - \frac{1}{j \log(4s/\varepsilon) \log s}\right)\right). \end{aligned}$$

Using the fact that $\log(1+t) < t$,

$$\begin{aligned} C_k &\leq C_0 \exp\left(-\sum_{j=1}^{k} \frac{1}{j \log(4s/\varepsilon) \log s}\right) \\ &\leq C_0 \exp\left(-\frac{\log k}{\log(4s/\varepsilon) \log s}\right). \end{aligned}$$

We know that $C_0 \leq \log s$ because a size-s decision tree has total influence at most $\log s$ (see e.g. [OS07]). Choosing

$$k = \exp\big(\log(4s/\varepsilon) \log(s) \log(1/\varepsilon)\big) = s^{\log(4s/\varepsilon) \log(1/\varepsilon)}$$

it must be true that $C_k \leq \varepsilon \log(4s/\varepsilon)$.

Phase 2. This phase combines Lemmas 5.1 and 5.2, which together imply that

$$C_{j+1} \leq C_j - \frac{\varepsilon}{(j+1) \log s}.$$

This means that, for $m > k$,

$$C_k - C_m \geq \sum_{j=k+1}^{m} \frac{\varepsilon}{(j+1) \log s} \geq \frac{\varepsilon}{\log s} (\log m - \log k).$$

We are guaranteed to terminate at the first j such that $C_j \leq \varepsilon$, or earlier. Choosing

$$\log m = \frac{\log s}{\varepsilon} C_k + \log k$$

ensures that $C_m \leq 0$, which means BuildTopDownDT must terminate before step m. Plugging in $C_k \leq \varepsilon \log(4s/\varepsilon)$ and $k = s^{\log(4s/\varepsilon) \log(1/\varepsilon)}$ gives that

$$m \leq s^{2 \log(4s/\varepsilon) \log(1/\varepsilon)}.$$

Since BuildTopDownDT terminates after at most m steps, it returns a tree of size at most $m+1$. □

The proof of Theorem 5 is mostly the same as Phase 2 from the proof of Theorem 3, except we have a better guarantee on the starting cost. We will use the following upper bound on the total influence of monotone decision trees, due to O'Donnell and Servedio [OS07]:

Theorem 13 ([OS07]). *Let f be a size-s monotone decision tree. Then* $\mathrm{Inf}(f) \leq \sqrt{\log s}$.

Theorem 5 (Upper bound for approximate representation of monotone functions.). *For every $\varepsilon \in (0, \frac{1}{2})$ and every size-s monotone decision tree f, we have* $\textsc{TopDownDTSize}(f, \varepsilon) \leq s^{O(\sqrt{\log s}/\varepsilon)}$.

Proof. We use C_j to refer to $\mathrm{cost}_f(T^\circ)$ after j steps of BuildTopDownDT. By combining Lemma 5.2 and Lemma 5.1, we know that

$$C_{j+1} \leq C_j - \frac{\varepsilon}{(j+1)\log s}.$$

At any step k,

$$C_0 - C_k \geq \sum_{j=0}^{k-1} \frac{\varepsilon}{(j+1)\log s} \geq \frac{\varepsilon \cdot \log k}{\log s}.$$

Since f is a monotone decision tree of size s, it has total influence at most $\sqrt{\log s}$ (Theorem 13). This means that $C_0 \leq \sqrt{\log s}$. We choose

$$k = \exp(\log(s)^{1.5}/\varepsilon) = s^{\sqrt{\log s}/\varepsilon}$$

at which point, $C_k \leq 0 \leq \varepsilon$, so BuildTopDownDTreturns a tree of size $k+1$. □

6 Lower bounds on TopDownDTSize for general functions: Proof of Theorem 4

6.1 Size separation for exact representation: Proof of Theorem 4(a)

We begin with a simple family of functions $\{f_h\}_{h \in \mathbb{N}}$ whose BuildTopDownDT tree has exponential size compared to the optimal tree. Each f_h is a function over $3h+1$ boolean variables $x_1^{(1)}, x_2^{(1)}, \ldots, x_1^{(h)}, x_2^{(h)}, y^{(1)}, \ldots, y^{(h)}, z$, and is defined inductively as follows:

$$f_0(z) = z,$$

and for $h \geq 1$,

$$f_h(x, y, z) = \begin{cases} y^{(h)} & \text{if } x_1^{(h)} \vee x_2^{(h)} \\ f_{h-1}(x, y, z) & \text{otherwise.} \end{cases}$$

The structure of BuildTopDownDT(f_h). We see that $y^{(h)}$ has influence $\frac{3}{4}$, both $x_1^{(h)}$ and $x_2^{(h)}$ have influence $\frac{1}{4}$, and each variable in f_{h-1} has influence $< \frac{1}{4}$. BuildTopDownDT(f_h) therefore queries y_k at the root. In the restrictions of f_h obtained by setting $y^{(h)}$ to a constant, $x_1^{(h)}$ and $x_2^{(h)}$ have equal influence of $\frac{1}{4}$ and each variable in f_{h-1} has influence $< \frac{1}{4}$. By setting either $x_1^{h)}$ or $x_2^{(h)}$ to a constant, we get a subfunction where the other $x^{(h)}$-variable has influence $\frac{1}{2}$ and each node in f_{h-1} has influence $< \frac{1}{2}$. Thus, BuildTopDownDT(f_h) builds the tree T_h depicted in Figure 3.

We see that each T_h contains two copies of T_{h-1}. It follows that the optimal size of f_h is $O(h)$, whereas the size of T_h is $2^{\Omega(h)}$: a size separation of $\textsc{TopDownDTSize}(f_h) = 2^{\Omega(s)}$ where s denotes the optimal size of f_h.

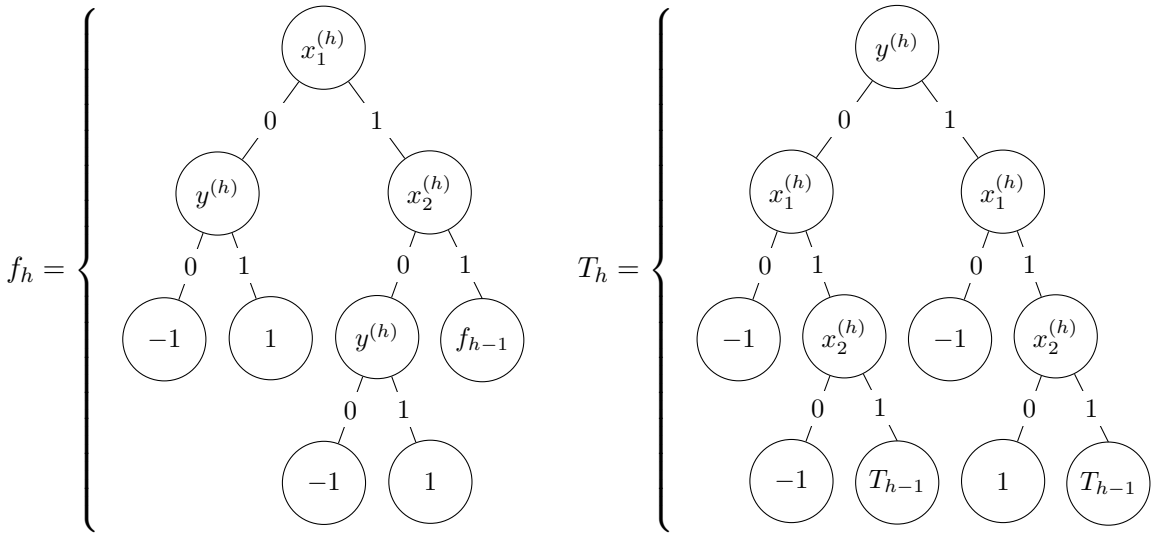

Figure 3: Diagrams exhibiting a function with exponential difference between the optimal decision tree size and TopDownDTSize. The left diagram shows how to compute f_h with a decision tree of size $O(h)$. The right diagram shows T_h, the tree BuildTopDownDT builds, which has size $2^{\Omega(h)}$.

6.2 Size separation for approximate representation: Proof of Theorem 4(b)

Warmup/intuition: An s versus $s^{\Omega(\log(1/\varepsilon))}$ separation. Before proving Theorem 4(b), we first give a brief, informal description of how a simple modification to the family of functions $\{f_h\}_{h\in\mathbb{N}}$ in Theorem 4(a) above yields a separation of $\textsc{TopDownDTSize}(f,\varepsilon) = s^{\Omega(\log(1/\varepsilon))}$ for approximate representation. Theorem 4(b)—which improves this to a superpolynomial separation even for constant ε—builds on these ideas, but the family of functions and the proof of the lower bound are significantly more involved.

Consider replacing each $y^{(h)}$ variable in the definition of f_h with the parity of k variables $y_1^{(h)} \oplus \cdots \oplus y_k^{(h)}$, i.e. consider the following variant $\tilde{f}_h$ of f_h:

$$\tilde{f}_h(x,y,z) = \begin{cases} y_1^{(h)} \oplus \cdots \oplus y_k^{(h)} & \text{if } x_1^{(h)} \vee x_2^{(h)} \\ \tilde{f}_{h-1}(x,y,z) & \text{otherwise.} \end{cases}$$

Just like the single $y^{(h)}$ variable in f_h, we see that the k many $y_i^{(h)}$ variables are the most influential in $\tilde{f}_h$ (each having influence $\frac{3}{4}$). Furthermore, each $y_i^{(h)}$ variable remains the most influential even under *any* restriction to *any* number of the other $y_j^{(h)}$ variables. Therefore the tree $\tilde{T}_h$ that BuildTopDownDT builds for $\tilde{f}_h$ first queries all k many $y^{(h)}$ variables. At each of the 2^k resulting leaves, $x_1^{(h)}$ and $x_2^{(h)}$ are then queried, followed by a copy of $\tilde{T}_{h-1}$, the tree that BuildTopDownDT recursively constructs for $\tilde{f}_h$, in the branch corresponding to $x_1^{(h)} = x_2^{(h)} = 1$. The fact that there are 2^k copies of $\tilde{T}_{h-1}$ within $\tilde{T}_h$ should be contrasted with the fact that the tree T_h in Theorem 4(a) contains just two copies of T_{h-1}; recall Figure 3.

It is straightforward to see that there is a tree of size $O(h \cdot 2^k)$ that computes $\tilde{f}_h$. This tree is built by first querying the $x^{(h)}$ variables before the $y^{(h)}$ variables, and recursing on just one of the $\Omega(2^k)$ many resulting leaves. On the other hand, by first querying the $y^{(h)}$ variables followed by the $x^{(h)}$ variables, BuildTopDownDT recurses on $\Omega(2^k)$ many branches while only correctly classifying a $\frac{3}{4}$ fraction of inputs. Choosing $h = \Theta(\log(1/\varepsilon))$, we get a separation of $O(h \cdot 2^k)$ versus $2^{\Omega(kh)}$, or equivalently, s versus $s^{\Omega(\log(1/\varepsilon))}$.

6.2.1 Proof of Theorem 4(b)

Before defining the family of functions witnessing the separation, we define a couple of basic boolean functions and state a few of their properties that will be useful for our analyses:

Definition 14 (Tribes)**.** For any input length r, let w be the largest integer such that $(1 - 2^{-w})^{r/w} \leq \frac{1}{2}$. The $\textsc{Tribes}_r : \{0,1\}^r \to \{\pm 1\}$ function is defined to be the function computed by the read-once DNF with $\lfloor \frac{r}{w} \rfloor$ terms (over disjoint sets of variables) of width exactly w:

$$\textsc{Tribes}_r(z) = (z_{1,1} \wedge \cdots \wedge z_{1,w}) \vee \cdots \vee (z_{t,1} \wedge \cdots \wedge z_{t,w}) \quad \text{where } t \coloneqq \lfloor \tfrac{r}{w} \rfloor,$$

and where we adopt the convention that -1 represents logical False and 1 represents logical True.

The following facts about the Tribes function are standard (see Chapter §4.2 of [O'D14]) and can be easily verified:

Fact 6.1 (Properties of $\textsc{Tribes}_r$)**.**

- $\Pr[\textsc{Tribes}_r(\boldsymbol{z}) = 1] = \frac{1}{2} - O\big(\frac{\log r}{r}\big)$.
- $\mathrm{Inf}(\textsc{Tribes}_r) = (1 \pm o(1)) \cdot \ln r$ *and consequently,* $\mathrm{Inf}_i(\textsc{Tribes}_r) = (1 \pm o(1)) \cdot \frac{\ln r}{r}$ *for all* $i \in [n]$.
- $w = \log r - \log \ln r \pm O(1)$.
- $\mathrm{size}(\textsc{Tribes}_r) \leq w^{O(r/w)} = 2^{O(r \log\log r / \log r)}$.

