Optimal Mistake Bounds for Transductive Online Learning

Zachary Chase
Kent State University
zchase2@kent.edu

Steve Hanneke
Purdue University
steve.hanneke@gmail.com

Shay Moran
Departments of Mathematics, Computer Science, and Data and Decision Sciences
Technion – Israel Institute of Technology;
Google Research
smoran@technion.ac.il

Jonathan Shafer
MIT
shaferjo@mit.edu

Abstract

We resolve a 30-year-old open problem concerning the power of unlabeled data in online learning by tightly quantifying the gap between transductive and standard online learning. In the standard setting, the optimal mistake bound is characterized by the Littlestone dimension d of the concept class $\mathcal{H}$ (Littlestone, 1987). We prove that in the transductive setting, the mistake bound is at least $\Omega\left(\sqrt{d}\right)$. This constitutes an exponential improvement over previous lower bounds of $\Omega(\log\log(d))$, $\Omega\left(\sqrt{\log(d)}\right)$, and $\Omega(\log(d))$, due respectively to Ben-David, Kushilevitz, and Mansour (1995, 1997), and Hanneke, Moran, and Shafer (2023). We also show that this lower bound is tight: for every d, there exists a class of Littlestone dimension d with transductive mistake bound $O\left(\sqrt{d}\right)$. Our upper bound also improves upon the best known upper bound of $(2/3)\cdot d$ from Ben-David et al. (1997). These results establish a quadratic gap between transductive and standard online learning, thereby highlighting the benefit of advance access to the unlabeled instance sequence. This contrasts with the PAC setting, where transductive and standard learning exhibit similar sample complexities.

1 Introduction

The transductive model is a basic and well-studied framework in learning theory, dating back to the early works of Vapnik. It has been investigated both in statistical and online settings, and is motivated by the principle that to make good predictions on a specific set of test instances, one need not construct a fully general classifier that performs well on the entire domain — including points that may never actually appear. Rather, it may be sufficient to tailor predictions for a fixed, known set of instances.

This perspective naturally connects to a broader question in learning theory: what is the value of unlabeled data? In the transductive setting, the learner is given the sequence of unlabeled test instances in advance and is then required to predict their labels one by one. Thus, the transductive model can be viewed as a natural formalization of learning with unlabeled data: the test instances are known in advance, but their labels are not. The central question is whether such prior access to the

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

unlabeled sequence can help reduce the number of prediction mistakes — compared to the standard online model, where the instances arrive and are labeled one at a time.

Recall for instance that in the standard PAC[1] model of supervised learning, there are cases where access to unlabeled data is not helpful. Indeed, the "hard population distributions" used to prove the standard VC[2] lower bound are constructed by taking a fixed and known marginal distribution over a VC-shattered set. Namely, the cases that are hardest to learn in the PAC setting include ones where the learner knows the marginal distribution over the domain, and can therefore generate as much unlabeled data as it wishes. And yet, in those cases, access to unlabeled data provides no acceleration compared to an algorithm (like ERM[3]) that does not use unlabeled data.

Seeing as unlabeled data is often a lot easier to obtain than labeled data, there have been considerable efforts to understand when and to what extent can access to unlabeled data accelerate learning.[4]

In particular, it is natural to ask, for which plausible models of learning is access to unlabeled data beneficial? Online learning (Littlestone, 1987) is perhaps the model of learning that is most-extensively studied in learning theory after the PAC model and its variants. Therefore, the general question considered in this paper is:

Question 1. *Quantitatively, how much (if at all) is access to unlabeled data beneficial for learning in the online learning setting?*

This question is naturally instantiated by comparing *transductive* online learning — where the learner has advance access to the full sequence $x_1, x_2, \ldots, x_n$ of unlabeled instances — with *standard* online learning, where no such access is given. This perspective has also been adopted in prior work: for example, Kakade and Kalai (2005), Cesa-Bianchi and Shamir (2013), and Hoi, Sahoo, Lu, and Zhao (2021) (Section 7.3) all describe transductive online learning as a setting in which the learner has access to "unlabeled data". We thus refine the question above as follows:

Question 2. *Quantitatively, how much (if at all) is learning in the transductive online learning setting easier than learning in the standard online learning setting? Specifically, how much is the optimal number of mistakes in the transductive setting smaller than in the standard setting?*

Addressing this question, our main result (Theorem 1.1) states that the optimal number of mistakes in the transductive setting (with access to unlabeled data) is at most quadratically smaller than in the standard setting (without unlabeled data). Furthermore, there are hypothesis classes for which a quadratic gap is achieved.

1.1 Setting: *Standard* vs. *Transductive* Online Learning

Standard online learning (Littlestone, 1987) is a zero-sum, perfect- and complete-information game played over n rounds between two players, a *learner* and an *adversary*. The game is played with respect to a *domain* set $\mathcal{X}$ and a *hypothesis class* $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ (consisting of functions $\mathcal{X} \to \{0,1\}$), where n, $\mathcal{X}$ and $\mathcal{H}$ are fixed and known to both players. The game proceeds as in Game 1. The *number of mistakes* for a learner L and an adversary A is $M_{\mathsf{std}}(\mathcal{H}, n, L, A) = |\{t \in [n] : \hat{y}_t \neq y_t\}|$. We are interested in understanding the *optimal number of mistakes*, which is

$$M_{\mathsf{std}}(\mathcal{H}) = \sup_{n \in \mathbb{N}} \inf_{L \in \mathcal{L}} \sup_{A \in \mathcal{A}} M_{\mathsf{std}}(\mathcal{H}, n, L, A),$$

where $\mathcal{A}$ and $\mathcal{L}$ are the set of all deterministic adversaries and learners, respectively.[5]

[1] Probably Approximately Correct. For an exposition of the standard terminology and results mentioned in this paragraph see, e.g., Shalev-Shwartz and Ben-David (2014).

[2] Vapnik–Chervonenkis.

[3] Empirical Risk Minimization.

[4] The literature on semi-supervised learning is surveyed in Joachims (1999); Zhu (2005); Zhu and Goldberg (2009); Zhu (2010); Chapelle, Schölkopf, and Zien (2006). Theoretical works on the topic include Benedek and Itai (1991); Blum and Mitchell (1998); Ben-David, Lu, Pál, and Sotáková (2008); Balcan and Blum (2010); Darnstädt, Simon, and Szörényi (2013); Göpfert, Ben-David, Bousquet, Gelly, Tolstikhin, and Urner (2019).

[5] Because the adversary selects y_t *after* seeing $\hat{y}_t$, randomness is not beneficial for either party, and we assume without loss of generality that both the learner and the adversary are deterministic. As is common in learning theory, we avoid questions of computability and allow the learner and adversary to be any function. See Section A for formal definitions of $\mathcal{A}$ and $\mathcal{L}$.

For each round $t = 1, 2, \ldots, n$:

a. The adversary selects an *instance* $x_t \in \mathcal{X}$ and sends it to the learner.

b. The learner selects a *prediction* $\hat{y}_t \in \{0, 1\}$ and sends it to the adversary.

c. The adversary selects a *label* $y_t \in \{0, 1\}$ and sends it to the learner. The selected label must be *realizable*, meaning that $\exists h \in \mathcal{H}\ \forall i \in [t]\colon\ h(x_i) = y_i$.

Game 1: The standard online learning setting.

The adversary selects a *sequence* $x_1, x_2, \ldots, x_n \in \mathcal{X}$ and sends it to the learner.

For each round $t = 1, 2, \ldots, n$:

a. The learner selects a *prediction* $\hat{y}_t \in \{0, 1\}$ and sends it to the adversary.

b. The adversary selects a *label* $y_t \in \{0, 1\}$ and sends it to the learner. The selected label must be *realizable*, meaning that $\exists h \in \mathcal{H}\ \forall i \in [t]\colon\ h(x_i) = y_i$.

Game 2: The transductive online learning setting.

It is well known that $M_{\mathsf{std}}(\mathcal{H})$ is characterized by the the Littlestone dimension, namely, $M_{\mathsf{std}}(\mathcal{H}) = \mathsf{LD}(\mathcal{H})$ (see Theorem A.7 and Definition A.6).