Definition 15 (Threshold)**.** For any input length ℓ and $t \in \{0, 1, \ldots, \ell\}$, the $\textsc{Threshold}_{\ell,t} : \{0,1\}^\ell \to \{\pm 1\}$ function is defined to be

$$\textsc{Threshold}_{\ell,t}(x) = 1 \iff \sum_{i=1}^{\ell} x_i \leq t.$$

Defining the family of functions witnessing the separation. Consider the following family of functions $\{f_h\}_{h \in \mathbb{N}}$. Each f_h is a function over $h(\ell + k) + r$ boolean variables $x^{(1)}, x^{(2)}, \ldots, x^{(h)} \in \{0,1\}^\ell$,$y^{(1)}, \ldots, y^{(h)} \in \{0,1\}^k$, and $z \in \{0,1\}^r$, and is defined inductively as follows:

$$f_0(z) = \textsc{Tribes}_r(z),$$

and for $h \geq 1$,

$$f_h(x, y, z) = \begin{cases} \textsc{Parity}_k(y^{(h)}) & \text{if } \textsc{Threshold}_{\ell,1}(x^{(h)}) = 1 \\ f_{h-1}(x, y, z) & \text{otherwise.} \end{cases}$$

Claim 6.2 (Optimal decision tree size of f_h)**.**

$$\begin{aligned}\text{size}(f_h) &\le \ell^{O(h)} \cdot (\text{size}(\textsc{Parity}_k) + \text{size}(\textsc{Tribes}_r)) \\ &\le \ell^{O(h)} \cdot (2^k + 2^{O(r \log\log r / \log r)}).\end{aligned}$$

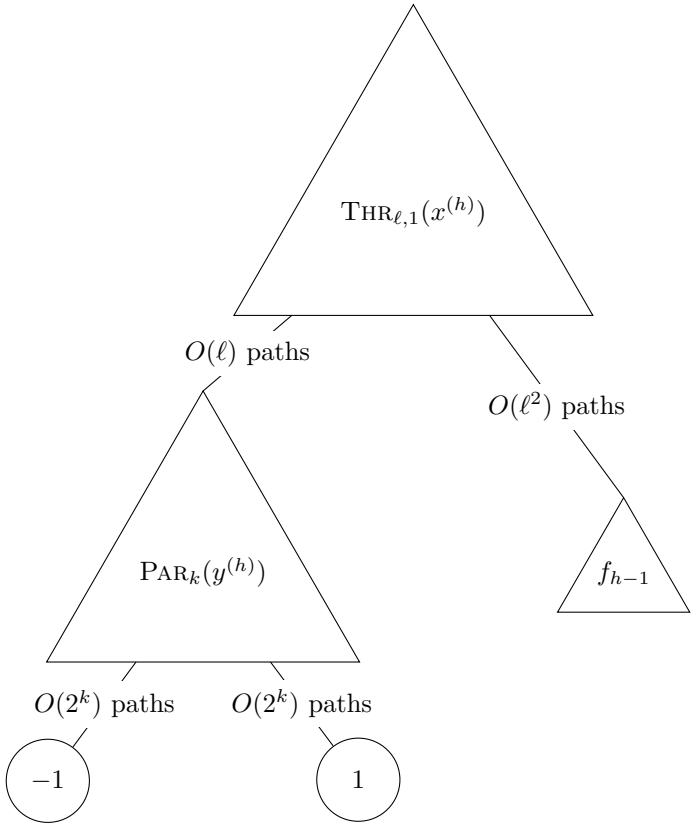

Figure 4: A small decision tree for f_h.

Proof. Please refer to figure Figure 4. We first build a tree of size $O(\ell^2)$ that evaluates $\textsc{Threshold}_{\ell,1}(x^{(h)})$. Of these leaves, $\ell + 1$ descend into a tree computing $\textsc{Parity}_k(y^{(h)})$, which has size 2^k. The others descend into a tree computing f_{h-1}. This yields the recurrence

$$\begin{aligned}\text{size}(f_h) &\le O(\ell) \cdot \text{size}(\textsc{Parity}_k) + O(\ell^2) \cdot \text{size}(f_{h-1}) \\ &\le O(\ell \cdot 2^k) + O(\ell^2) \cdot \text{size}(f_{h-1}) \\ \text{size}(f_0) &= \text{size}(\textsc{Tribes}_r) \le 2^{O(r \log\log r / \log r)}, \qquad \text{(Recall Fact 6.1)}\end{aligned}$$

and the claim follows. □

The remainder of this section will be devoted to proving a lower bound on $\textsc{TopDownDTSize}(f, \varepsilon)$. Figure 5 should be contrasted with Figure 4.

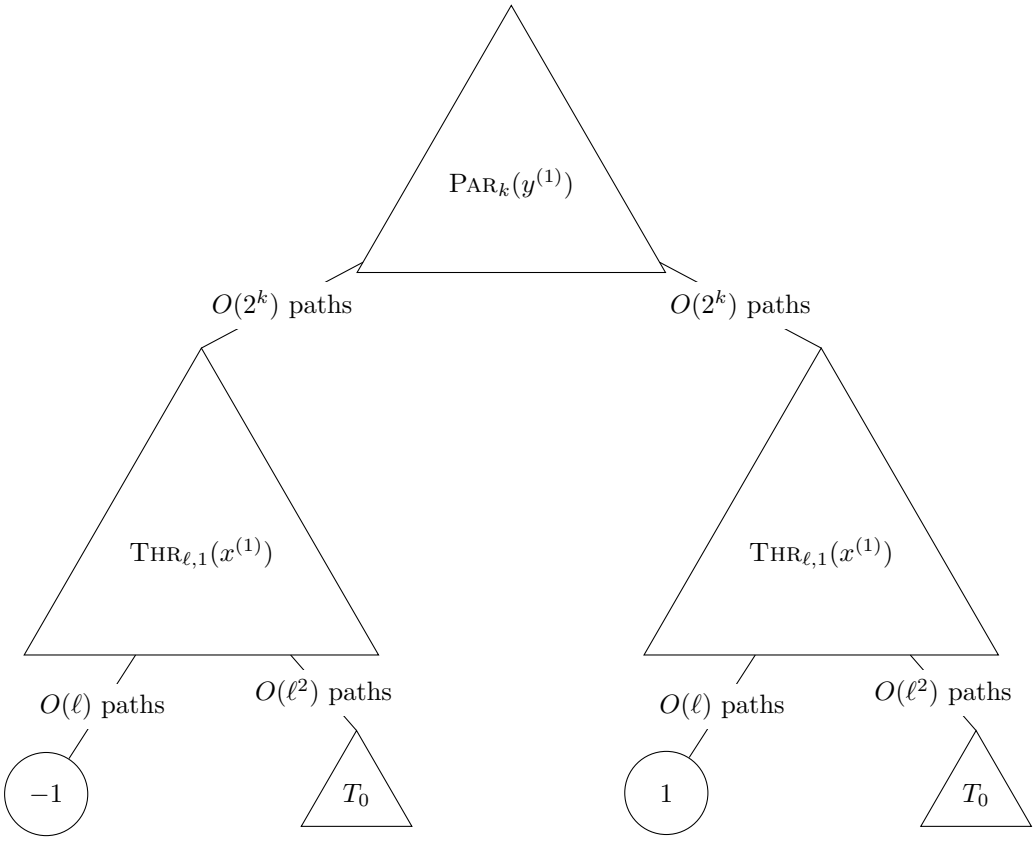

Figure 5: The tree T_1 that BuildTopDownDT builds for f_1. Since $y^{(1)}$ has all the most influential variables, BuildTopDownDT puts them all at the root. As a result, it ends up building a significantly larger tree than optimal (cf. Figure 4). Notice that the size of T_1 is $\Omega(2^k)$ times as large as T_0. This leads to exponential growth of the tree size as a function of h.

The structure of BuildTopDownDT(f_h) The following helper lemma will be useful in determining the structure of the tree BuildTopDownDT produces.

Lemma 6.3 (Preservation of influence order)**.** *Let $f : \{0,1\}^{\overline{S}} \times \{0,1\}^S \to \{\pm 1\}$ and $\tilde{f} : \{0,1\}^S \to \{\pm 1\}$ be two functions satisfying the following: there is a function $g : \{0,1\}^{\overline{S}} \times \{\pm 1\} \to \{\pm 1\}$ such that:*

$$f(a,b) = g(a, \tilde{f}(b)) \quad \text{for all } a \in \{0,1\}^{\overline{S}},\ b \in \{0,1\}^S.$$

Then for all variables $v_1, v_2 \in S$,

$$\mathrm{Inf}_{v_1}(\tilde{f}) \geq \mathrm{Inf}_{v_2}(\tilde{f}) \quad \text{if and only if} \quad \mathrm{Inf}_{v_1}(f) \geq \mathrm{Inf}_{v_2}(f).$$

Proof. This holds by noting that for $v \in \{v_1, v_2\}$,

$$\begin{aligned}
\mathrm{Inf}_v(f) &= \Pr_{\boldsymbol{a},\boldsymbol{b}}[f(\boldsymbol{a},\boldsymbol{b}) \neq f(\boldsymbol{a},\boldsymbol{b}^{\oplus v})] \\
&= \Pr_{\boldsymbol{a},\boldsymbol{b}}[g(\boldsymbol{a},\tilde{f}(\boldsymbol{b})) \neq g(\boldsymbol{a},\tilde{f}(\boldsymbol{b}^{\oplus v}))] \\
&= \Pr_{\boldsymbol{b}}[\tilde{f}(\boldsymbol{b}) \neq \tilde{f}(\boldsymbol{b}^{\oplus v})] \cdot \Pr_{\boldsymbol{a}}[g(\boldsymbol{a},-1) \neq g(\boldsymbol{a},1)] \\
&= \mathrm{Inf}_v(\tilde{f}) \cdot \Pr_{\boldsymbol{a}}[g(\boldsymbol{a},-1) \neq g(\boldsymbol{a},1)].
\end{aligned}$$

The lemma follows since $\Pr_{\boldsymbol{a}}[g(\boldsymbol{a},-1) \neq g(\boldsymbol{a},1)]$ does not depend on v (and hence is the same regardless of whether $v = v_1$ or $v = v_2$). □

Lemma 6.3 is especially well-suited for our inductively-defined family of functions $\{f_h\}_{h\in\mathbb{N}}$. For each $i \in \{0,1,\ldots,h\}$, we let S_i denote the relevant variables of f_i. Therefore

$$\begin{aligned}
S_0 &= \{z_1,\ldots,z_r\} \\
S_{i+1} &= S_i \sqcup \{x_1^{(i)},\ldots,x_\ell^{(i)},y_1^{(i)},\ldots,y_k^{(i)}\}
\end{aligned}$$

Observation 16. *For all $i \in \{0,1,\ldots,h\}$, there exists g_i such that*

$$f_h(a,b) = g_i(a,h_i(b)) \quad \text{for all } a \in \{0,1\}^{S_h \setminus S_i} \text{ and } b \in \{0,1\}^{S_i}. \tag{3}$$

Consequently, we may apply Lemma 6.3 to get that for all $v_1, v_2 \in S_i$, we have that

$$\mathrm{Inf}_{v_1}(f_i) \geq \mathrm{Inf}_{v_2}(f_i) \quad \text{if and only if} \quad \mathrm{Inf}_{v_1}(f_h) \geq \mathrm{Inf}_{v_2}(f_h).$$

We note the following corollary, which is a straightforward consequence of the observation that the property (3) is preserved under restrictions:

Corollary 6.4 (Preservation of influence order under restrictions)**.** *Let π be any restriction. For all $i \in \{0,1,\ldots,h\}$, we have that*

$$\mathrm{Inf}_{v_1}((f_i)_\pi) \geq \mathrm{Inf}_{v_2}((f_i)_\pi) \quad \text{if and only if} \quad \mathrm{Inf}_{v_1}((f_h)_\pi) \geq \mathrm{Inf}_{v_2}((f_h)_\pi).$$

Lower bounding the size of BuildTopDownDT(f,ε). Let T_{exact} denote the tree returned by BuildTopDownDT(f_h), and T_{approx} denote the tree returned by BuildTopDownDT(f_h,ε). (So T_{exact} computes f_h, and T_{approx} is an ε-approximation of f_h.) Our goal is to lower bound the size of T_{approx}. We will in fact establish something stronger: our lower bound holds for *any pruning* of T_{exact} that is an ε-approximation of T_{exact}, where a pruning of a tree T is any tree obtained by iteratively removing leaves from T in a bottom-up fashion. Since BuildTopDownDT(f,ε) is simply BuildTopDownDT(f_h) terminated early, we have that T_{approx} is indeed a pruning of T_{exact}.

Let V_{exact} be defined as follows:

$$V_{\text{exact}} \coloneqq \{v\colon v \text{ is the first node in a path of } T_{\text{exact}} \text{ that queries a } z\text{-variable}\}.$$

We define $V_{\text{approx}} \subseteq V_{\text{exact}}$ analogously. At a very high level, our proof of Theorem 4(b) will proceed by showing that V_{exact} has large size, and V_{approx} has to contain many nodes in V_{exact}. For the remainder of this proof, we will need that r and ℓ are chosen to satisfy:

$$\frac{2\ln r}{r} < 2^{-\ell}. \tag{4}$$

Lemma 6.5 (All nodes in V_{exact} occur deep within T_{exact})**.** *Fix $v \in V_{\text{exact}}$ and let π denote the path in T_{exact} that leads to v. Then $|\pi| \geq kh$.*

Proof. Suppose without loss of generality that v is a query to z_1. We claim that $y_j^{(i)} \in \pi$ for all $i \in [h]$ and $j \in [k]$, from which the lemma follows. Fix $i \in [h]$. We will prove there are at least k queries to variables in S_i within π, and that the first k of these queries have to be $y_j^{(i)}$ for $j \in [k]$. We prove both these claims simultaneously by induction on k.

- (Base case.) Seeking a contradiction, suppose π does not contain any queries to variables in S_i, in which case $(f_i)_\pi \equiv f_i$. Since z_1 is the variable queried at the root of $(f_h)_\pi$, it is the most influential variable within $(f_h)_\pi$. By Corollary 6.4, it follows that z is the most influential variable within $(f_i)_\pi \equiv f_i$. This contradicts Equation (4) since

$$\text{Inf}_{y_1^{(i)}}(f_i) = \frac{\ell+1}{2^\ell} \quad \text{and} \quad \text{Inf}_{z_1}(f_i) < \text{Inf}_{z_1}(\textsc{Tribes}_r) = (1 \pm o(1)) \cdot \frac{\ln r}{r}.$$

  Therefore π has to contain at least one variable in S_i. Let $u \in \pi$ be the first query to a variable in S_i, which we claim must be $y_j^{(i)}$ for some j. Let $\pi_u \subset \pi$ be the path in T_{exact} that leads to u. Again, we have that u must be the most influential variable within $(f_h)_{\pi_u}$, and hence, by Corollary 6.4, it is the most influential within $(f_i)_{\pi_u}$. Since π_u does not contain any queries to variables in S_i, we have that $(f_i)_{\pi_u} \equiv f_i$, and hence u must be $y_j^{(i)}$ for some j since these are the most influential variables within f_i.

- (Inductive step.) Fix $k' < k$, and suppose we have established that there are at least k' queries to variables in S_i within π, the first k' of which are to $y^{(i)}$-variables. We first claim that there is at least one more query to variable in S_i within π. Suppose not. It follows that z_1 must be the most influential variable within $(f_h)_\pi$, and hence, by Corollary 6.4, it is the most influential variable within $(f_i)_\pi$. This is a contradiction, since z_1 is less influential than any of the $k - k'$ many $y_j^{(i)}$ variables that are not queried by π.

  Therefore π has to contain at least one more query to a variable in S_i. Let $u \in \pi$ be the $(k+1)^{st}$ query to a variable in S_i, which we claim must be $y_j^{(i)}$ for some j. Let $\pi_u \subset \pi$ be the path in T_{exact} that leads to u. Again, we have that u must be the most influential variable within $(f_h)_{\pi_u}$, and hence, by Corollary 6.4, it is the most influential within $(f_i)_{\pi_u}$. Since π_u contains exactly k' queries to variables S_i, and all these queries are to $y^{(i)}$ variables, we have that u must be $y_j^{(i)}$ for one of the remaining $k - k'$ many $y^{(i)}$-variables since these are the most influential variables within $(f_i)_{\pi_u}$.

This completes the inductive proof of Lemma 6.5. □

Lemma 6.6. *Fix $v \in V_{\text{exact}}$ and let π denote that path in T_{exact} that leads to v. Then $(f_h)_\pi \equiv \textsc{Tribes}_r$.*

Proof. Suppose $(f_h)_\pi \not\equiv \textsc{Tribes}_r$. Our proof of Lemma 6.5 shows that π contains every y-variable, so it must be the case that some x-variable remains relevant (i.e. has nonzero influence) in $(f_h)_\pi$. Let $i^* \geq 1$ be the highest value of i for which there is a relevant $x^{(i)}$-variable in $(f_h)_\pi$. Assume without loss of generality that $x_1^{(i)}$ remains relevant, and that z_1 is that z-variable that is queried at v.

Since z_1 is queried at the root of $(f_h)_\pi$, we have that it must be maximally influential in $(f_h)_\pi$, and in particular,

$$\mathrm{Inf}_{z_1}((f_h)_\pi) \geq \mathrm{Inf}_{x_1^{(i^*)}}((f_h)_\pi).$$

Applying Corollary 6.4, we infer that

$$\mathrm{Inf}_{z_1}((f_{i^*})_\pi) \geq \mathrm{Inf}_{x_1^{(i^*)}}((f_{i^*})_\pi). \tag{5}$$

Let us say that an input (x, y, z) to f_{i^*} is *z-dependent* if

$$\textsc{Threshold}_{\ell,1}(x^{(i)}) = 0 \quad \text{for all } 1 \leq i \leq i^*.$$

Note that the output of f_{i^*} on any z-dependent input is $\textsc{Tribes}_r(z)$. Since π contains every y-variable, it fixes $\textsc{Parity}_k(y^{(i^*)})$ to either -1 or 1; we assume without loss of generality that $\textsc{Parity}_k(y^{(i^*)})_\pi \equiv 1$. We have that

$$\begin{aligned}
\mathrm{Inf}_{x_1^{(i^*)}}((f_{i^*})_\pi) &\geq \Pr_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}[(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \text{ is } z\text{-dependent}] \cdot \Pr_{\boldsymbol{z}}[\textsc{Tribes}_r(\boldsymbol{z}) \neq 1] \\
&\quad \times \mathrm{Inf}_{x_1^{(i^*)}}\big(\textsc{Threshold}_{\ell,1}(x^{(i^*)})_\pi\big) \\
&\geq \Pr_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}[(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \text{ is } z\text{-dependent}] \cdot \tfrac{1}{2} \cdot 2^{-(\ell-1)} \\
&= \Pr_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}[(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \text{ is } z\text{-dependent}] \cdot 2^{-\ell}.
\end{aligned}$$

The second inequality uses the fact that $\mathrm{Inf}_{x_1^{(i^*)}}(\textsc{Threshold}_{\ell,1}(x^{(i^*)})_\pi) \geq 2^{-(\ell-1)}$, which holds with equality when exactly one other $x^{(i^*)}$-variable is in π and that variable is set to 1.[16]

On the other hand, we have that

$$\begin{aligned}
\mathrm{Inf}_{z_1}((f_{i^*})_\pi) &\leq \Pr_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}[(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \text{ is } z\text{-dependent}] \cdot \mathrm{Inf}_{z_1}[\textsc{Tribes}_r(z)] \\
&< \Pr_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}[(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \text{ is } z\text{-dependent}] \cdot \frac{2\ln r}{r}. \qquad \text{(Fact 6.1)}
\end{aligned}$$

These bounds on influences, along with Equation (5), imply that $2^{-\ell} < \frac{2\ln r}{r}$. This contradicts our assumption on the relationship between ℓ and r (Equation (4)), and the proof is complete. □

We are now ready to lower bound the size of T_{approx}.

Claim 6.7 (Lower bound on the size of T_{approx})**.** *Fix $\varepsilon \in (0, \frac{1}{2})$ and let $c = (\frac{1}{2} - \varepsilon)/2$. If*

$$\left(1 - \frac{\ell+1}{2^\ell}\right)^h \geq (2+c)\varepsilon, \tag{6}$$

then $|V_{\mathrm{approx}}| \geq \Omega(\varepsilon \cdot 2^{kh})$. Consequently, the size of T_{approx} is also at least $\Omega(\varepsilon \cdot 2^{kh})$.

[16] In this derivation, we have assumed that $\textsc{Tribes}_r$ is *perfectly* balanced, i.e. that $\Pr[\textsc{Tribes}_r(\boldsymbol{z}) = 1] = \frac{1}{2}$, when in fact $\Pr[\textsc{Tribes}_r(\boldsymbol{z}) = 1] = \frac{1}{2} \pm o(1)$ (recall Fact 6.1). The same proof goes through if one carries around the additive $o(1)$ factor.

Proof. An input to f_h reaches *some* node in V_{exact} if and only if $\textsc{Threshold}_{\ell,1}(x^{(i)}) = 0$ for all $1 \le i \le h$. The fraction of inputs that satisfies this is exactly $(1 - \frac{\ell+1}{2^\ell})^h$, which is at least $(2+c)\varepsilon$ by our choice of parameters given by Equation (6).

Fix $v \in V_{\text{exact}}$. If $v \notin V_{\text{approx}}$, then T_{approx} assigns all inputs reaching v the same -1 or $+1$ value, whereas f_h labels half of them -1 and half of them $+1$ (Lemma 6.6). Therefore, T_{approx} errors on half of the inputs that each v. On the other hand, if $v \in V_{\text{approx}}$, we have by Lemma 6.5 that at most a 2^{-kh} fraction of inputs reach this specific v. Combining all of the above observations, it follows that

$$\text{error}(T_{\text{exact}}, T_{\text{approx}}) \ge \tfrac{1}{2}\left((2+c)\varepsilon - |V_{\text{approx}}| \cdot 2^{-kh}\right).$$

Since $\text{error}(T_{\text{exact}}, T_{\text{approx}}) \le \varepsilon$, it follows that

$$\varepsilon \ge \tfrac{1}{2}\left((2+c)\varepsilon - |V_{\text{approx}}| \cdot 2^{-kh}\right),$$

and the claim follows by rearranging. □

Theorem 4(b) now follows from Claim 6.2 and Claim 6.7 by setting parameters appropriately:

Proof of Theorem 4(b). Choosing

$$\begin{aligned} h &= \Theta\left(\frac{2^\ell}{\ell} \cdot \log(1/\varepsilon)\right) && \text{(to satisfy Equation (6))} \\ r &= \Theta(\ell 2^\ell) && \text{(to satisfy Equation (4))} \\ k &= \Theta(h \log \ell), \end{aligned}$$

we may apply Claim 6.2 and Claim 6.7 to get that

$$\text{size}(f_h) \le 2^{O(k \log k)} \quad \text{whereas} \quad \textsc{TopDownDTSize}(f, \varepsilon) \ge 2^{\Omega_\varepsilon(k^2/\log\log k)}.$$

This is a separation of s versus $s^{\tilde{\Omega}(\log s)}$. □

Remark 17. For our choice of parameters above, we have that $s(n) = \text{size}(f_h) = 2^{\tilde{\Theta}(\sqrt{n})}$, where $n = h(\ell + k) + r$ is the number of variables of f_h. A standard padding argument yields the same s versus $s^{\tilde{\Omega}(\log s)}$ separation for any function $s(n) \le 2^{\tilde{O}(\sqrt{n})}$.

7 Lower bounds on TopDownDTSize for monotone functions: Proof of Theorem 6

7.1 Size separation for exact representation: Proof of Theorem 6(a)

We will give a family of monotone functions, $\{f_h\}_{h \in \mathbb{N}}$ whose $\textsc{BuildTopDownDT}$ tree has exponential size compared to the optimal tree. First, we define a few terms which will be useful for our monotone constructions.

Definition 18 (Comparing vectors and upper/lower shadows)**.** For any $x, y \in \{0,1\}^n$, we use $x \preceq y$ to represent

$$x \preceq y \iff x_i \le y_i \text{ for all } i \in [n]$$

and $\succeq$ is defined similarly. For any vector x, the *upper shadow* of x is the set of all vectors y such that $x \preceq y$. Similarly, the *lower shadow* of x is the set of all vectors y such that $x \succeq y$.

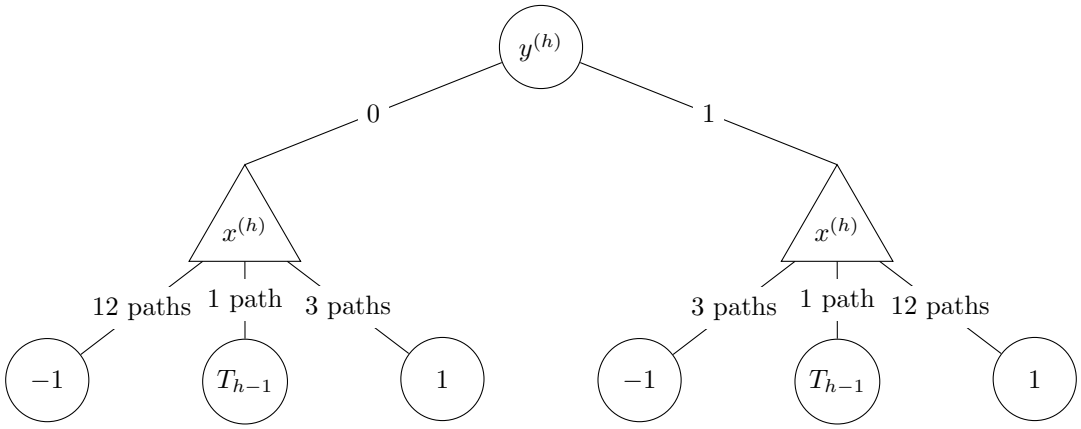

Figure 6: The tree that BuildTopDownDT builds for f_h. It will first query $y^{(h)}$, followed by the variables of $x^{(h)}$. For most choices of $y^{(h)}$ and $x^{(h)}$, the function is determined, and BuildTopDownDT will place a constant leaf equal to ± 1. However, the paths with $y = 0, x^{(h)} = x^*$ and $y = 1, x^{(h)} = x^*$ each include a copy of the tree for T_{h-1}.