The *transductive* online learning setting (Ben-David et al., 1995, 1997) is similar, except that the learner has access to the full sequence of unlabeled instances in advance. Namely, as in Game 2. The optimal number of mistakes for the transductive setting is defined exactly as before,

$$M_{\mathsf{tr}}(\mathcal{H}, n, L, A) = |\{t \in [n] :\ \hat{y}_t \neq y_t\}|, \ \text{ and } \ M_{\mathsf{tr}}(\mathcal{H}) = \sup_{n \in \mathbb{N}} \inf_{L \in \mathcal{L}} \sup_{A \in \mathcal{A}} M_{\mathsf{tr}}(\mathcal{H}, n, L, A),$$

with the only difference between the standard quantity $M_{\mathsf{std}}(\mathcal{H})$ and the transductive quantity $M_{\mathsf{tr}}(\mathcal{H})$ being in how the game is defined.

1.2 Main Result

Notice that for every hypothesis class $\mathcal{H}$, $M_{\mathsf{tr}}(\mathcal{H}) \leq M_{\mathsf{std}}(\mathcal{H})$. Indeed, in the transductive setting the adversary declares the sequence x at the start of the game. This reduces the number of mistakes because the transductive adversary is less powerful (it cannot adaptively alter the sequence mid-game), and also because the transductive learner is more powerful (it has more information).[6]

While for some classes $M_{\mathsf{tr}}(\mathcal{H}) = M_{\mathsf{std}}(\mathcal{H})$, we study the largest possible separation. The best previous lower bound on M_{tr}, due to Hanneke, Moran, and Shafer (2023), states that for every class $\mathcal{H}$,

$$M_{\mathsf{tr}}(\mathcal{H}) \geq \Omega(\log(d)),$$

where $d = M_{\mathsf{std}}(\mathcal{H})$. In the other direction, Ben-David et al. (1997) constructed[7] a class $\mathcal{H}$ such that $M_{\mathsf{std}}(\mathcal{H}) = d$ and $M_{\mathsf{tr}}(\mathcal{H}) \leq \frac{2}{3}d$. This left an exponential gap between the best known lower and upper bounds on M_{tr}, namely $\Omega(\log d)$ versus $\frac{2}{3}d$. Our main result closes this gap:

Theorem 1.1 (Main result)**.**

- *For every hypothesis class* $\mathcal{H} \subseteq \{0, 1\}^{\mathcal{X}}$,

$$M_{\mathsf{tr}}(\mathcal{H}) = \Omega\left(\sqrt{d}\right),$$

where $d = M_{\mathsf{std}}(\mathcal{H})$.

[6] One could also define an intermediate setting, where the adversary is less powerful because it must select the sequence at the start of the game and cannot change it during the gameplay, but the learner does not have more information because the adversary only reveals the instances in the sequence one at a time as in the standard setting. However, this intermediate setting would not model the learner having *access* to unlabeled data.

[7] Their class consists of all disjoint unions of $\Theta(d)$ functions from a specific constant-sized class.

- *On the other hand, for every d there exists a hypothesis class $\mathcal{H}$ with $M_{\mathsf{std}}(\mathcal{H}) = d$ and*

$$M_{\mathsf{tr}}(\mathcal{H}) = O\left(\sqrt{d}\right).$$

This result is stated in considerably greater detail in Theorems B.1 and D.1.

1.3 Related Works

The notion of *transductive inference* as a more efficient alternative to *inductive inference* in statistical learning theory was introduced by Vapnik (1979, 2006); Gammerman, Vovk, and Vapnik (1998); Chapelle, Vapnik, and Weston (1999). The *online learning* setting is due to Littlestone (1987), who also proved that the optimal number of mistakes is characterized by the Littlestone dimension (see Theorem A.7).

The *transductive online learning* setting studied in the current paper, was first defined by Ben-David, Kushilevitz, and Mansour (1995), who used the name *worst sequence off-line model*. Among other results, they showed a lower bound of $\Omega(\log\log(d))$ on the number of mistakes required to learn a class with Littlestone dimension d. The authors subsequently presented an exponentially stronger lower bound of $\Omega\left(\sqrt{\log(d)}\right)$ in Ben-David, Kushilevitz, and Mansour (1997). However, understanding where the optimal number of mistakes is situated within the range $\left[\Omega\left(\sqrt{\log(d)}\right), 2d/3\right]$ remained an open question.

Kakade and Kalai (2005) presented an oracle-efficient algorithm for the transductive online learning setting, and may have been the first to use that name. Their result was subsequently improved upon by Cesa-Bianchi and Shamir (2013).

The present work is most similar to that of Hanneke, Moran, and Shafer (2023) which, among other results, gave a quadratically-stronger mistake lower bound of $\Omega(\log(d))$ for classes with Littlestone dimension d in the transductive online setting. The proof of our lower bound utilizes some of their ideas, but yields a quantitative improvement by combining it with some new ideas.

Hanneke, Raman, Shaeiri, and Subedi (2024) studied a setting of *multi-class* transductive online learning where the number of possible labels is unbounded.

2 Technical Overview

In this section we explain some of the main ideas in our proofs. Formal definition appear in Section A. Full formal statements of the results, as well as detailed rigorous proofs, appear in Sections B to D.

2.1 Paths in Trees

We make extensive use of the following notion. Given a perfect binary tree T_d of depth d, every function $f : T_d \to \{0,1\}$ defines a unique *path* in the tree. The path is a sequence of nodes $\mathrm{path}(f) = (x_{i_0}, x_{i_1}, \ldots, x_{i_d})$, as explained in Figure 1c. See Section A for formal definitions.

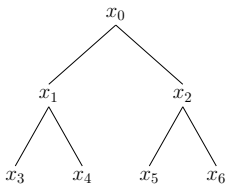
2.2 Proof Ideas for the Lower Bound

We start with an elementary observation about the adversary's dilemma in the transductive online learning setting. Before round t of the game, the adversary selected a full sequence of instances $x_1, x_2, \ldots, x_n \in \mathcal{X}$, and assigned some initial labels $y_1, y_2, \ldots, y_{t-1} \in \{0,1\}$. At the start of round t, the adversary must consider the *version space*,

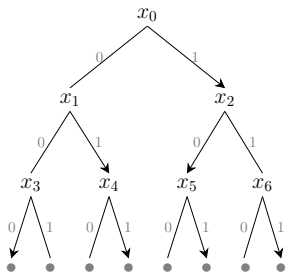
$$\mathcal{H}_t = \left\{h \in \mathcal{H} : \ (\forall i \in [t-1] : \ h(x_i) = y_i)\right\}.$$

If all $h \in \mathcal{H}_t$ assign $h(x_t) = b$ for some $b \in \{0,1\}$, then the adversary has no choice but to assign the label $y_t = b$. Otherwise, the adversary can *force a mistake* at time t. Namely, after seeing the learner's prediction $\hat{y}_t$, the adversary can assign $y_t = 1 - \hat{y}_t$, incrementing the number of learner mistakes by 1.

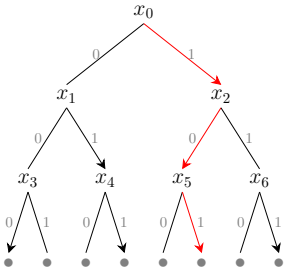
But "just because you can, doesn't mean you should". If the adversary is greedy and forces a mistake at time t, they may pay dearly for that later. As an extreme example, consider the case where there



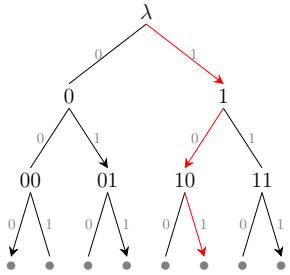
(a) A *perfect binary tree* of depth 2. Each *node* is labeled by an element of the domain $\mathcal{X}$. These labels need not be distinct (e.g., it is possible that $x_1 = x_6$). x_0 is the *root* of the tree, x_0, x_1 and x_2 are *internal nodes*, and $x_3, \ldots, x_6$ are *leaves*.



(b) A function $f : \mathcal{X} \to \{0, 1\}$ assigns a binary label to each node in the tree, represented here by edges with arrowhead tips. This figure depicts the function $f(x_i) = \mathbb{1}(i \notin \{2, 3\})$. (Note that the gray dots (•) in the figure are purely a pictorial detail. In this paper they are not considered nodes or leaves of the tree.)



(c) Every function $f : \mathcal{X} \to \{0, 1\}$ defines a *path* in the tree, which is a sequence $u_0, u_1, u_2, \ldots, u_{d-1}$, where u_0 is the root, d is the depth of the tree, and for each $i \in [d-1]$, u_i is the b-child of u_{i-1} with $b = f(u_{i-1}) \in \{0, 1\}$. This figure shows that the function f from Figure 1b has $\mathrm{path}(f) = (x_0, x_2, x_5)$, depicted in red. In particular, x_2 is 'on-path' for f, but x_6 is 'off-path' for f.



(d) In this paper we use a naming convention where, without loss of generality, we identify the domain elements x_i that are assigned to nodes with bit strings. The root is identified with the empty string λ, and for each pair of nodes u, v such that u is the b-child of v (for $b \in \{0, 1\}$), we have $u = v \circ b$, where '$\circ$' denotes string concatenation. (Because the x_i's may not be distinct, a domain element may be identified with more than one bit string.)

Figure 1: Paths in trees.

is a single $h_1 \in \mathcal{H}_t$ that assigns $h_1(x_t) = 1$, and all other functions $h \in \mathcal{H}_t$ assign $h(x_t) = 0$. If the learner selects $\hat{y}_t = 1$ and the adversary forces a mistake at time t, the version space at all subsequent times $s > t$ will be $\mathcal{H}_s = \{h_1\}$, and the adversary will be prevented from forcing any further mistakes.

A natural strategy for the adversary is therefore to be greedy up to a certain limit. Namely, at each time t the adversary computes the ratio[8]

$$r_t = \frac{|\{h \in \mathcal{H}_t : \ h(x_t) = 1\}|}{|\mathcal{H}_t|}.$$

If $r_t \in [\varepsilon, 1 - \varepsilon]$ for some parameter $\varepsilon > 0$ ("the version space is not too unbalanced"), then the adversary forces a mistake. Otherwise, the adversary assigns the majority label, i.e., $y_t = \mathbb{1}(r_t \geq 1/2)$. This ensures that the version space does not shrink too fast:

- If no mistake is forced, then $|\mathcal{H}_{t+1}| \geq (1 - \varepsilon) \cdot |\mathcal{H}_t|$, and
- If a mistake is forced, $|\mathcal{H}_{t+1}| \geq \varepsilon \cdot |\mathcal{H}_t|$.

[8]For a class $\mathcal{H}$ of Littlestone dimension d, the adversary will use only a subset of $\mathcal{H}$ of cardinality 2^d that shatters a Littlestone tree of depth $d - 1$. So without loss of generality, we assume that $\mathcal{H}$ has cardinality 2^d (in particular, $\mathcal{H}$ is finite), and the ratio is well-defined.

In particular, at the end of the game, the version space $\mathcal{H}_{n+1}$ is of size

$$|\mathcal{H}_{n+1}| \geq \varepsilon^M \cdot (1-\varepsilon)^{n-M} \cdot |\mathcal{H}| \geq \varepsilon^M \cdot (1-\varepsilon)^n \cdot 2^d, \tag{1}$$

where M is the number of mistakes that the adversary forces and n is the length of the sequence. The class has size $|\mathcal{H}| \geq 2^d$ because $\mathsf{LD}(\mathcal{H}) = d$, and by removing functions from the class if necessary (which can only make learning easier), we may assume without loss of generality that $|\mathcal{H}| = 2^d$. Namely, the class precisely shatters a Littlestone tree of depth $d-1$ such that for every assignment of labels to a root-to-leaf path in the tree, the class contains exactly one function that agrees with that assignment (see Definition A.6 for detail).

Notice that we have not yet specified how the adversary selects the sequence x. While the adversary's labeling strategy is extremely simple (determined by the ratio r_t and the prediction $\hat{y}_t$), constructing of the sequence x requires some care, to ensure that it has the following two properties:

- **Property I:** The length n of the sequence satisfies $n = 2^{\Theta\left(\sqrt{d}\right)}$, and
- **Property II:** For every sequence of predictions $\hat{y}_1, \ldots, \hat{y}_n$ selected by the learner, the resulting sequence of labels $y_1, \ldots, y_n$ selected by the adversary are consistent with some function $h \in \mathcal{H}$ such that x contains all the nodes in $\mathrm{path}(h)$.[9]

These properties can be achieved by carefully simulating all possible execution paths of the adversary.

Observe that if $\mathrm{path}(h) = (u_1, \ldots, u_d)$ then the sequence of labels $h(u_1), h(u_2), \ldots, h(u_d)$ uniquely identifies the function h within the class $\mathcal{H}$. Hence, Property II and the assumption $|\mathcal{H}| = 2^d$ imply that at the end of the game, the version space $\mathcal{H}_{n+1}$ has cardinality

$$|\mathcal{H}_{n+1}| = 1. \tag{2}$$

Combining Property I ($n = 2^{\Theta\left(\sqrt{d}\right)}$), Eqs. (1) and (2), and choosing $\varepsilon = 2^{-\Theta\left(\sqrt{d}\right)}$ gives

$$1 \geq \varepsilon^M \cdot (1-\varepsilon)^n \cdot 2^d \geq 2^{-\Theta\left(M \cdot \sqrt{d}\right)} \cdot 2^d,$$

which implies $M = \Omega\left(\sqrt{d}\right)$, as desired.

2.3 Proof Ideas for the Upper Bound

In this section we explain the main ideas in the proof of Theorem D.1, which states that for every $d \in \mathbb{N}$, there exists a class of Littlestone dimension d that is learnable in the transductive online setting with a mistake bound of $O\left(\sqrt{d}\right)$.

Of course, not every Littlestone class satisfies this property. For instance, the set of all functions $[d] \to \{0,1\}$ has Littlestone dimension d, but the adversary can force the learner to make d mistakes when learning this class in the transductive setting.[10] So our task in this proof is to construct a class that is especially easy to learn in the transductive setting (i.e., learnable with $O\left(\sqrt{d}\right)$ mistakes), while still being hard (requiring d mistakes) in the standard setting.

2.3.1 Sparse Encodings are Easy to Guess

We start with an elementary observation. Consider the following two bit strings:

Binary: 110101

One-hot: 00000000000000000000000000000000000000000000000000000100000000000

Both of these strings encode the number 53. However, one of the encodings is much easier to guess than the other: suppose we are tasked with guessing the bits in an encoding of an integer between 0 and $2^6 - 1$. We guess the bits one at a time, and after each guess, an adaptive adversary tells us whether our guess was correct.

[9] Recall that the *path* of a function h is depicted in Figure 1c, and defined in Definition A.5.

[10] The adversary simply selects the sequence $x = (1, 2, 3, \ldots, d)$, and for each x_i, the adversary forces a mistake by selecting $y_i = 1 - \hat{y}_i$. The adversary's choice of labels is realizable because we are working with the class of all function $[d] \to \{0,1\}$.

Now, if the bit string is a binary encoding, the task is hard. Each bit can either be 0 or 1, regardless of the values of the previous bits, and so the adversary can force a mistake on every bit. On the other hand, if we know that the string is a one-hot encoding, there exists an attractive strategy — always guess 0. This ensures that we will make at most 1 mistake.

Note that at the end of the guessing game we have learned the same amount of *information* (for a number between 0 and $2^n - 1$, we learned n bits of information), but the number of *mistakes* is very different (n mistakes vs. 1 mistake).

2.3.2 Construction of the Hypothesis Class

We now describe a construction of a hypothesis class that is easy to learn in the transductive setting, using the idea of a sparse encoding. Recall that a class $\mathcal{H}$ has Littlestone dimension at least d (Definition A.6 in Section A) if there exists a Littlestone tree of depth $d-1$ such that for every $b \in \{0,1\}^d$ there exists $h = h_b \in \mathcal{H}$ such that the values on the path of h agree with b. More formally, $\forall i \in [d] :\ h(b_{<i}) = b_i$, and in particular $\mathrm{path}(h) = (\lambda, b_{\leq 1}, b_{\leq 2}, b_{\leq 3}, \ldots, b_{\leq d-1})$. Thus, when constructing a class that shatters a specific Littlestone tree of depth $d-1$, we need to define 2^d functions $\left\{h_b :\ b \in \{0,1\}^d\right\}$. For each function h_b, the on-path values of the function are fixed (fully determined by b), while for the remaining values there is complete freedom (for the nodes u that are off-path we may assign any values $h_b(u) \in \{0,1\}$).