Defining the family of functions witnessing the separation. Each f_h in $\{f_h\}_{h\in\mathbb{N}}$ is a function over $5h+1$ boolean variables $x^{(1)}, x^{(2)}, \ldots, x^{(h)} \in \{0,1\}^4$, $y^{(1)}, \ldots, y^{(h)} \in \{0,1\}$, and $z \in \{0,1\}$, and is defined inductively as follows:

$$f_0(z) = z,$$

and for $h \geq 1$, we fix $x^* := (0,0,1,1)$ and define

$$f_h(x,y,z) = \begin{cases} f_{h-1}(x,y,z) & \text{if } x^{(h)} = x^* \\ +1 & \text{if } x^{(h)} \succeq x^* \text{ and } x^{(h)} \neq x^* \\ -1 & \text{if } x^{(h)} \preceq x^* \text{ and } x^{(h)} \neq x^* \\ y & \text{otherwise.} \end{cases}$$

It is straightforward to verify that f_h is indeed monotone. We will show that f_h can be computed by a tree of size $O(h)$, but that BuildTopDownDT produces a tree of size $2^{\Omega(h)}$. For the first claim, we construct a decision tree for f_h directly from its definition. We start with a complete tree on the $x^{(h)}$ variables—this complete tree has size 2^4, a constant. At one of the branches, we recursively build a tree for f_{h-1}; at all the other branches, we build a tree of size 1 or 2 computing one of $-1, 1$, or $y^{(h)}$. The result is a tree of size $O(h)$.

On the other hand, we claim that BuildTopDownDT will build a tree of size $2^{\Omega(h)}$, as depicted in Figure 6. In f_h, $y^{(h)}$ has influence $\frac{9}{16}$ and all the other variables have influence at most $\frac{1}{2}$. Hence, $y^{(h)}$ will be placed at root. Then, BuildTopDownDT will query enough of $x^{(h)}$ to determine whether the output should be $-1, +1$, or f_{h-1}. If the output should be f_{h-1}, which will occur once for each choice of y, then the entire tree T_{h-1} will be placed. Hence, the size of T_h is more than double the size of T_{h-1}, and BuildTopDownDT builds a tree of size $2^{\Omega(h)}$.

7.2 Size separation for approximate representation: Theorem 6(b)

For any ε, we will prove there exists a function f with optimal tree size s but for which the tree BuildTopDownDT(f, ε) builds has size $s^{\tilde{\Omega}(\sqrt[4]{\log s})}$. The following function, a biased version of the Tribes function defined in Definition 14, will be used as a building block in our monotone construction.

Definition 19 (Biased Tribes)**.** Fix any input length ℓ and $\delta \in (0,1)$. We define $\text{Tribes}_{\ell,\delta} : \{0,1\}^\ell \to \{\pm 1\}$ to be the read-once DNF with $\lfloor \frac{\ell}{w} \rfloor$ terms of width exactly w over disjoint sets of variables (with some variables possibly left unused), where $w = w(\ell, \delta) \approx \log(\ell) \pm \log\log(1/\delta)$ is chosen such that $\Pr[\text{Tribes}_{\ell,\delta}(\boldsymbol{x}) = 1]$ is as close to δ as possible.[17]

Fact 7.1 (Variable influences in biased Tribes)**.** *All variables in* $\text{Tribes}_{\ell,\delta}$ *and* $\text{Tribes}_{\ell,1-\delta}$ *have influence at most*

$$(2 + o(1)) \cdot \delta \log(1/\delta) \cdot \frac{\log \ell}{\ell}.$$

Proof. We prove the lemma for the case of $\text{Tribes}_{\ell,1-\delta}$. (The calculations for $\text{Tribes}_{\ell,\delta}$ are very similar, and both claims are special cases of more general facts about variable influences in DNF formulas [ST13].) Suppose

$$\text{Tribes}_{\ell,1-\delta}(x) = T_1(x) \vee \cdots T_{\frac{\ell}{w}}(x),$$

where the T_i's are disjoint terms of width exactly w. We first observe that since

$$\begin{aligned} \delta = \Pr_{\boldsymbol{x} \sim \{0,1\}^n}[\text{Tribes}_{\ell,1-\delta}(\boldsymbol{x}) = 1] &= \Pr[\text{all } T_i(\boldsymbol{x}) \text{ are falsified by } \boldsymbol{x}] \\ &= (1 - 2^{-w})^{\ell/w} \approx e^{-\ell/w 2^w}, \end{aligned}$$

we have that $w = (1 \pm o(1))(\log \ell - \log\log \ell - \log\log(1/\delta))$. The influence of any variable $i \in [n]$ on $\text{Tribes}_{\ell,1-\delta}$ is the probability, over a uniform $\boldsymbol{x}$ that each other variable j in i's term has $\boldsymbol{x}_j = 1$ and all other clauses evaluate to 0 under $\boldsymbol{x}$:

$$\begin{aligned} \text{Inf}_i(\text{Tribes}_{\ell,1-\delta}) &= 2^{-(w-1)} \cdot (1 - 2^{-w})^{(\ell/w)-1} \\ &\leq 2\delta \cdot 2^{-w} \\ &= (1 \pm o(1)) \cdot 2\delta \log(1/\delta) \cdot \frac{\log \ell}{\ell}. \end{aligned}$$

□

Defining the family of functions witnessing the separation. Each f_h in the family $\{f_h\}_{h \in \mathbb{N}}$ is a function over $h(2\ell + k) + r$ boolean variables $x^{(1,1)}, x^{(1,2)}, \ldots, x^{(h,1)}, x^{(h,2)} \in \{0,1\}^\ell$, $y^{(1)}, \ldots, y^{(h)} \in \{0,1\}^k$, and $z \in \{0,1\}^r$, and is defined inductively as follows:

$$f_0(z) = \text{Tribes}_r(z),$$

[17] Although the acceptance probability of $\text{Tribes}_{\ell,\delta}$ cannot be made *exactly* δ due to granularity issues, it will be the case that $\text{Tribes}_{\ell,\delta} = \delta \pm o(1)$. For clarity, we will assume for the rest of this paper that the acceptance probability of Tribes_δ is exactly δ, noting that all of our proofs still go through if one carries around the $o(1)$ factor.

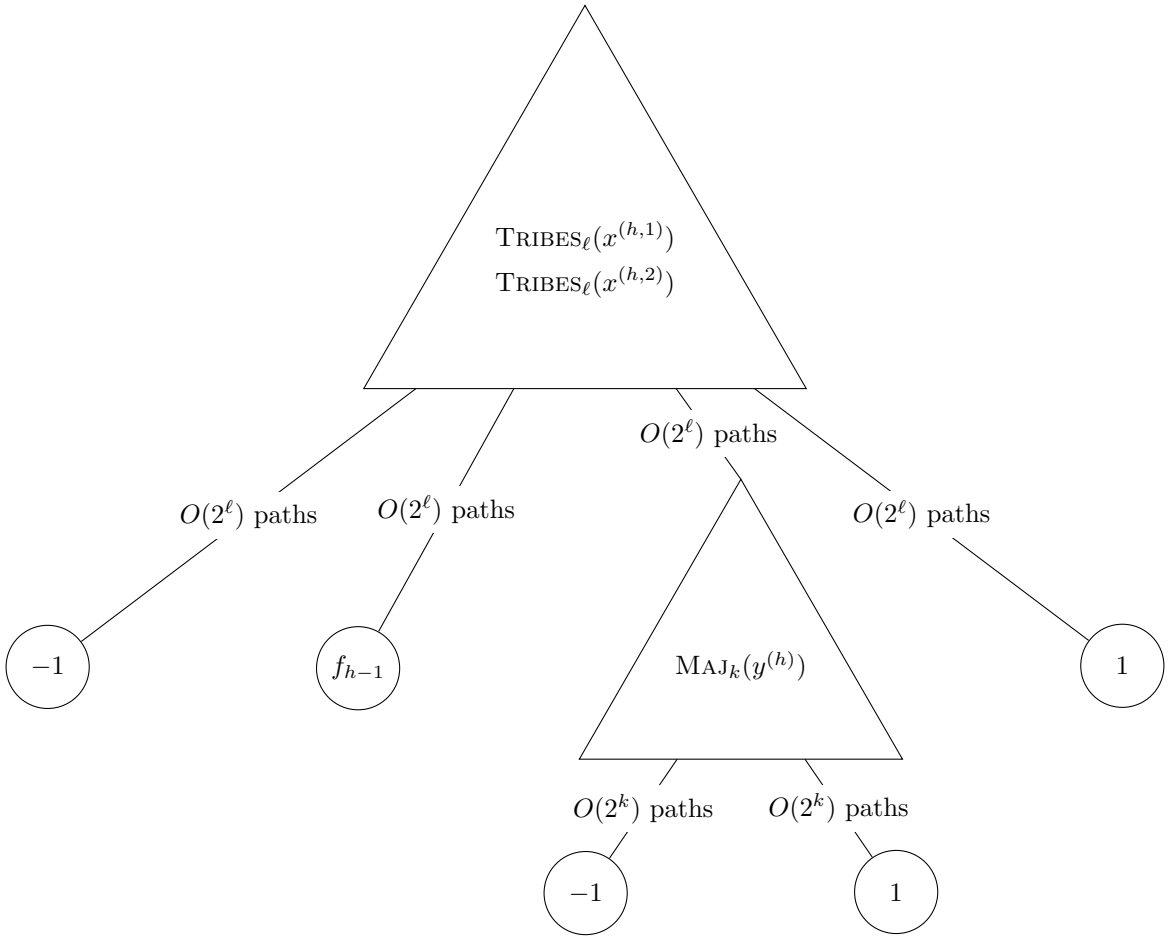

Figure 7: A small decision tree that computes f_h

and for $h \geq 1$,

$$f_h(x,y,z) = \begin{cases} -1 & \text{if } \textsc{Tribes}_{\ell,\delta}(x^{(h,1)}) = \textsc{Tribes}_{\ell,1-\delta}(x^{(h,2)}) = 0 \\ f_{h-1}(x,y,z) & \text{if } \textsc{Tribes}_{\ell,\delta}(x^{(h,1)}) = 0 \text{ and } \textsc{Tribes}_{\ell,1-\delta}(x^{(h,2)}) = 1 \\ \textsc{Maj}_k(y^{(h)}) & \text{if } \textsc{Tribes}_{\ell,\delta}(x^{(h,1)}) = 1 \text{ and } \textsc{Tribes}_{\ell,1-\delta}(x^{(h,2)}) = 0 \\ +1 & \text{otherwise.} \end{cases}$$

Clearly f_h is monotone in $x^{(h,1)}$ and $x^{(h,2)}$. Furthermore, since each of the functions $-1, +1$, and $\textsc{Maj}_k(y^{(h)})$, are monotone, if f_{h-1} is monotone then so is f_h.

Claim 7.2 (Optimal size of f_h)**.** *Choose any integers $\ell, h, r, k > 0$ and let Then, $f_{h,\ell,k,r}$ has optimal decision tree size*

$$\begin{aligned} \text{size}(f_h) &\leq (\text{size}(\textsc{Tribes}_{\ell,\delta}) \cdot \text{size}(\textsc{Tribes}_{\ell,1-\delta}))^{O(h)} \cdot (\text{size}(\textsc{Maj}_k) + \text{size}(\textsc{Tribes}_r)) \\ &\leq 2^{O(h \cdot \ell \log\log \ell / \log \ell)} \cdot (2^k + 2^{O(r \log\log r / \log r)}). \qquad \text{(Fact 6.1)} \end{aligned}$$

Proof. As in the proofs of the previous separations, this upper bound is witnessed by the natural decision tree that one builds by following the definition of f_h. This tree first evaluates $\textsc{Tribes}_{\ell,\delta}(x^{(h,1)})$ followed by $\textsc{Tribes}_{\ell,1-\delta}(x^{(h,2)})$, resulting in a tree of size $(\text{size}(\textsc{Tribes}_{\ell,\delta}) \cdot \text{size}(\textsc{Tribes}_{\ell,1-\delta}))$. At the end of each branch, we either recursively build a tree for f_{h-1}, or a tree for $\textsc{Maj}_k(y^{(h)})$, or place constants $\{\pm 1\}$ as leaves. Please refer to Figure 7. □

The remainder of this section is devoted to lower bounding $\textsc{TopDownDTSize}(f_h, \varepsilon)$, the size of the tree T_{approx} that $\textsc{BuildTopDownDT}$ constructs to ε-approximate f_h.

7.2.1 "Mostly precedes"

By choosing parameters appropriately, we will ensure that when T_{approx} begins by querying the variables of $\textsc{Maj}_k(y^{(h)})$. The first technical challenge that arises is the following: unlike $\textsc{Parity}_k(y^{(h)})$ in our proof of Theorem 4(b), the influence of variables in $\textsc{Maj}_k(y^{(h)})$ changes as variables are queried. For example, the influence of the remaining variables of $\textsc{Maj}_k(y^{(h)})$ after $\frac{k}{2}$ variables have been queried is 0 if all of the queried variables are 1 and is $\Theta(\frac{1}{\sqrt{k}})$ if half of the variables queries are 0 and half are 1. Hence, in T_{approx}, the number of nodes from $y^{(h)}$ queried before some non-$y^{(h)}$-variable is queried will vary by path. (In other words, the analogue of Lemma 6.5 in the proof of Theorem 4(b) is somewhat trickier to establish.)

To handle this, we define the following notion, which will allow us to show that *most* $y^{(h)}$-variables are before other variables in *most* paths of the tree (Corollary 7.5).