Perhaps the simplest way to construct a class of Littlestone dimension d is simply to assign all on-path values as required, and assign 0 to all other values. Namely, if u is a prefix of b then $h_b(u) = b_{|u|+1}$, and otherwise $h_b(u) = 0$. In a sense, this is the 'minimal' class of Littlestone dimension d for a specific Littlestone tree.[11]

Observe that the 'minimal' class does not have the desired property of being easy to learn in the transductive setting.[12] However, a certain variation of the 'minimal' class that embeds a sparse encoding does satisfy the requirement. In this variation, on-path value of the function h_b are assigned as they must (as determined by b), while the off-path values are sampled independently using a biased coin, such that each of them is 0 with high probability, but has a small probability of being 1. The probability is chosen carefully so that the class satisfies some simple combinatorial properties, as described further in Section 2.3.6 and Lemma D.2.

2.3.3 Naïve Learning Strategy

We now explain in broad strokes how the probabilistic construction of the hypothesis class in Section 2.3.2 is useful for learning with few mistakes in the transductive setting.

Notice that when predicting labels for the 'minimal' class with nodes in breadth-first order, the learner knows at each step whether they are labeling an on-path or off-path node, because the learner has already seen the correct labels for all ancestors of the current node. For off-path nodes, the learner knows that the true label is 0, so it never makes mistakes on off-path nodes, but it also gains no new information when the true labels for off-path nodes are revealed. No risk, but no reward either. Instead, all the information about the true labeling function is revealed only at on-path nodes, where the adversary has complete freedom to assign labels and force mistakes. That's why the adversary can force d mistakes.

For the randomly-chosen class, when predicting labels for off-path nodes, the learner may still safely predict a label of 0. But the reasoning for this is quite different. Conceptually, every off-path label is part of a sparse codeword that identifies the correct labeling function.[13] Because the coin is biased, each bit of the codeword is easy to guess (it is likely to be 0), but every time that the adversary reveals that the true label for an off-path node is indeed 0, the learner gains a small (nonzero) amount of

[11] More formally, this is a class with a minimal number of nodes labeled 1.

[12] The adversary can declare a sequence x consisting of all the nodes in the tree in breadth-first order, and then force d mistakes — one mistake in each layer (depth) of the tree. Specifically, regardless of how the adversary selects the labels, for each $i \in [d]$ there exists a node u_i at depth i that is on-path. When it is time for the learner to predict a label for this u_i, the learner knows that u_i is on-path because it has seen the correct labels for all the ancestors of u_i. However, the adversary has the freedom to extend the path arbitrarily to the left or to the right, and can therefore force a mistake on u_i.

[13] The coin-flips for off-path labels are all independent. For example, if X is a set of nodes all of which are off-path for a subset H of the hypothesis class, then the random variables $\{h(x) :\ h \in H,\ x \in X\}$ are i.i.d.

information about the true labeling function. Additionally, when the adversary selects an off-path label of 1, that reveals a lot of information about the true labeling function (such labels are rare in the hypothesis class), and therefore the adversary cannot force many off-path mistakes. Overall, the information about the true labeling function is 'smeared' throughout all labels of the tree (0s and 1s, on-path and off-path).[14]

Thus, the naïve general strategy for the learner when using the probabilistically-constructed class is to learn most of the information about the true labeling function by observing off-path labels. By the time the learner reaches an on-path node, it hopefully has already learned enough about the true labeling function in order to make a good prediction on that node.

However, making this general strategy work requires overcoming some very substantial obstacles:

1. Recall that in the transductive setting, the adversary can present the nodes of the tree in any order of its choosing — it does not have to present the tree in breadth-first order. The naïve strategy works only if the learner sees many off-path nodes before it sees most on-path nodes. But what happens if the adversary decides to present many on-path nodes near the beginning of the sequence? To handle this, the learner incorporates a strategy we call 'danger zone minimization', as described in Section 2.3.4.
2. Another, equally problematic, issue also arises from the fact that the sequence presented by the adversary might not be in breadth-first order. Recall that breadth-first order[15] has the property that for every node u in the sequence, all the ancestors of u appear *before* u in the sequence. This means that by the time the learner needs to predict a label for u, the learner knows whether u is on-path or off-path for the true labeling function. But what happens if the adversary presents u before some of u's ancestors? Or omits some of u's ancestors from the sequence altogether? In this case the learner doesn't know if u is on-path or off-path, and this presents a double hazard. One hazard is that the leaner doesn't know what label to predict for u — if u is off-path, the learner can simply predict 0, but if it is on-path it must do something more elaborate. The second hazard is that, after seeing the correct label for u, it is not clear what the learner can infer from it. If u is off-path, its label should be interpreted as part of a sparse encoding of the labeling function. But if u is on-path, the interpretation must be entirely different. To overcome this challenge, the learner incorporates a strategy we call 'splitting experts', described in Section 2.3.5.
3. Limiting off-path mistakes. Thanks to the coin's bias, most off-path nodes have a true label of 0. Nonetheless, each function in the hypothesis class still has an expected number of $2^{\Omega(d)}$ off-path nodes labeled 1, so the learner can afford to misclassify only a vanishing fraction of them! To limit the number of mistakes, the learner extracts information from the sparse encoding and executes a 'transition to Halving' strategy, as described in Section 2.3.6.

2.3.4 Danger Zone Minimization

Utilizing information from the 'sparse encoding' of the off-path nodes to make good predictions for on-path nodes requires that the learner first see the true labels for many off-path nodes. Until that happens, the learner expects to make many mistakes on on-path nodes. However, whether a node is on-path or off-path is not fixed in advanced — the adversary may decide this adaptively, in response to the learners predictions.

Danger zone minimization is a strategy used by the learner, to force the adversary to assign few nodes in the beginning of the sequence as on-path (otherwise, if initial nodes are assigned to be on-path by the adversary, then the learner will make few mistakes on those nodes). This is analogous to the standard Halving algorithm (Algorithm 7), but instead of minimizing the cardinality of the set of consistent hypotheses (the 'version space'), the learner minimizes a subset of the domain (the 'danger zone').

[14] Furthermore, the labels for most not-too-small subsets of the nodes reveal a lot of information about the correct labeling function — not just for a particular subset of nodes. These properties led us to code-name this construction while working on the paper as 'everything everywhere all at once' (in reference to a 2022 film of that name). This is in contrast to the 'minimal' function, where the information is concentrated entirely on the function path. The asymmetry between the 'minimal' class and the probabilistic class is similar to that between the binary and one-hot encodings in Section 2.3.1 above.

[15] As well as depth-first order.

Concretely, at the beginning of the game the learner initializes a set $S = \{x_1, x_2, \ldots, x_{t_{\mathsf{max}}}\}$ consisting of the first $t_{\mathsf{max}} = 2^{\Omega(\sqrt{d})}$ instances in the sequence x selected by the adversary. This set represents the 'danger zone' — nodes in the beginning of the sequence that have not been labeled yet, that *might* be on-path, and that are not ancestors of a previously-labeled on-path node.[16] To predict a label for an instance x_i, the learner selects a label $\hat{y}_i$ such that if $\hat{y}_i$ is wrong, the danger zone will shrink by at least $1/3$. That is, for $b \in \{0, 1\}$, if the set S_b of b-descendants of x_i has cardinality $|S_b| \geq |S|/3$, the learner predicts $\hat{y}_i = b$. Then, if the adversary selects $y_i = 1 - b$, that implies that all b-descendants of x_i are off-path for the true labeling functions. Therefore, the learner removes all b-descendants of x_i from the danger zone, and the new cardinality is $|S \setminus S_b| \leq (2/3) \cdot |S|$. This guarantees that the learner can make at most $O(\log(t_{\mathsf{max}})) = O\left(\sqrt{d}\right)$ such mistakes before the danger zone is empty.[17]

If neither S_0 nor S_1 have cardinality at least $|S|/3$, the learner predicts $\hat{y}_i = 0$. If $y_i = 1$ and x_i is on-path for the true labeling function, then the learner updates the danger zone to be $S_0 \cup S_1$,[18] again shrinking the danger zone by a factor of at most $2/3$. Otherwise, if $y_i = 1$ and x_i is off-path, then it was an off-path node labeled 1 (which is rare), and the learner can afford to misclassify it (see Section 2.3.6).

2.3.5 Splitting Experts

The danger zone minimization strategy requires that the learner know whether the node u being classified is on-path or off-path for the true labeling function. However, if u appears in the sequence before some of its ancestors, the learner does not know this. To overcome this difficulty, the learner implements a variant of the standard *multiplicative weights algorithm* using *splitting experts*. This means that initially there is a single expert executing danger zone minimization. When a node u is reached for which danger zone minimization requires knowing whether u is on-path or off-path and that information is not yet evident, each expert is split into two experts, one of which continues the execution of danger zone minimization under the assumption that u is on-path, and the other under the opposite assumption. Thus, at each point in time, there exists precisely one expert for which all path-related assumptions are correct, and therefore that expert will make at most $O\left(\sqrt{d}\right)$ mistakes. The multiplicative weights algorithm guarantees that the overall number of mistakes will be linear in the the number of mistakes of the best expert, i.e., $O\left(\sqrt{d}\right)$.

2.3.6 Transition to Halving

The hypothesis class is engineered such that it satisfies the following property: there are at most $2^{O(\sqrt{d})}$ functions in the hypothesis class that agree with any set of $t_{\mathsf{max}} = 2^{\Omega(\sqrt{d})}$ labels, or that agree that a set of $\Theta\left(\sqrt{d}\right)$ nodes are all off-path and labeled 1 (this follows from Lemma D.2).

Therefore, once the true labels for the first t_{max} instances $x_1, x_2, \ldots, x_{t_{\mathsf{max}}}$ have been revealed, or once $\Theta\left(\sqrt{d}\right)$ off-path labels of 1 have been revealed (whichever happens first), the learner can *transition to halving*: stop doing danger zone minimization, and instead predict the labels for the remaining nodes using the standard Halving algorithm (Algorithm 7) on the subset of the hypothesis class that survived. Halving on $2^{O(\sqrt{d})}$ functions is guaranteed to make at most $O\left(\sqrt{d}\right)$ mistakes (Fact E.1).

However, seeing as the learner lacks information on which nodes are off-path, it uses experts, and each expert maintains different path-related assumptions. Thus, each expert decides separately at which point to transition to Halving. The unique expert that makes only correct assumptions will

[16] If u is an ancestor of some on-path node v, and v is a b-descendant of u for $b \in \{0, 1\}$, then the true label for u must be b.

[17] Once the danger zone is empty, the learner cannot make any further on-path mistakes within the prefix $x_1, x_2, \ldots, x_{t_{\mathsf{max}}}$. And it will make at most $O\left(\sqrt{d}\right)$ mistakes on the remaining nodes $x_{t_{\mathsf{max}}+1}, x_{t_{\mathsf{max}}+2}, \ldots$, as explained in Section 2.3.6.

[18] Because on-path nodes must be either be descendants or ancestors of x_i, and the definition of the danger zone does not require that it contain ancestors of nodes that have been labeled.

transition 'at the right time'. That expert will make at most $O\left(\sqrt{d}\right)$ mistakes during danger zone minimization, and then at most $O\left(\sqrt{d}\right)$ additional mistakes during halving.

2.4 Some Intuition for the Quantity $\sqrt{d}$

We briefly sketch where the quantity $\sqrt{d}$ arises from. This is a back-of-the-envelope calculation without proof, intended purely as an aid for intuition. Suppose we assigned off-path labels of 1 with probability 2^{-k} instead of $2^{-\sqrt{d}}$. Consider a sequence $x_1, \ldots, x_n$ of $n = d/2k$ leaves. For any sequence of labels $y_1, \ldots, y_n \in \{0, 1\}$, taking $s = \sum_{i \in [n]} y_i$, there exist roughly

$$2^d \cdot \left(2^{-k}\right)^s \cdot \left(1 - 2^{-k}\right)^{n-s} \geq 2^d \cdot \left(2^{-k}\right)^n \gg 0$$

functions in the class for which these leaves are off-path and which agree with the labels $y_1, \ldots, y_n$. Therefore, the adversary can force at least $n = \Omega(d/k)$ mistakes.

Similarly, for the sequence $x_1, \ldots, x_n$ consisting of all the nodes in the tree of depth at most $k/2$ in breadth-first order, the adversary can force a mistake on every on-path node while assigning a label of 0 to all off-path nodes, for a total of $k/2$ mistakes. This is true because for any assignment of on-path labels, the fraction of functions which agree with the on-path labels that assign a label of 0 to all off-path nodes is roughly $\left(1 - 2^{-k}\right)^{2^{k/2}} \approx 1$, so in particular for any labeling of the on-path nodes there exists a function in the class that agrees with that labeling and assigns 0 to all off-path nodes.

Therefore, for any k, we obtain a *lower bound* of $\Omega\left(\frac{d}{k} + k\right)$ on the number of mistakes. For any k, $\frac{d}{k} + k \geq \sqrt{d}$, giving a lower bound of $\Omega\left(\sqrt{d}\right)$. Choosing $k = \sqrt{d}$ to minimize the lower bound will in fact yield a matching upper bound of $O\left(\sqrt{d}\right)$, as we show in this paper. This completes our overview of the upper bound.

3 Directions for Future Work

Following are some interesting open questions:

1. Does there exist an efficient learning algorithm that achieves the $O\left(\sqrt{d}\right)$ upper bound of Theorem D.1? One needs to be careful about the definition of efficiency here, but one possible formalization is as follows. Does there exist a learning algorithm A and a sequence of classes $\mathcal{H}_1, \mathcal{H}_2, \ldots$, such that for every $d \in \mathbb{N}$:
   - $\mathsf{LD}(\mathcal{H}_d) = d$, and
   - Given as input the index d and a sequence $x_1, \ldots, x_n$, the algorithm A runs in time $\mathsf{poly}(d, n)$ and makes at most $O\left(\sqrt{d}\right)$ mistakes assuming the labels are realizable by $\mathcal{H}_d$.
2. Is there a tradeoff between the cardinality of the domain $\mathcal{X}$ and the upper bound on the number of mistakes? We used a domain of size roughly 2^d in order to obtain our upper bound of $O\left(\sqrt{d}\right)$. Is it possible to get the same bound with a domain of size $\mathsf{poly}(d)$?
3. Obtaining more precise asymptotics; for example, is there (an explicit) constant $\alpha > 0$ such that the optimal transductive mistake bound is $\left(\alpha + o(1)\right)\sqrt{d}$?

4 Organization

Complete rigorous mathematical details are deferred to the appendices. Formal definitions appear in Section A. Formal statements and proofs for the lower bound and upper bound appear in Section B and Section D, respectively. Optimal sequence length is discussed in Section C.

Acknowledgments and Disclosure of Funding

ZC is supported in part by NSF EnCORE inst (award #2217058) and by Shachar Lovett's Simons Investigator Award (#929894). SM is a Robert J. Shillman Fellow; he acknowledges support by ISF grant 1225/20, by BSF grant 2018385, by Israel PBC-VATAT, by the Technion Center for Machine Learning and Intelligent Systems (MLIS), and by the the European Union (ERC, GENERALIZATION, 101039692). JS is supported in part by NSF CNS-2154149, an Amazon Research Award, and by Vinod Vaikuntanathan's Simons Investigator Award.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

Maria-Florina Balcan and Avrim Blum. A discriminative model for semi-supervised learning. *J. ACM*, 57(3):19:1–19:46, 2010. doi:10.1145/1706591.1706599. URL https://doi.org/10.1145/1706591.1706599.

Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. Online learning versus offline learning. In Paul M. B. Vitányi, editor, *Computational Learning Theory, Second European Conference, EuroCOLT '95, Barcelona, Spain, March 13-15, 1995, Proceedings*, volume 904 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 1995. doi:10.1007/3-540-59119-2_167. URL https://doi.org/10.1007/3-540-59119-2_167.

Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. Online learning versus offline learning. *Mach. Learn.*, 29(1):45–63, 1997. doi:10.1023/A:1007465907571. URL https://doi.org/10.1023/A:1007465907571.

Shai Ben-David, Tyler Lu, Dávid Pál, and Miroslava Sotáková. Learning low-density separators. *CoRR*, abs/0805.2891, 2008. URL http://arxiv.org/abs/0805.2891.

Gyora M. Benedek and Alon Itai. Learnability with respect to fixed distributions. *Theor. Comput. Sci.*, 86(2):377–390, 1991. doi:10.1016/0304-3975(91)90026-X. URL https://doi.org/10.1016/0304-3975(91)90026-X.

Avrim Blum and Tom M. Mitchell. Combining labeled and unlabeled data with co-training. In Peter L. Bartlett and Yishay Mansour, editors, *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT 1998, Madison, Wisconsin, USA, July 24-26, 1998*, pages 92–100. ACM, 1998. doi:10.1145/279943.279962. URL https://doi.org/10.1145/279943.279962.