Definition 20 (Mostly precedes)**.** Let S be a subset of the relevant variables of f_h. We say that *$y^{(i)}$-variables mostly precede S in T_{approx}* if for every path π in T_{approx} leading to a first query to a variable in S, and every $j \in [k]$,

$$\text{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big) \leq \frac{1}{100\sqrt{k}}.$$

(For some intuition behind Definition 20, we note that *pre-restriction*, the influences of variables in $\textsc{Maj}_k$ are given by:

$$\text{Inf}_j(\textsc{Maj}_k) = \frac{1}{k} \cdot \binom{k}{\frac{k}{2}} \sim \frac{\sqrt{2/\pi}}{\sqrt{k}} \quad \text{for all } j \in [k],$$

which is significantly larger than the $\frac{1}{100\sqrt{k}}$ of Definition 20.) With Definition 20 in hand, we now begin to formalize the structure of T_{approx} as depicted in Figure 8. For each $i \in [h]$, we define

$$R_i = \{x^{(i,1)}, x^{(i,2)}, \text{ and } z \text{ variables}\}.$$

Lemma 7.3. *There is a universal constant c such that the following holds. Suppose*

$$\frac{c}{\sqrt{k}} \geq \frac{1}{\delta^2} \cdot \max\left\{\frac{\delta \log(1/\delta) \log \ell}{\ell}, \frac{\log r}{r}\right\}. \tag{7}$$

Then for all $i \in [h]$, we have that $y^{(i)}$-variables mostly precede R_i in T_{approx}.

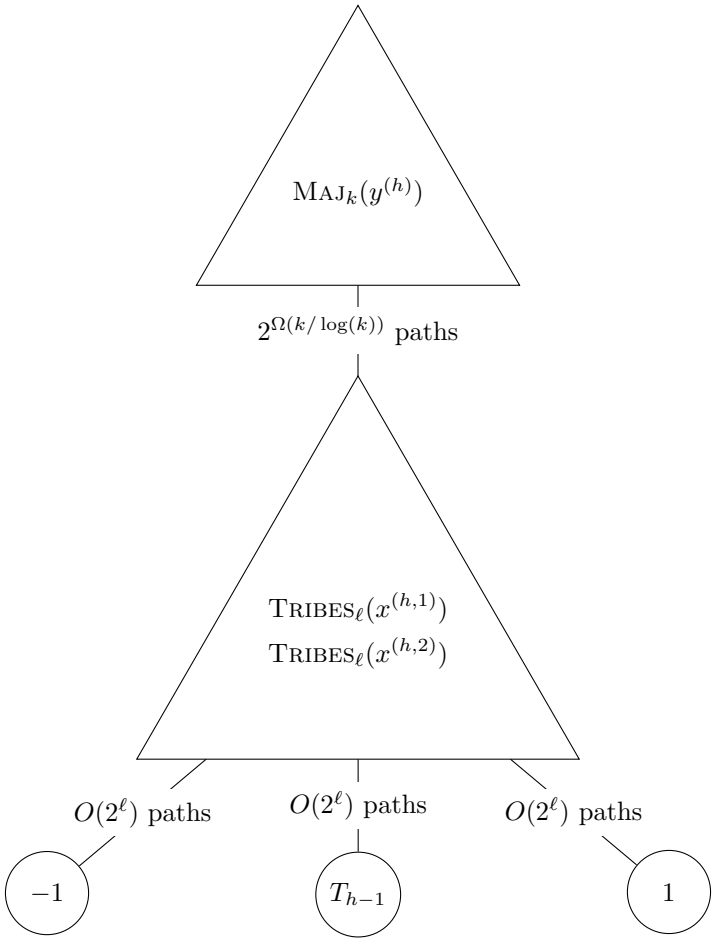

Figure 8: With appropriately chosen parameters, the $y^{(h)}$-variables are the most influential in f_h, so the tree built by BuildTopDownDT will query them first. Our analysis shows that this leads to a tree of size $2^{\Omega(kh/\log k)}$ (cf. Figure 7).

Proof. Fix $i \in [h]$. Let π be a path in T_{approx} that leads to a first query to a variable in $v \in R_i$. Since v is maximally influential in $(f_h)_\pi$, we may apply Corollary 6.4 to infer that v is also maximally influential in $(f_i)_\pi$ (and in particular, v is more influential than any $y^{(i)}$ variable). We have that

$$\begin{aligned}\mathrm{Inf}_v((f_i)_\pi) &\le \max\left\{\mathrm{Inf}_j(\textsc{Tribes}_{\ell,\delta}(x^{(i,1)})), \mathrm{Inf}_j(\textsc{Tribes}_{\ell,1-\delta}(x^{(i,2)})), \mathrm{Inf}_j(\textsc{Tribes}_r(z))\right\} \\ &\le \max\left\{(2+o(1))\cdot\delta\log(1/\delta)\cdot\frac{\log\ell}{\ell}, (1+o(1))\cdot\frac{\ln r}{r}\right\}. \qquad \text{(Fact 6.1 and Fact 7.1)}\end{aligned}$$

On the other hand, for any $j \in [k]$,

$$\begin{aligned}\mathrm{Inf}_{y_j^{(i)}}((f_i)_\pi) &= \Pr\left[\textsc{Tribes}_{\ell,\delta}(\boldsymbol{x}^{(h,1)}) = 1 \text{ and } \textsc{Tribes}_{\ell,1-\delta}(\boldsymbol{x}^{(h,2)}) = 0\right]\cdot\mathrm{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big) \\ &= \delta^2\cdot\mathrm{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big).\end{aligned}$$

Since $\mathrm{Inf}_{y_j^{(i)}}((f_i)_\pi) \le \mathrm{Inf}_v((f_i)_\pi)$, the bounds above imply that

$$\mathrm{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big) \le \frac{1}{\delta^2} \cdot \max\left\{(2+o(1)) \cdot \delta \log(1/\delta) \cdot \frac{\log \ell}{\ell}, (1+o(1)) \cdot \frac{\ln r}{r}\right\}.$$

The lemma follows: by choosing c to be a sufficiently small constant in Equation (7), we can ensure that $\mathrm{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big) \le \frac{1}{100\sqrt{k}}$. □

Lemma 7.4. *There is a universal constant c such that the following holds. Fix $i \in [h]$ and consider a uniform random $\boldsymbol{y}^{(i)} \in \{0,1\}^k$. The probability there is an input u to f_h consistent with $\boldsymbol{y}^{(i)}$ such that T_{approx}, on input u, queries an R_i-variable before a querying at least $ck/\log k$ many $\boldsymbol{y}^{(i)}$-variables is $O(k^{-2})$.*

Proof. Fix an outcome $y^{(i)}$ of $\boldsymbol{y}^{(i)}$. Suppose that there is an input u consistent with $y^{(i)}$ such that T_{approx}, on input u, queries an R_i-variable after querying only $< ck/\log k$ many $y^{(i)}$-variables. Call such a $y^{(i)}$ outcome *bad*, and let π denote the corresponding path in T_{approx} that leads to the first query to an R_i-variable. Since $y^{(i)}$-variables mostly precede R_i in T_{approx}, we have that

$$\mathrm{Inf}_j\big(\textsc{Maj}_k(y^{(i)})_\pi\big) \le \frac{1}{100\sqrt{k}} \quad \text{for all } j \in [k].$$

For this to hold, it must be the case that among the $t < ck/\log k$ many $y^{(i)}$-variables that occur in π, the discrepancy between the number of 0's and 1's is $\Omega(\sqrt{k})$. We can therefore bound

$$\begin{aligned}
\Pr_{\boldsymbol{y}^{(i)} \in \{0,1\}^k}[\boldsymbol{y}^{(i)} \text{ is bad}] &\le \sum_{t=1}^{ck/\log k} \Pr_{\boldsymbol{b} \sim \mathrm{Bin}(t,\frac{1}{2})}\left[|\boldsymbol{b} - \tfrac{t}{2}| \ge \Omega(\sqrt{k})\right] \\
&\le \sum_{t=1}^{ck/\log k} e^{-\Theta(k/t)} && \text{(Hoeffding's inequality)} \\
&\le \frac{ck}{\log k} \cdot e^{-\Theta((\log k)/c)} \ll \frac{1}{k^2},
\end{aligned}$$

where the final inequality holds by choosing c to be a sufficiently small constant. □

By a union bound over $i \in [h]$, we have the following corollary of Lemma 7.4 (which can be thought of as being roughly analogous to Lemma 6.5 in the proof of Theorem 4(b)):

Corollary 7.5 (Most queries to z-variables are deep within T_{approx})**.** *Let $\boldsymbol{y} = (\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(k)})$ be uniform random. The probability that there is an input u to f_h consistent with $\boldsymbol{y}$ such that T_{approx}, on input u, queries a z-variable before querying at least $(ck/\log k) \cdot h$ many $\boldsymbol{y}$-variables is $O(h/k^2)$.*

We are finally ready to lower bound the size of T_{approx}:

Claim 7.6 (Lower bound on the size of T_{approx})**.** *Fix $\varepsilon \in (0, \frac{1}{2})$ and let $c = (\frac{1}{2} - \varepsilon)/2$. If*

$$(1-\delta)^{2h} \ge (2+c)\varepsilon \tag{8}$$

$$h \le k, \tag{9}$$

then the size of T_{approx} is at least $2^{\Omega(hk/\log k)}$.

Proof. We will call an input to f_h *z-dependent* if it satisfies:

$$\textsc{Tribes}_{\ell,\delta}(x^{(i,1)}) = 0 \text{ and } \textsc{Tribes}_{\ell,1-\delta}(x^{(i,2)}) = 1 \text{ for all } i \in [h].$$

Note that the output of f_h on any z-dependent input is $\textsc{Tribes}_r(z)$. Let us define $\zeta(y^{(1)}, \ldots, y^{(h)})$ to be the $\{0,1\}$-valued indicator of whether there is an input u consistent with $y^{(1)}, \ldots, y^{(h)}$ such that T_{approx} on input u queries a z-variable. For any fixed $y = (y^{(1)}, \ldots, y^{(h)})$,

- If $\zeta(y) = 0$, then T_{approx} must assign the same -1 or $+1$ value to all z-dependent inputs consistent with y;
- The fraction of z-dependent inputs that are consistent with y is

  $$\Pr\left[\textsc{Tribes}_{\ell,\delta}(\boldsymbol{x}^{(i,1)}) = 0 \text{ and } \textsc{Tribes}_{\ell,1-\delta}(\boldsymbol{x}^{(i,2)}) = 1 \text{ for all } i \in [h]\right] = (1-\delta)^{2h},$$

  which is at least $(2+c)\varepsilon$ by Equation (8). Furthermore, since output of f_h on any z-dependent input is $\textsc{Tribes}_r(z)$, among the z-dependent inputs that are consistent with y, we have that f_h labels half of them -1 and half of them $+1$.

Therefore,

$$\text{error}(T_{\text{approx}}, f_h) \geq \tfrac{1}{2} \cdot (2+c)\varepsilon \cdot \Pr[\zeta(\boldsymbol{y}) = 0].$$

Since $\text{error}(T_{\text{approx}}, f_h) \leq \varepsilon$, it follows that $\Pr[\zeta(\boldsymbol{y}) = 1] \geq \Omega(1)$. Next, applying Corollary 7.5 along with our assumption that $h \leq k$ (Equation (9)), we further have that

$$\Pr_{\boldsymbol{y}}\left[\exists\ \boldsymbol{y}\text{-consistent } u \text{ s.t. } T_{\text{approx}}(u) \text{ queries } z\text{-variable after} \geq \frac{ck}{\log k} \cdot h \text{ many } \boldsymbol{y}\text{-variables}\right] \geq \Omega(1).$$

On the other hand, for any fixed path π in T_{approx} that queries $\geq \Omega(kh/\log k)$ many y-variables, at most a $2^{-\Omega(kh/\log k)}$ fraction of $\boldsymbol{y}$'s can be consistent with this specific π. We conclude that the size of T_{approx} must be at least $2^{\Omega(kh/\log k)}$, and the proof is complete. □

Theorem 6(b) now follows from Claim 7.2 and Claim 7.6 by setting parameters appropriately:

Proof of Theorem 6(b). By choosing

$$\begin{aligned}
\delta &= \Theta\left(\sqrt[3]{(\log \ell)^4 \log(1/\varepsilon)/\ell}\right) \\
k &= \Theta\left(\sqrt[3]{\ell^4 \log(1/\varepsilon)^2/(\log \ell)^4}\right) \\
r &= \Theta(k) \\
h &= \Theta\left(\frac{1}{\delta} \cdot \log(1/\varepsilon)\right),
\end{aligned}$$

we satisfy Equations (7) to (9). We may therefore apply Claim 7.2 to get that the optimal size of f_h is upper bounded by:

$$\text{size}(f_h) \leq \exp\left(O(\sqrt[3]{\ell^4 \log(1/\varepsilon)^2/(\log \ell)^4})\right).$$

On the other hand, by Claim 7.6, we have that

$$\textsc{TopDownDTSize}(f_h, \varepsilon) \geq 2^{\Omega(kh/\log k)} = \exp\left(\Omega(\sqrt[3]{\ell^5 \log(1/\varepsilon)^4/(\log \ell)^{11}})\right).$$

This is a separation of s versus $s^{\tilde{\Omega}(\sqrt[4]{\log(s)})}$. □

Remark 21. For our choice of parameters above, we have that $s(n) = \text{size}(f_h) = 2^{\tilde{\Theta}(n^{4/5})}$, where $n = h(2\ell + k) + r$ is the number of variables of f_h. A standard padding argument yields the same s versus $s^{\tilde{\Omega}(\sqrt[4]{\log s})}$ separation for any function $s(n) \leq 2^{\tilde{O}(n^{4/5})}$.