Olivier Bousquet, Steve Hanneke, Shay Moran, Ramon van Handel, and Amir Yehudayoff. A theory of universal learning. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC 2021: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 532–541. ACM, 2021. doi:10.1145/3406325.3451087. URL https://doi.org/10.1145/3406325.3451087.

Nicolò Cesa-Bianchi and Ohad Shamir. Efficient transductive online learning via randomized rounding. In Bernhard Schölkopf, Zhiyuan Luo, and Vladimir Vovk, editors, *Empirical Inference - Festschrift in Honor of Vladimir N. Vapnik*, pages 177–194. Springer, 2013. doi:10.1007/978-3-642-41136-6_16. URL https://doi.org/10.1007/978-3-642-41136-6_16.

Olivier Chapelle, Vladimir N. Vapnik, and Jason Weston. Transductive inference for estimating values of functions. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 421–427. The MIT Press, 1999. URL http://papers.nips.cc/paper/1699-transductive-inference-for-estimating-values-of-functions.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, editors. *Semi-Supervised Learning*. The MIT Press, 2006. ISBN 9780262033589. doi:10.7551/MITPRESS/9780262033589.001.0001. URL https://doi.org/10.7551/mitpress/9780262033589.001.0001.

Malte Darnstädt, Hans Ulrich Simon, and Balázs Szörényi. Unlabeled data does provably help. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoreti-*

cal Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany, volume 20 of *LIPIcs*, pages 185–196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.STACS.2013.185. URL https://doi.org/10.4230/LIPIcs.STACS.2013.185.

Alexander Gammerman, Volodya Vovk, and Vladimir N. Vapnik. Learning by transduction. In Gregory F. Cooper and Serafín Moral, editors, *UAI 1998: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998*, pages 148–155. Morgan Kaufmann, 1998. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=243&proceeding_id=14.

Christina Göpfert, Shai Ben-David, Olivier Bousquet, Sylvain Gelly, Ilya O. Tolstikhin, and Ruth Urner. When can unlabeled data improve the learning rate? In Alina Beygelzimer and Daniel Hsu, editors, *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, pages 1500–1518. PMLR, 2019. URL http://proceedings.mlr.press/v99/gopfert19a.html.

Steve Hanneke, Shay Moran, and Jonathan Shafer. A trichotomy for transductive online learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/3e32af2df2cd13dfbcbe6e8d38111068-Abstract-Conference.html.

Steve Hanneke, Vinod Raman, Amirreza Shaeiri, and Unique Subedi. Multiclass transductive online learning. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/6f244818d72b2a4be9b1225d1344e950-Abstract-Conference.html.

Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021. doi:10.1016/J.NEUCOM.2021.04.112. URL https://doi.org/10.1016/j.neucom.2021.04.112.

Thorsten Joachims. Transductive inference for text classification using support vector machines. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 200–209. Morgan Kaufmann, 1999.

Sham M. Kakade and Adam Kalai. From batch to transductive online learning. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pages 611–618, 2005. URL https://proceedings.neurips.cc/paper/2005/hash/17693c91d9204b7a7646284bb3adb603-Abstract.html.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1987. doi:10.1007/BF00116827. URL https://doi.org/10.1007/BF00116827.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. ISBN 978-1-10-705713-5. URL http://www.cambridge.org/de/academic/subjects/computer-science/pattern-recognition-and-machine-learning/understanding-machine-learning-theory-algorithms.

Vladimir N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Nauka, Moscow, 1979. URL https://www.ipu.ru/node/63854/publications. In Russian.

Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, 2nd edition, 2006. ISBN 978-0-387-30865-4. doi:10.1007/0-387-34239-7. URL https://doi.org/10.1007/0-387-34239-7.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, Department of Computer Sciences, University of Wisconsin–Madison, 2005.

Xiaojin Zhu. Semi-supervised learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 892–897. Springer, 2010. doi:10.1007/978-0-387-30164-8_749. URL https://doi.org/10.1007/978-0-387-30164-8_749.

Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009. ISBN 978-3-031-00420-9. doi:10.2200/S00196ED1V01Y200906AIM006. URL https://doi.org/10.2200/S00196ED1V01Y200906AIM006.

Technical Appendices and Supplementary Material

A Preliminaries

A.1 Basic Notation

Notation A.1. $\mathbb{N} = \{1, 2, 3, \ldots\}$, *i.e.,* $0 \notin \mathbb{N}$. $\log(\cdot)$ *and* $\ln(\cdot)$ *denote logarithm to base* 2 *and* e*, respectively.*

Notation A.2 (Sequences)**.** *Let* $\mathcal{X}$ *be a set and* $n, k \in \mathbb{N}$*. For a sequence* $x = (x_1, \ldots, x_n) \in \mathcal{X}^n$*, we write* $x_{\leq k}$ *to denote the subsequence* $(x_1, \ldots, x_k)$*. If* $k \leq 0$ *then* $x_{\leq k}$ *denotes the empty sequence, which is also denoted by* $\lambda = \mathcal{X}^0$*. We use the notation* $\mathcal{X}^{\leq n} = \cup_{i=0}^{n} \mathcal{X}^i$*.*

A.2 Standard Online Learning

Let $\mathcal{X}$ be a set, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a collection of functions called a *hypothesis class*. A *learner strategy* or simply *learner* for the standard online learning game (Game 1) is a function

$$L : \bigcup_{i=0}^{n-1} (\mathcal{X} \times \{0,1\})^i \times \mathcal{X} \to \{0,1\},$$

where $n \in \mathbb{N}$ is the number of rounds in the game. The set of all such learner strategies is denoted $\mathcal{L}_n$. An *adversary strategy* or simply *adversary* for the standard online learning game is a pair of functions

$$A_{\mathsf{instance}} : \bigcup_{i=0}^{n-1} (\mathcal{X} \times \{0,1\} \times \{0,1\})^i \to \mathcal{X}, \text{ and}$$
$$A_{\mathsf{label}} : \bigcup_{i=1}^{n-1} (\mathcal{X} \times \{0,1\} \times \{0,1\})^i \times \{0,1\} \to \{0,1\}.$$

The set of all such adversary strategies is denoted $\mathcal{A}_n$.

Semantically, the interpretation of these strategies is that in each round $t \in [n]$ of Game 1, the adversary selects an instance

$$x_t = A_{\mathsf{instance}}(x_1, \hat{y}_1, y_1, \ldots, x_{t-1}, \hat{y}_{t-1}, y_{t-1}) \in \mathcal{X},$$

then the learner makes a prediction

$$\hat{y}_t = L(x_1, y_1, \ldots, x_{t-1}, y_{t-1}, x_t) \in \{0,1\},$$

and finally, the adversary assigns a label

$$y_t = A_{\mathsf{label}}(x_1, \hat{y}_1, y_1, \ldots, x_{t-1}, \hat{y}_{t-1}, y_{t-1}, \hat{y}_t) \in \{0,1\}.$$

The adversary's function A_{label} must satisfy *realizability*, meaning that there exists $h \in \mathcal{H}$ such that

$$\forall t \in [n] : \; y_t = h(x_t).$$

The number of mistakes in a game with n rounds and hypothesis class $\mathcal{H}$ between learner L and adversary A is

$$M_{\mathsf{std}}(\mathcal{H}, n, L, A) = |\{t \in [n] : \; \hat{y}_t \neq y_t\}|.$$

A.3 Transductive Online Learning

Given $\mathcal{X}$ and $\mathcal{H}$ as in Section A.2, a learner strategy for the *transductive online learning setting* (Game 2) is a function

$$L : \; \mathcal{X}^n \times \bigcup_{i=0}^{n-1} \{0,1\}^i \to \{0,1\},$$

where $n \in \mathbb{N}$ is the number of rounds in the game. An adversary strategy consists of a sequence $x \in \mathcal{X}^n$ and an *adversary labeling strategy*, which is a function

$$A : \left(\bigcup_{i=0}^{n-1} \{0,1\}^{2i} \right) \times \{0,1\} \to \{0,1\}.$$

The sets of all such learner and adversary strategies are denoted $\mathcal{L}_n$ and $\mathcal{A}_n$ respectively.

Semantically, the interpretation of these strategies is that at the start of Game 2, the adversary selects the sequence x. Then, in each round $t \in [n]$, the learner makes a prediction

$$\hat{y}_t = L(x, y_1, \ldots, y_{t-1}) \in \{0,1\},$$

and then the adversary assigns a label

$$y_t = A(\hat{y}_1, y_1, \ldots, \hat{y}_{t-1}, y_{t-1}, \hat{y}_t) \in \{0,1\}.$$

Exactly as in Section A.2, the adversary's function A must satisfy realizability, namely,

$$\exists h \in \mathcal{H}\ \forall t \in [n] :\ y_t = h(x_t),$$

and the number of mistakes in a game with sequence length n and hypothesis class $\mathcal{H}$ between learner L and adversary A is

$$M_{\mathsf{tr}}(\mathcal{H}, n, L, A) = |\{t \in [n] :\ \hat{y}_t \neq y_t\}|.$$

A.4 Mistake Bounds

In this paper, we study *optimal mistake bounds*, or the *optimal number of mistakes*, which is the value of Games 1 and 2. For $M \in \{M_{\mathsf{std}}, M_{\mathsf{tr}}\}$, the optimal number of mistakes in a game with hypothesis class $\mathcal{H}$ and sequence length n is,

$$M(\mathcal{H}, n) = \inf_{L \in \mathcal{L}_n} \sup_{A \in \mathcal{A}_n} M(\mathcal{H}, n, L, A).$$

The optimal number of mistakes for hypothesis class $\mathcal{H}$ is

$$M(\mathcal{H}) = \sup_{n \in \mathbb{N}} M(\mathcal{H}, n).$$

Remark A.3. As is common in learning theory literature, in both Game 1 and Game 2, we take the sets $\mathcal{L}_n$ and $\mathcal{A}_n$ to be the sets of all (deterministic) functions. In this paper, we do not consider randomized strategies. By allowing arbitrary functions, we ignore issues relating to computability.

A.5 Trees

Definition A.4 (Notation for binary trees)**.** *Let $d \in \mathbb{N} \cup \{0\}$. A perfect binary tree of depth d is a collection of $2^{d+1} - 1$ nodes, which we identify with the collection of binary strings*

$$T_d = \left\{\{0,1\}^k :\ k \in \{0,1,2,\ldots,d\}\right\}.$$

The empty string, denoted $\lambda = \{0,1\}^0$, is a member of T_d and is called the root of the tree. Every string $u \in \{0,1\}^d$ is called a leaf. The depth of a node $u \in T_d$, denoted $|u|$, is the length of u as a string, namely, the integer k such that $u \in \{0,1\}^k$.

For two nodes $u, v \in T_d$, we say that u is a parent of v, and that v is a child of u, if $v = u \circ 0$ or $v = u \circ 1$, where $\circ$ denotes string concatenation. More fully, for $b \in \{0,1\}$, we say that v is a b-child of u if $v = u \circ b$.

Recursively, we define that u is an ancestor of v and that v is a descendant of u, and write $u \preccurlyeq v$, if one of the following holds:

- *$u = v$, or*
- *$\exists w \in T_d\ \exists b \in \{0,1\} :\ (u \preccurlyeq w)\ \wedge\ (w \circ b = v)$.*

For $b \in \{0,1\}$, we say that v is a b-descendant of u, denoted $u \preccurlyeq_b v$, if v is a descendant of the b-child of u.

A function $f :\ T_d \to \{0,1\}$ specifies a particular root-to-leaf path in the tree T_d (see Figure 1). The *on-path* nodes for f are the set of $d+1$ nodes on that root-to-leaf path, as in the following definition.

Definition A.5 (Paths in a binary tree)**.** *Let $d, k \in \mathbb{N}$, $k \leq d$. Let $u \in \{0,1\}^k$ be a node in T_d. The path to u is the unique sequence $\mathrm{path}(u) = (u_0, u_1, u_2, \ldots, u_k)$ such that $u_0 = \lambda$ is the root, $u_k = u$, and u_i is a child of u_{i-1} for all $i \in [k]$.*

Let $f : T_d \to \{0,1\}$ be a function. The path of f is the unique sequence $\mathrm{path}(f) = (u_0, u_1, u_2, \ldots, u_d)$ such that $u_0 = \lambda$ is the root, and for each $i \in [d]$, $u_i = u_{i-1} \circ f(u_{i-1})$. Namely, u_i is the $f(u_{i-1})$-child of u_{i-1}.

For a node $v \in T_d$ and a function $f : T_d \to \{0,1\}$, we write $v \in \mathrm{path}(f)$ if $\mathrm{path}(f) = (u_0, \ldots, u_d)$ and there exists $i \in \{0, \ldots, d\}$ such that $u_i = v$. Otherwise, we write $v \notin \mathrm{path}(f)$.

For a node $v \in T_d$ and a set of functions $\mathcal{F} \subseteq \{0,1\}^{T_d}$, we write $v \in \mathrm{path}(\mathcal{F})$ if

$$\forall f \in \mathcal{F} : \ v \in \mathrm{path}(f).$$

Otherwise, we write $u \notin \mathrm{path}(\mathcal{F})$.

A.6 Littlestone Dimension

Definition A.6 (Littlestone, 1987)**.** *Let $\mathcal{X}$ be a set, let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$, and let $d \in \mathbb{N} \cup \{0\}$. We say that $\mathcal{H}$ shatters the binary tree T_d if there exists a mapping $T_d \to \mathcal{X}$ given by $u \mapsto x_u$ such that for every $u \in \{0,1\}^{d+1}$ there exists $h_u \in \mathcal{H}$ such that*

$$\forall i \in [d+1] : \ h(x_{u_{\leq i-1}}) = u_i.$$

The Littlestone dimension of $\mathcal{H}$, denoted $\mathsf{LD}(\mathcal{H})$, is the supremum over all $d \in \mathbb{N}$ such that there exists a Littlestone tree of depth $d-1$ that is shattered by $\mathcal{H}$.

Note that by defining the Littlestone dimension this way, every class with Littlestone dimension $d \in \mathbb{N}$ contains at least 2^d functions.

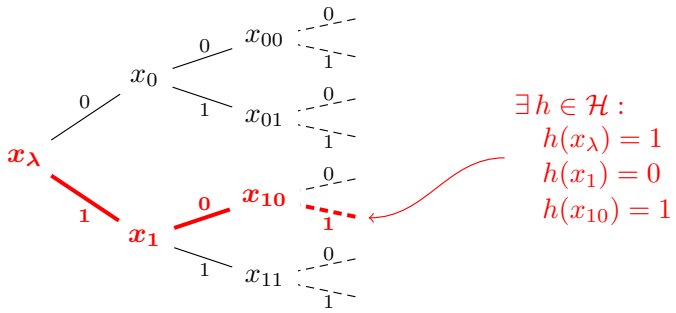


Figure 2: A shattered Littlestone tree of depth 2. The empty sequence is denoted by λ.
(Source: Bousquet et al., 2021)

Theorem A.7 (Littlestone, 1987)**.** *Let $\mathcal{X}$ be a set and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ such that $d = \mathsf{LD}(\mathcal{H}) < \infty$. Then there exists a strategy for the learner that guarantees that the learner will make at most d mistakes in the standard (non-transductive) online learning setting, regardless of the adversary's strategy and of the number n of instances to be labeled. Furthermore, there exists an adversary that forces every learner to make at least $\min\{n, d\}$ mistakes.*

B Lower Bound

B.1 Statement

Our $\Omega\left(\sqrt{d}\right)$ lower bound states the following.