Remark 22 (Depth separation)**.** The same proof witnesses a separation of d versus $\tilde{\Omega}_\varepsilon(d^{5/4})$ for the optimal *depth* of f_h versus the depth of the tree that BuildTopDownDT(f_h, ε) builds. This disproves the conjecture of Fiat and Pechyony [FP04] discussed in Section 2, which states that BuildTopDownDT builds a tree of optimal depth for all monotone functions, even in the case of exact representation ($\varepsilon = 0$).

7.3 Lower bounds for all impurity-based heuristics

Proposition 7.7 (Splitting on the most influential variable of a monotone function maximizes purity gain)**.** *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be a monotone boolean function.*[18] *Let $\mathscr{G} : [-1, 1] \to [0, 1]$ be a concave function that is symmetric around 0, and satisfies $\mathscr{G}(-1) = \mathscr{G}(1) = 0$ and $\mathscr{G}(0) = 1$. Suppose $i \in [n]$ maximizes:*

$$\mathscr{G}(\mathbb{E}[f]) - \tfrac{1}{2}(\mathscr{G}(\mathbb{E}[f_{x_i=-1}]) + \mathscr{G}(\mathbb{E}[f_{x_i=1}])), \tag{10}$$

Then $\mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_i] \geq \mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_j]$ for all $j \in [n]$.

Proof. For all functions f, not necessarily monotone, $\mathbb{E}[f]$ is precisely the average of $\mathbb{E}[f_{x_i=0}]$ and $\mathbb{E}[f_{x_i=1}]$. Because $\mathscr{G}$ is concave everywhere on its domain, Jensen's inequality ensures that $\mathscr{G}(\mathbb{E}[f])$ is greater than $\frac{1}{2}(\mathscr{G}(\mathbb{E}[f_{x_i=0}]) + \mathscr{G}(\mathbb{E}[f_{x_i=1}])$. Furthermore, again by concavity, we have that this difference increases with the difference between $\mathbb{E}[f_{x_i=1}]$ and $\mathbb{E}[f_{x_i=-1}]$. Therefore the variable $i \in [n]$ that maximizes purity gain (10) also maximizes $|\mathbb{E}[f_{x_i=1}] - \mathbb{E}[f_{x_i=-1}]|$.

For a monotone function $f : \{\pm 1\}^n \to \{\pm 1\}$, we have the following identity for all variables $j \in [n]$:

$$\begin{aligned}\text{Inf}_j(f) &= \Pr[f(\boldsymbol{x}) \neq f(\boldsymbol{x}^{\oplus j})] \\ &= \mathbb{E}[f(\boldsymbol{x})\boldsymbol{x}_j] \\ &= \mathbb{E}[f_{x_j=1}] - \mathbb{E}[f_{x_j=-1}].\end{aligned}$$

Thus, the variable that maximizes purity gain (10) is also the most influential variable of f. □

Recall that in Theorem 7, we claimed that our lower bound on TopDownDTSize(f_h, ε) holds not just for the specific algorithm BuildTopDownDT, but in fact *any* impurity-based top-down heuristic. To see this, note that in our proof of Theorem 6(b) described in Section 7.2, we never used any information about *which* leaf BuildTopDownDT chooses to split on at each stage, only that when a leaf *is* split, it is replaced by the most influential variable of the corresponding subfunction. In other words, just like our proof of Theorem 4(b), our proof of Theorem 6(b) applies not just to the specific tree build by BuildTopDownDT(f_h, ε); it in fact lower bounds the size of *any* pruning T_{approx} of $T_{\text{exact}} =$ BuildTopDownDT$(f, \varepsilon = 0)$ that is an ε-approximator to f_h.

By Proposition 7.7, any tree build by a impurity-based top-down heuristic is a pruning of T_{exact}, and hence our proof of Theorem 6(b) extends to establish Theorem 7.

[18] For this proof, for notational reasons it will be slightly more convenient for us to work with $\{\pm 1\}^n$ instead of $\{0, 1\}^n$ as the domain of f.

8 New proper learning algorithms: Proofs of Theorem 8 and Theorem 9

Recall that BuildTopDownDT builds an approximation to f iteratively. It starts with an empty bare tree T° and repeatedly replaces the leaf with the highest score with a query to that leaf's most influential variable. In section Section 5.2, we proved lower bounds on the score of the leaf that BuildTopDownDT selects. Using those lower bounds, in section Section 5.3, we are able to prove upper bounds on the size of the tree BuildTopDownDT needs to produce to guarantee at most ε error. If, instead, we only guaranteed that we would pick a leaf with score a fourth of that guaranteed by the lower bounds in Section 5.2, all of our upper bounds would still hold, up to constant factors in the exponent. In this section, we will show that it is possible to accurately enough estimate influences to guarantee we pick a leaf with score at least a fourth the maximum score. First, we provide a definition of score that takes into account both the leaf and the variable selected.

Definition 23 (score)**.** Given any function f, we define the score of a leaf ℓ and variable i as follows.

$$\text{score}(\ell, i) := \Pr_{\boldsymbol{x}\sim\{0,1\}^n}[\,\boldsymbol{x} \text{ reaches } \ell\,] \cdot \text{Inf}_i(f_\ell) = 2^{-|\ell|} \cdot \text{Inf}_i(f_\ell).$$

We first show that it is possible to estimate scores sufficiently accurately for monotone functions just from random samples of a function, which proves Theorem 9. Let $\boldsymbol{S}$ be a random sample from a monotone function f, and recall Fact 2.1. We can estimate the score of a particular leaf and variable as follows.

$$\text{score}(\ell, i, \boldsymbol{S}) = \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\in\boldsymbol{S}}\big[\mathbb{1}[\boldsymbol{x} \text{ reaches } \ell] \cdot f(\boldsymbol{x})(2\boldsymbol{x}_i - 1)\big].$$

Note that $\mathbb{E}_{\boldsymbol{S}}[\text{score}(\ell, i, \boldsymbol{S}))] = \text{score}(\ell, i)$. Let t be any score threshold and m be the number of examples in $\boldsymbol{S}$. By Chernoff bounds, for any particular leaf ℓ and variable x_i,

$$\Pr_{\boldsymbol{S}}\left[\text{score}(\ell, i, \boldsymbol{S}) \le \tfrac{t}{2}\right] \le e^{-\frac{1}{8}t\cdot m} \quad \text{if score}(\ell, i) \ge t$$
$$\Pr_{\boldsymbol{S}}\left[\text{score}(\ell, i, \boldsymbol{S}) \ge \tfrac{t}{2}\right] \le e^{-\frac{1}{12}t\cdot m} \quad \text{if score}(\ell, i) \le t/4.$$

At step j in BuildTopDownDT, there are $j+1$ leaves in T°. If t is the maximum score possible at that step, with probability at least $1-(j+1)e^{-\frac{1}{12}t\cdot m}$, the leaf and variable with maximum empirical score will have true score at least $\frac{t}{4}$. By Lemma 5.2, BuildTopDownDT(f, ε), at step j, there will always be a leaf with score at least $\frac{\varepsilon}{(j+1)\log(s)}$, where s is the decision tree size of f. Hence, the maximum empirical score will have true score at least $\frac{1}{4}$ the optimal value with probability at least $1-(j+1)e^{-\frac{\varepsilon\cdot m}{12(j+1)\log(s)}}$.

The probability that selecting the maximum empirical score is always within $\frac{1}{4}$ of the optimal value for every step from $j=0$ to $j=k-1$ is at least $1-k^2e^{-\frac{\varepsilon\cdot m}{12k\log(s)}}$. By setting the sample size to

$$m = O\left(\frac{k\log s}{\varepsilon}(\log k + \log(1/\delta))\right) \tag{11}$$

with probability at least $1-\delta$, we choose a sufficiently good leaf for k steps of BuildTopDownDT. Recall that, for monotone functions, BuildTopDownDT builds a decision tree of size at most

$$k = \min(s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}, s^{O(\sqrt{\log s}/\varepsilon)}).$$

Hence, with probability at least $1-\delta$, taking $\min(s^{O(\log(s/\varepsilon)\log(1/\varepsilon))}, s^{O(\sqrt{\log s}/\varepsilon)})\log(1/\delta)$ is enough to learn to accuracy ε. This proves Theorem 9.

If f is not monotone, we cannot accurately estimate influences from just random samples. However, we can estimate influences if given access to *random edges* from f.

Definition 24 (Random edges)**.** For any function $f : \{0,1\}^n \to \{\pm 1\}$, a *random edge* is two points of the form $((\boldsymbol{x}, f(\boldsymbol{x})), (\boldsymbol{x}^{\oplus \boldsymbol{i}}, f(\boldsymbol{x}^{\oplus \boldsymbol{i}})))$, where $\boldsymbol{x} \in \{0,1\}^n$ and $\boldsymbol{i} \in [n]$ are both picked uniformly at random. A *random edge sample* $\boldsymbol{E}$ is a collection of random edges. Given random edge sample $\boldsymbol{E}$, we will use $\boldsymbol{E}_i$ to refer to all those edges in $\boldsymbol{E}$ in which the i^{th} bit of x is flipped.

Given a random edge sample $\boldsymbol{E}$ of a function f, we will be able to accurately estimate influences of the variables in f, and learn f using BuildTopDownDT. We use the following estimate of score:

$$\text{score}(\ell, i, \boldsymbol{E}) = \mathop{\mathbb{E}}_{((\boldsymbol{x}_1,\boldsymbol{y}_1),(\boldsymbol{x}_2,\boldsymbol{y}_2))\in \boldsymbol{E}_i} \left[\mathbb{1}[\boldsymbol{x}_1 \text{ and } \boldsymbol{x}_2 \text{ reach } \ell] \cdot \mathbb{1}[\boldsymbol{y}_1 \neq \boldsymbol{y}_2]\right]$$

If we desire there to be m samples in each $\boldsymbol{E}_i$ with probability at least $1-\delta$, then having $\boldsymbol{E}$ by size $O(n \cdot (m + \log(\frac{1}{\delta}))$ is sufficient, where m is as defined in Equation (11). Since one can certainly general a random edge sample $\boldsymbol{E}$ if given membership query access to f, this proves Theorem 8.

Learning trees with small average depth: Theorem 10. Let f be a monotone function computed by a decision tree T of average depth $\triangle$.

1. We first observe that the total influence of f is at most $\triangle$. To see this, first recall that $\text{Inf}(f) = \mathbb{E}[\text{sens}_f(\boldsymbol{x})]$, where $\text{sens}_f(x) = |\{i \in [n] \colon f(x) \neq f(x^{\oplus i})\}|$, i.e. that total influence is equivalent to average sensitivity. For any x, the sensitivity of f at x is at most the depth of the path that x follows in T, and hence the average sensitivity of f is at most the average depth $\triangle$ of T.

2. Recall Theorem 13, which says that monotone functions with decision tree size s have total influence at most $\sqrt{\log s}$. In fact, [OS07] proves a stronger statement: if f is monotone, then it has total influence at most $\sqrt{\triangle}$. (This is indeed a stronger statement because $\triangle \leq \log s$.)

3. Similarly, [OSSS05] also establishes a stronger version of Theorem 12, showing that f has a variable of influence at least $\text{Var}(f)/\triangle$ (rather than just $\text{Var}(f)/\log s$). Hence an equivalent statement to Lemma 5.2 holds, where BuildTopDownDT selects a leaf with score at least $\frac{\varepsilon}{(j+1)\triangle}$.

Combining these observations, with the same proof as for Theorem 5, we get that BuildTopDownDT produces a tree of size $2^{O(\triangle^2/\varepsilon)}$, and if T is monotone, size only $2^{O(\triangle^{3/2}/\varepsilon)}$. Then, for the same reasons as Theorems 8 and 9 hold, Theorem 10 holds.

9 Proper learning with polynomial sample and memory complexity

In this section we give a quasipolynomial-time algorithm for properly learning decision trees under the uniform distribution, where sample and memory of our algorithm are both polynomial. To our knowledge, this is the first algorithm for properly learning decision trees that achieves polynomial memory complexity. (Recall Table 1.)

Background: Ehrehfeucht–Haussler and Mehta–Raghavan. At the core of most learning algorithms is an algorithm that achieves low error on a set of samples. We will use the following notation in this section:

Notation: A sample, S, is a set of examples of the form (x, y) where $x \in \{0,1\}^n$ and $y \in \{-1, 1\}$. The error of a decision tree, T, with respect to the samples is

$$\text{error}_S(T) = \Pr_{\boldsymbol{x},\boldsymbol{y}\in S}[T(\boldsymbol{x}) \neq \boldsymbol{y}].$$

We say that a set of samples is *exactly fit* by a tree of size s if there exists a zero-error tree with size at most s. Furthermore, we use S_0^v and S_1^v to refer to all the points in the sample S where the variable corresponding to v is 0 and 1 respectively. Lastly, all learning statements in this section are with respect to the uniform distribution.

Ehrenfeucht and Haussler's algorithm makes the following guarantee:

Theorem 25 (Algorithmic core of [EH89])**.** *There is an algorithm that takes in a set of samples, S, over n variables that can be exactly fit by a decision tree of size s and returns a tree of size at most $n^{\log(s)}$ that exactly fits S. Furthermore, that algorithm runs in time $|S| \cdot n^{O(\log s)}$.*

One downside of their algorithm is that it leads to a large hypothesis class—the class of all decision trees of size $n^{\log s}$—so in order to generalize with high probability, they require $\text{poly}(n^{\log s}, \frac{1}{\varepsilon})$ samples.

Mehta and Raghavan observe that if a function is computable by a tree of size s, then it is also ε-approximated by a tree of depth at most $\log(\frac{s}{\varepsilon})$. They combine this observation with a new algorithm that makes the following guarantee:

Theorem 26 (Algorithmic core of [MR02])**.** *There is an algorithm that takes a sample, S, over n variables as well as budgets for size s and depth d and returns the decision tree of size at most s and depth at most d with minimal error on S.*[19] *Furthermore, the algorithm runs in time $n^{O(d)} \cdot (s^2 + |S|)$.*

Importantly, there are only $2 \cdot (4n)^s$ decision trees of size at most s, a much smaller hypothesis class than for Ehrenfeucht and Haussler's algorithm. As a result, they only need $\text{poly}(s, \frac{1}{\varepsilon}) \cdot \log n$ samples to generalize with high probability. A downside of their work, relative to Ehrenfeucht and Haussler's, is that they need to set $d = O(\log(\frac{s}{\varepsilon}))$, so their algorithm has a runtime of approximately $n^{O(\log(s/\varepsilon))}$ instead of $n^{O(\log s)}$.

Neither [EH89] nor [MR02] are able to learn decision trees with only $\text{poly}(n, s, \frac{1}{\varepsilon})$ memory. [EH89] uses a sample of size approximately $n^{O(\log s)}$ to guarantee generalization, and their algorithm

[19] They also guarantee that if their are multiple trees with minimal error, they return a tree with minimal size among those with minimal error.

must store all of the samples, so it needs at least that much memory. [MR02] use a dynamic programming algorithm that stores computation for each restriction of the n variables of size at most $d = O(\log(\frac{s}{\varepsilon}))$. There are $\binom{n}{d} \cdot 2^d = \Theta\binom{n}{\log(s/\varepsilon)}$ such restrictions, resulting in superpolynomial memory complexity.

Our algorithm: proper learning with polynomial sample and memory complexity. We introduce a new algorithm that makes more assumptions about its input than either [EH89]'s or [MR02]'s algorithms. It requires the samples it receives to be *well-distributed*, a property we will later define (Definition 28). In exchange, it only uses polynomial memory. The following should be contrasted with Theorems 25 and 26:

Theorem 27 (Core of our algorithm)**.** *There is an algorithm (Figure 9) that when given a depth budget d and a well-distributed sample S that can be exactly fit by a tree of size s returns a tree with depth at most d and error at most $(\frac{3}{4} + o(1))^d \cdot s$ on the samples. Furthermore, the algorithm runs in time $|S| \cdot n^{O(\log s)}$ and uses $\mathrm{poly}(2^d, \log n, |S|)$ memory.*

Note that if the goal is to learn the sample to accuracy ε, we can set $d = O(\log(\frac{s}{\varepsilon}))$. The result is an algorithm that runs in time $|S| \cdot n^{O(\log s)}$ and uses memory $\mathrm{poly}(n, s, 1/\varepsilon, |S|)$. Furthermore, the well-distributed requirement turns out to be true for nearly all uniformly random samples that are sufficiently large. The result is, to the best of our knowledge, the first polynomial memory proper learning algorithm for decision trees.

Our algorithm (Figure 9) is a surprisingly simple modification of [EH89]'s algorithm, but our analysis is quite a bit more involved. A key difference is that [EH89]'s algorithm is an Occam algorithm, whereas ours is not. The original [EH89] algorithm breaks down when in cannot fully fit the sample; the analysis showing that our algorithm *is* able to handle a sample it cannot fully fit is subtle.

Lemma 9.1 (Correctness of Find)**.** *If S can be exactly fit by a tree of size s, then* Find(S, s, d) *will not return "None."*

Proof. By induction. If $s = 1$ and S can be fit by a tree of size s, then all samples in S will have the same label. Hence, Find will return a tree on line 1, and not return "None".

For $s \geq 2$, there are only two spots where Find could return "None":

Line 4.c.ii Find returns "None" on line 4.c.ii only if a call of the form Find$(S^v_a, s-1, d-1)$ returns "None" where $a = \pm 1$ and v is relevant. Let T_S be a minimal size tree that fits S, which by assumption, has size at most s. Since v is relevant, a node labeled with it must appear somewhere in that tree. This means that there is a size $s-1$ tree that will exactly fit S^v_a. By induction, this means that Find$(S^v_a, s-1, d-1)$ will not return "None."

Line 5. Find returns "None" on line 5 only if there was not a single relevant variable v for which either of the calls Find$(S^v_0, \frac{s}{2}, d-1)$ or Find$(S^v_1, \frac{s}{2}, d-1)$ on line 4.a succeeded (i.e. did not return "None"). Once again, let T_S be a minimal size tree that fits S. Then, every node in T_S must be relevant for S. Furthermore, T_S has some root variable v^* and subtrees $(T_S)_0$ and $(T_S)_1$. At least one of $(T_S)_0$ or $(T_S)_1$ must have size at most $\frac{s}{2}$. If $(T_S)_0$ has size at most $\frac{s}{2}$, then by the inductive hypothesis, Find$(S^v_0, \frac{s}{2}, d-1)$ does not return "None." Otherwise Find$(S^v_1, \frac{s}{2}, d-1)$ does not return "None." Hence, Find won't return "None" on line 5. □

FIND(S, s, d):

Input: A sample S that can be exactly fit by a tree of size s and depth budget d.

Output: A decision tree T with depth at most d that approximately fits S. If S cannot be fit by a tree of size s, may return "None."

1. If all samples in S have the same label, return the single-node tree computing that label.
2. If $s \leq 1$ return "None."
3. If $d = 0$ return the single-node tree computing the majority label of S.
4. For each relevant[a] variable v:
   (a) Let $T_0^v = \text{FIND}(S_0^v, \frac{s}{2}, d-1)$ and $T_1^v = \text{FIND}(S_1^v, \frac{s}{2}, d-1)$.
   (b) If both T_0^v and T_1^v are not "None", return the tree with root labeled v, 0-subtree T_0^v and 1-subtree T_1^v.
   (c) If one of T_0^v and T_1^v is "None" and the other is not:
      i. Reexecute the recursive call for the side that was "None" with size $s-1$ instead of size $\frac{s}{2}$. For example, if T_1^v was "None", set $T_1^v = \text{FIND}(S_1^v, s-1, d-1)$
      ii. If the reexecuted call still returns "None", return "None."
      iii. Return the tree with root labeled v, 0-subtree T_0^v and 1-subtree T_1^v.
5. Return "None".

[a] "relevant" means that neither S_0^v nor S_1^v is empty

Figure 9: Our variant of Ehrenfeucht and Haussler's FIND algorithm.

We hope to prove that FIND will produce low error trees, but it turns out to be difficult to guarantee this for arbitrary samples. One particular sample we can guarantee this for is the sample containing all possible points. If S contains all 2^n possible points, then FIND will return a tree with error at most $\frac{1}{4} \cdot (\frac{3}{4})^d$, which we will prove in Lemma 9.2. The following property allows us to quantify how close S is to the full sample.

Definition 28 (Well-distributed samples)**.** We say that a sample of points S is *c-well-distributed* to depth d if, for any restriction α where $|\alpha| \leq d$,

$$||(S_\alpha)| - \mu| \leq c\mu$$

where $\mu = 2^{-|\alpha|} \cdot |\boldsymbol{S}|$ is the expected size of $\boldsymbol{S}_\alpha$ if $\boldsymbol{S}$ is chosen uniformly at random.

For example, if S contains all possible 2^n points, then S is 0-*well-distributed* to any depth.

Lemma 9.2 (Error of FIND on well-distributed samples)**.** *Let S be c-well-distributed to depth d. If* FIND(S, s, d) *does not return "None," it returns a tree with error at most $\frac{1}{4}(\frac{3}{4} + \frac{c}{4})^d \cdot s$ with respect to S.*

Proof. By induction on the d; If $d = 0$ and $s \geq 2$, then this lemma requires the error to be less than $\frac{1}{2}$, which FIND satisfies since it places the majority node. If $s = 1$ and FIND doesn't return "None," it must have returned a zero-error tree on Line 1, satisfying the desired error bound.

Next, consider $d \geq 1$. If all samples have the same label, FIND returns a 0 error tree. Otherwise, it returns a tree, T, with 0-subtree T_0^v and 1-subtree T_1^v for some variable v. Let ℓ_0 and ℓ_1 be the number of points in S_0^v and S_1^v respectively. Then, we can relate the errors of the trees as follows:

$$\text{error}(T) = \frac{\ell_0}{\ell_0 + \ell_1} \cdot \text{error}(T_0^v) + \frac{\ell_1}{\ell_0 + \ell_1} \cdot \text{error}(T_1^v)$$

At least one of T_0^v and T_1^v was generated using a recursive call to FIND with size parameter $\frac{s}{2}$. Without loss of generality, let that tree be T_0^v. The other tree, T_1^v was generated by a recursive call with size at most s. By the inductive hypothesis,

$$\text{error}(T) \leq \frac{\ell_0}{\ell_0 + \ell_1} \cdot \frac{1}{4}\left(\frac{3}{4} + \frac{c}{4}\right)^d \cdot \frac{s}{2} + \frac{\ell_1}{\ell_0 + \ell_1} \cdot \frac{1}{4}\left(\frac{3}{4} + \frac{c}{4}\right)^d \cdot s \tag{12}$$

Since S is c-well-distributed to depth d, $(1-c)\mu \leq \ell_0, \ell_1 \leq (1+c)\mu$, where $\mu = \frac{|S|}{2}$. Choosing $\ell_0 = (1-c)\mu$ and $\ell_1 = (1+c)\mu$ maximizes equation Equation (12) and so results in a valid upper bound.

$$\begin{aligned}
\text{error}(T) &\leq \frac{\mu(1-c)}{2\mu} \cdot \frac{1}{4}\left(\frac{3}{4} + \frac{c}{4}\right)^{d-1} \cdot \frac{s}{2} + \frac{\mu(1+c)}{2\mu} \cdot \frac{1}{4}\left(\frac{3}{4} + \frac{c}{4}\right)^{d-1} \cdot s \\
&= \frac{1}{4} \cdot \left(\frac{3}{4} + \frac{c}{4}\right)^{d-1} \cdot \left(\frac{3}{4} + \frac{c}{4}\right) \cdot s \\
&= \frac{1}{4}\left(\frac{3}{4} + \frac{c}{4}\right)^{d} \cdot s
\end{aligned}$$

□

The above Lemma shows that if a sample is sufficiently well-distributed, FIND will return a tree with low error. We next show that, with high probability, sufficiently large samples will be well-distributed.

Lemma 9.3 (Well-distributed samples are common)**.** *Choose any $0 < c < 1.0, \delta > 0$. Then for*

$$m = O\left(\frac{2^d}{c^2} \cdot (d\log(n) + \log(1/\delta))\right),$$

a sample of size m chosen uniformly at random from $\{0,1\}^n$ is c-well-distributed to depth d with probability at least $1 - \delta$

Proof. Consider an arbitrary restriction α of length $h \leq d$. By Chernoff bounds,

$$\Pr[||\boldsymbol{S}_\alpha| - \mu| \geq c\mu] \leq 2e^{-\mu c^2/3}$$

where $\mu = \mathbb{E}[|\boldsymbol{S}_\alpha|] = m \cdot 2^{-h}$. Since $h \leq d$, we can upper bound the probability as follows.

$$\Pr[||\boldsymbol{S}_\alpha| - \mu| \geq c\mu] \leq 2e^{-m \cdot 2^{-d} c^2/3}$$

$\boldsymbol{S}$ is c-well-distributed if $||\boldsymbol{S}_\alpha| - \mu| \leq c\mu$ for all possible restrictions α of size at most d. There are $\sum_{i=0}^{d} \binom{n}{i} 2^i = n^{O(d)}$ such restrictions. Thus, by a union bound:

$$\Pr[\boldsymbol{S} \text{ is } c\text{-well-distributed}] \geq 1 - n^{O(d)} \cdot e^{-m \cdot 2^{-d} c^2/3}.$$

We set the right-hand side of the above equation to be at least $1 - \delta$ and solve for m, which proves this Lemma. □

Our analysis of the time complexity of FIND is very similar to Ehrenfeucht and Haussler's:

Lemma 9.4 (Time complexity of FIND)**.** FIND(S, s, d) *takes time* $|S| \cdot (n+1)^{2\log(s)}$.

Proof. Fix a total number of variables n and sample size m. Let $T(i, s)$ be the maximum time needed by FIND(S, s, d) where S has size at most m, i is the number of relevant variables in S, and d is arbitrary.

If $i = 0$ or $s = 1$, then FIND must return on Line 1 or 2, using only $O(m)$ time. Otherwise, the FIND makes at most $2i$ recursive calls on line 4.(a) each of which takes time at most $T(i-1, \frac{s}{2})$ It also makes zero or one recursive call on line 4.(c).i which takes time up to $T(i-1, s-1) \leq T(i-1, s)$. In addition to these recursive calls, all of the auxiliary computations can be done in $O(mn)$ time. Hence, we have the following recurrence relation:

$$T(i, s) \leq 2i \cdot T\big(i-1, \tfrac{s}{2}\big) + T(i-1, s) + O(mn).$$

If we substitute $r = \log(s)$, then equivalently, we have the relation:

$$\tilde{T}(i, r) \leq 2i \cdot \tilde{T}(i-1, r-1) + \tilde{T}(i-1, r) + O(mn).$$

The above relation is shown to be upper bounded by $\tilde{T}(i, r) = O(m \cdot (n+1)^{2r})$ in [EH89]. Substituting back $r = \log(s)$ gives that $T(i, s) \leq O(m \cdot (n+1)^{2\log(s)})$. □

Lemma 9.5 (Memory complexity of FIND)**.** FIND(S, s, d) *takes memory* $O(2^d(|S| + \log n))$.

Proof. Each call to FIND with depth d will only ever need simultaneously need to run up to 2 calls to FIND, each with depth $d-1$. Hence, there are at most 2^d copies of FIND that need to be stored in memory at any one time. At worst, each copy stores the sample as well as pointers to it and the n different variables. This means each copy uses $O(|S| + \log n)$ memory, for a total of $O(2^d(|S| + \log n))$ memory. □

The last step in this analysis is a standard generalization argument relying on Chernoff bounds.

Lemma 9.6 (Generalization)**.** *Choose any* $\delta, \varepsilon \geq 0$. *Suppose that* $\boldsymbol{S}$ *is a uniformly random sample from a function,* f*, that can be computed by a decision tree of size at most* s *and depth at most* d. *If the number of samples in* $\boldsymbol{S}$ *is at least*

$$m = O\left(\frac{2^d \log(n) + \log(\frac{1}{\delta})}{\varepsilon}\right)$$

and FIND$(\boldsymbol{S}, s, d)$ *returns a decision tree that fits* $\boldsymbol{S}$ *with error at most* $\frac{\varepsilon}{4}$. *Then, with probability at least* $1 - \delta$, *the decision tree returned by* FIND *has error at most* ε *on* f.

Proof. We will upper bound the number of different decision trees FIND could return when given depth limit d. There up to 2^d spots where a decision tree of depth $\leq d$ could have a node. In each of these spots, the decision tree could either have one of n variables, a leaf that is either $+1$ or -1, or nothing. Thus, the number of decision trees of depth at most d is at most $(n+3)^{2^d}$.

We call a decision tree, T, "bad", if T has error at least ε on f. For any particular bad tree T, the probability it will have error less than $\frac{\varepsilon}{4}$ on $\boldsymbol{S}$ can be upper bounded using a Chernoff Bound:

$$\Pr_{\boldsymbol{S}}\left[\text{error}_{\boldsymbol{S}}(T) \leq \tfrac{\varepsilon}{4}\right] \leq \exp(-\tfrac{9}{32}\varepsilon m).$$

Since there are most $(n+3)^{2^d}$ bad trees, the probability that any bad tree has error at most $\frac{\varepsilon}{4}$ is at most $(n+3)^{2^d} \cdot \exp(-\frac{9}{32}\varepsilon m)$. Setting this equal to δ and solving for m completes the proof of this lemma. □

Finally, we are able to put all these pieces together to prove our main theorem of this section, and show how FIND is used to learn decision trees:

Theorem 29 (Proper learning with polynomial sample and memory complexity)**.** *Let f be any function over n variables computable by a size s decision tree. Choose any $\varepsilon, \delta > 0$. There is an algorithm that*

- *runs in time* $\text{poly}(n^{\log s}, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$
- *requires memory* $\text{poly}(s, \log n, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$
- *uses* $\text{poly}(s, \log n, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$ *random samples from f*

and with probability at least $1-\delta$ returns a decision tree that is an ε-approximation of f.

Proof. Choose any constant $0 < c < 1$ and set $d = \log(\frac{s}{\varepsilon})/(-\log(\frac{3+c}{4}))$. Then, by taking a uniformly random sample, $\boldsymbol{S}$, of size

$$m = O\left(\frac{2^d}{\varepsilon c^2} \cdot (d\log(n) + \log(1/\delta))\right)$$

we have the following holds:

1. $\boldsymbol{S}$ is c-well-distributed with probability at least $1 - \frac{\delta}{2}$.
2. If $\boldsymbol{S}$ is c-well-distributed, then $\text{FIND}(\boldsymbol{S}, s, d)$ returns a tree, T, with error at most $\frac{1}{4}(\frac{3+c}{4})^d \cdot s = \frac{\varepsilon}{4}$ on $\boldsymbol{S}$.
3. If T has error less than $\frac{\varepsilon}{4}$, then with probability at least $1 - \frac{\delta}{2}$, T is a ε-approximation for f.

Furthermore, this procedure meets the time constraints by Lemma 9.4 and memory constraints by Lemma 9.5. □

Acknowledgments

We thank Clément Canonne, Adam Klivans, Charlotte Peale, Toniann Pitassi, Omer Reingold, and Rocco Servedio for helpful conversations and suggestions. We also thank the anonymous reviewers of ITCS 2020 for their valuable feedback.

The third author is supported by NSF grant CCF-1921795.

References

[AA14] Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *Theory of Computing*, 10(6):133–166, 2014.

[ABF+09] Misha Alekhnovich, Mark Braverman, Vitaly Feldman, Adam Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. *Journal of Computer & System Sciences*, 74(1):16–34, 2009.

[AFK13] Pranjal Awasthi, Vitaly Feldman, and Varun Kanade. Learning using local membership queries. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*, pages 398–431, 2013.

[BBL98] Avrim Blum, Carl Burch, and John Langford. On learning monotone boolean functions. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 408–415, 1998.

[BCO+15] Eric Blais, Clément Canonne, Igor Oliveira, Rocco Servedio, and Li-Yang Tan. Learning circuits with few negations. In *Proceedings of the 18th International Workshop on Randomization and Computation (RANDOM)*, pages 512–527, 2015.

[BDM19a] Alon Brutzkus, Amit Daniely, and Eran Malach. ID3 Learns Juntas for Smoothed Product Distributions. *ArXiv*, abs/1906.08654, 2019.

[BDM19b] Alon Brutzkus, Amit Daniely, and Eran Malach. On the Optimality of Trees Generated by ID3. *ArXiv*, abs/1907.05444, 2019.

[BFJ+94] Avirm Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 1994.

[BL97] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[Blu92] Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Inform. Process. Lett.*, 42(4):183–185, 1992.

[BMOS05] Nader H. Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning DNF from random walks. *J. Comput. System Sci.*, 71(3):250–265, 2005.

[BOGY18] Paul Beame, Shayan Oveis Gharan, and Xin Yang. Time-space tradeoffs for learning finite functions from random evaluations, with applications to polynomials. In *Proceedings of the 31st Conference On Learning Theory (COLT)*, volume 75, pages 843–856, 2018.

[Bre17] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[Bsh95] Nader Bshouty. Exact learning via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

[BT96] Nader Bshouty and Christino Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[BT15] Eric Blais and Li-Yang Tan. Approximating Boolean functions with depth-2 circuits. *SIAM J. Comput.*, 44(6):1583–1600, 2015.

[CM19] Sitan Chen and Ankur Moitra. Beyond the low-degree algorithm: mixtures of subcubes and their applications. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 869–880, 2019.

[DHK+10] Ilias Diakonikolas, Prahladh Harsha, Adam Klivans, Raghu Meka, Prasad Raghavendra, Rocco Servedio, and Li-Yang Tan. Bounding the average sensitivity and noise sensitivity of polynomial threshold functions. In *Proceedings of the 42nd Annual Symposium on Theory of Computing (STOC)*, pages 533–542, 2010.

[DKM96] Tom Dietterich, Michael Kearns, and Yishay Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proceedings of the 13th International Conference on Machine Learning (ICML)*, pages 96–104, 1996.

[DSLM+09] Dana Dachman-Soled, Homin K. Lee, Tal Malkin, Rocco A. Servedio, Andrew Wan, and Hoeteck Wee. Optimal cryptographic hardness of learning monotone functions. *Theory of Computing*, 5(13):257–282, 2009.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

[Fel16] Vitaly Feldman. Hardness of proper learning. In *Encyclopedia of Algorithms*, 2016.

[FK96] Ehud Friedgut and Gil Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124:2993–3002, 1996.

[FP04] Amos Fiat and Dmitry Pechyony. Decision trees: More theoretical justification for practical algorithms. In *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT)*, pages 156–170, 2004.

[Fri98] Ehud Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):474–483, 1998.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

[GKK08] Parikshit Gopalan, Adam Kalai, and Adam Klivans. Agnostically learning decision trees. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*, pages 527–536, 2008.

[GL89] Oded Goldreich and Leonid Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

[GRT18] Sumegha Garg, Ran Raz, and Avishay Tal. Extractor-based time-space lower bounds for learning. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 990–1002, 2018.

[GRT19] Sumegha Garg, Ran Raz, and Avishay Tal. Time-space lower bounds for two-pass learning. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, pages 22:1–22:39, 2019.

[GS10] Parikshit Gopalan and Rocco Servedio. Learning and lower bounds for AC^0 with threshold gates. In *Proceedings of the 14th International Workshop on Randomization and Computation (RANDOM)*, pages 588–601, 2010.

[HKY18] Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[HM91] Thomas Hancock and Yishay Mansour. Learning monotone k-μ DNF formulas on product distributions. In *Proceedings of the 4th Annual Conference on Computational Learning Theory (COLT)*, pages 179–193, 1991.

[JLSW11] Jeffrey Jackson, Homin Lee, Rocco Servedio, and Andrew Wan. Learning Random Monotone DNF. *Discrete Applied Mathematics*, 159(5):259–271, 2011.

[Kan14a] Daniel Kane. The average sensitivity of an intersection of halfspaces. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 437–440, 2014.

[Kan14b] Daniel Kane. The correct exponent for the Gotsman–Linial conjecture. *Computational Complexity*, 23(2):151–175, 2014.

[Kea96] Michael Kearns. Boosting theory towards practice: recent developments in decision tree induction and the weak learning framework (invited talk). In *Proceedings of the 13th National Conference on Artificial intelligence (AAAI)*, pages 1337–1339, 1996.

[KKL88] Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on boolean functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–80, 1988.

[KLV94] Michael Kearns, Ming Li, and Leslie Valiant. Learning Boolean formulas. *Journal of the ACM*, 41(6):1298–1328, 1994.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, December 1993.

[KM99] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 1999.

[KRT17] Gillat Kol, Ran Raz, and Avishay Tal. Time-space hardness of learning sparse parities. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1067–1080, 2017.

[KV94] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[Lee09] Homin Lee. *On the learnability of monotone functions.* PhD thesis, Columbia University, 2009.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[MM17] Dana Moshkovitz and Michal Moshkovitz. Mixing implies lower bounds for space bounded learning. In *Proceedings of the 30th Conference on Learning Theory (COLT)*, pages 1516–1566, 2017.

[MM18] Dana Moshkovitz and Michal Moshkovitz. Entropy samplers and strong generic lower bounds for space bounded learning. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 28:1–28:20, 2018.

[MOO10] Elchannan Mossel, Ryan O'Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: Invariance and optimality. *Annals of Mathematics*, 171:295–341, 2010.

[MOS04] Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *Journal of Computer and System Sciences*, 69(3):421–434, 2004.

[MR02] Dinesh Mehta and Vijay Raghavan. Decision tree approximations of boolean functions. *Theoretical Computer Science*, 270(1-2):609–623, 2002.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions.* Cambridge University Press, 2014. Available at http://analysisofbooleanfunctions.net/.

[OS07] Ryan O'Donnell and Rocco Servedio. Learning monotone decision trees in polynomial time. *SIAM Journal on Computing*, 37(3):827–844, 2007.

[OSSS05] Ryan O'Donnell, Michael Saks, Oded Schramm, and Rocco Servedio. Every decision tree has an influential variable. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 31–39, 2005.

[OW13] Ryan O'Donnell and Karl Wimmer. KKL, Kruskal–Katona, and Monotone Nets. *SIAM Journal on Computing*, 42(6):2375–2399, 2013.

[Qui86] Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui93] Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Raz17] Ran Raz. A time-space lower bound for a large class of learning problems. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 732–742, 2017.

[Raz18] Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *Journal of the ACM*, 66(1):3:1–3:18, December 2018.

[Riv87] Ronald Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

[Sel08] Linda Sellie. Learning random monotone DNF under the uniform distribution. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT)*, pages 181–192, 2008.

[Ser04] Rocco Servedio. On learning monotone DNF under product distributions. *Information and Computation*, 193(1):57–74, 2004.

[Sha14] Ohad Shamir. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *Proceedings of the 28th Conference on Neural Information Processing Systems*, pages 163–171, 2014.

[SM00] Yoshifumi Sakai and Akira Maruoka. Learning monotone log-term DNF formulas under the uniform distribution. *Theory of Computing Systems*, 33:17–33, 2000.

[ST13] Dominik Scheder and Li-Yang Tan. On the average sensitivity and density of k-CNF formulas. In *Proceedings of the 17th International Workshop on Randomization and Computation (RANDOM)*, pages 683–698, 2013.

[SVW16] Jacob Steinhardt, Gregory Valiant, and Stefan Wager. Memory, communication, and statistical queries. In *Proceedings of the 29th Annual Conference on Learning Theory (COLT)*, pages 1490–1516, 2016.

[Ver98] Karsten Verbeurgt. Learning sub-classes of monotone DNF on the uniform distribution. In *Proceedings of the 9th Conference on Algorithmic Learning Theory (ALT)*, pages 385–399, 1998.

[WFHP16] Ian Witten, Eibe Frank, Mark Hall, and Christopher Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il