Theorem B.1 (Lower bound)**.** *There exists a constant $d_0 \geq 0$ as follows. Let $d \in \mathbb{N}$, $d \geq d_0$, let $\mathcal{X}$ be a set, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a hypothesis class with $\mathsf{LD}(\mathcal{H}) = d$. Then there exist a sequence*

$x \in \mathcal{X}^n$ *of length* $n = O\left(d \cdot 2^{\sqrt{d}}\right)$ *and an adversary* A *that always selects the sequence* x *and uses a simple adaptive labeling strategy (as in Algorithm 1), such that for every learning rule* L*,*

$$M_{\mathsf{tr}}(\mathcal{H}, n, L, A) \geq \sqrt{d}/10. \tag{3}$$

Furthermore, for every integer $n \in \mathbb{N}$*,*

$$M_{\mathsf{tr}}(\mathcal{H}, n) \geq \min\left\{\sqrt{d}/10, \lfloor \log(n+1) \rfloor\right\}. \tag{4}$$

Remark B.2. The assumption $\mathsf{LD}(\mathcal{H}) = d$ implies that for all $k \in [d]$, $\mathcal{H}$ shatters a Littlestone tree of depth k. Thus, the lower bound of Eq. (3) in Theorem B.1 immediately implies that for every $k \in [d]$ there exists a sequence $x^{(k)} \in \mathcal{X}^{n_k}$ of length $n_k = O\left(k \cdot 2^{\sqrt{k}}\right)$ such that the adversary A_k that presents the sequence $x^{(k)}$ and assigns labels using the simple labeling strategy of Algorithm 1 ensures that for every learner L,

$$M_{\mathsf{tr}}(\mathcal{H}, n_k, L, A_k) \geq \sqrt{k}/10.$$

See Section 2.2 for a general overview of Theorem B.1 and the main proof ideas. In the following subsections we prove Theorem B.1. Algorithm 1 gives an explicit construction of the adversary that witnesses the lower bound, using Algorithm 2 as a subroutine. We start with presenting some initial observations about the behavior of these algorithms in Section B.2.

Assumptions:

- $d \in \mathbb{N}$, $\varepsilon = 2^{-\sqrt{d}/2}$.
- $T = T_d$ is a perfect binary tree of depth d.
- $\mathcal{H} \subseteq \{0,1\}^T$ is a class that shatters T.

TRANSDUCTIVEADVERSARY $(\mathcal{H})$:

$(x_1, x_2, \ldots, x_n) \leftarrow$ CONSTRUCTSEQUENCE $(\mathcal{H})$ ▷ See Algorithm 2.

send $(x_1, x_2, \ldots, x_n)$ to learner

$\mathcal{H}_0 \leftarrow \mathcal{H}$

for $t \in [n]$:

receive $\hat{y}_t$ from learner

$$r_t \leftarrow \frac{|\{h \in \mathcal{H}_{t-1} : \ h(x_t) = 1\}|}{|\mathcal{H}_{t-1}|}$$

$y_{\mathsf{maj}} \leftarrow \mathbb{1}(r_t \geq 1/2)$

$$y_t \leftarrow \begin{cases} y_{\mathsf{maj}} & r_t \notin [\varepsilon, 1-\varepsilon] \\ 1 - \hat{y}_t & \text{otherwise} \end{cases}$$

send y_t to learner

$\mathcal{H}_t \leftarrow \{h \in \mathcal{H}_{t-1} : \ h(x_t) = y_t\}$

Algorithm 1: The strategy for the adversary that achieves the lower bound in Theorem B.1. Note that while the construction of the sequence x is not entirely trivial, the adversary's strategy for labeling this sequence is very simple.

B.2 Analysis of the Adversary

Claim B.3. *Let* $d \in \mathbb{N}$*, let* $M = \sqrt{d}/10$*, and let* $\mathcal{H} \subseteq \{0,1\}^{T_d}$ *be a hypothesis class. Consider an execution of* CONSTRUCTSEQUENCE $(\mathcal{H})$ *as in Algorithm 2 that produces a sequence* $x_1, x_2, \ldots, x_n$*. Then:*

Assumptions:

- $d \in \mathbb{N}$, $M = \sqrt{d}/10$, $\varepsilon = 2^{-\sqrt{d}/2}$.
- $T = T_d$ is a perfect binary tree of depth d.
- λ, the empty string, is the root of T.
- $\mathcal{H} \subseteq \{0,1\}^T$ is a class that shatters T.

CONSTRUCTSEQUENCE$(\mathcal{H})$:

$\mathcal{H}_\lambda \leftarrow \mathcal{H}$

$\mathbb{H}_0 \leftarrow \{\mathcal{H}_\lambda\}$ ▷ A set of classes indexed by bit strings.

$\mathcal{Q} \leftarrow \{\lambda\}$ ▷ A set of nodes to be processed.

$t \leftarrow 0$

while $|\mathcal{Q}| > 0$:

$t \leftarrow t + 1$

$x_t \leftarrow$ arbitrary element from $\mathcal{Q}$ ▷ Pop an arbitrary element from $\mathcal{Q}$ and add it to the output sequence.

$\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{x_t\}$

$\mathbb{H}_t \leftarrow \varnothing$

for $\mathcal{H}_b \in \mathbb{H}_{t-1}$:

$$r \leftarrow \frac{|\{h \in \mathcal{H}_b : \ h(x_t) = 1\}|}{|\mathcal{H}_b|}$$

$$\mathcal{Y} \leftarrow \begin{cases} \{0,1\} & \big(r \in [\varepsilon, 1-\varepsilon]\big) \wedge \big(|b| < M\big) \\ \{\mathbb{1}(r \geq 1/2)\} & \text{otherwise} \end{cases}$$

▷ Adversary will force mistakes on the first M balanced nodes.

for $y \in \mathcal{Y}$:

$$b' \leftarrow \begin{cases} b & |\mathcal{Y}| = 1 \\ b \circ y & |\mathcal{Y}| = 2 \end{cases}$$

▷ Restrict class to agree with y. If splitting the class in two to force a mistake then create new indices.

$\mathcal{H}_{b'} \leftarrow \{h \in \mathcal{H}_b : h(x_t) = y\}$

$\mathbb{H}_t \leftarrow \mathbb{H}_t \cup \{\mathcal{H}_{b'}\}$

if $x_t \in \mathrm{path}(\mathcal{H}_{b'}) \ \wedge \ |x_t| < d$: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$ ▷ If x_t is on-path for $\mathcal{H}_{b'}$ and it has a y-child, add that child to $\mathcal{Q}$.

return $(x_1, x_2, \ldots, x_t)$

Algorithm 2: A subroutine of Algorithm 1 for selecting the sequence x.

(a) For all $i \in [n]$, $\mathrm{path}(x_i)$ is a subsequence of $x_0, x_1, \ldots, x_i$.

(b) The length n of the sequence satisfies $n < n_d$, where $n_d = (d+1) \cdot 2^{M+1}$.

Proof.

(a) Fix $i \in [n]$. It suffices to show that for all $u \in T_d$, if $u \preccurlyeq x_i$ then $u \in (x_1, x_2, \ldots, x_i)$. Proceed by induction on i. For the base case $i = 1$, the claim holds because $x_1 = \lambda$.

For the induction step, assume the claim holds for $i \in [n-1]$. Let $u \preccurlyeq x_{i+1}$, we prove that $u \in (x_1, x_2, \ldots, x_{i+1})$. Assume $x_{i+1} \neq \lambda$ (otherwise, there is nothing to prove).

Because x_{i+1} appears in the sequence x, it must have been added to $\mathcal{Q}$ before it was added to x. The only place where items that are not λ are added to $\mathcal{Q}$ is in the line $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$. Namely, there exist an index $j \in [i]$ and a bit $y \in \{0,1\}$ such that $x_{i+1} = x_j \circ y$ (note that $j < i+1$ because x_j was added to the sequence before x_{i+1}). If $x_j = u$ we are done. Otherwise, note that x_j is the parent of x_{i+1}, and therefore $u \preccurlyeq x_j$. By the induction hypothesis, $u \in (x_1, x_2, \ldots, x_j)$. This concludes the proof.

(b) Items are added to the sequence x only if they were previously added to $\mathcal{Q}$. By induction on $i \in [n]$, for each x_i in the sequence, there is at most one iteration of the "while $|Q| > 0$" loop in which x_i is added to Q. The base case $i = 1$ holds because $x_1 = \lambda$ is the root, which is added to Q before the while loop, and λ is never added to Q within that loop because the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" can only add non-empty bit strings. For the induction step, if the claim holds for all natural numbers j such that $1 \le j < i \le n$ then it holds for i. Indeed, for $i \ge 2$, x_i can be added to Q only via the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$", and only in the iteration of the while loop where x_t is the parent of x_i in the tree T_d. In that iteration, the parent x_t of x_i is popped from Q, which implies that x_t was added to Q in some previous iteration of the while loop ($t < i$), and is no longer in Q after being popped. By the induction hypothesis, x_t will never be added to Q again, and therefore in all subsequent iterations of the while loop x_t will not be the parent of x_i, so x_i cannot be added to Q in subsequent iterations via the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$".

Furthermore, if a node x_i is added to Q in some iteration of the while loop, then it remains in Q for the duration of that iteration. So for all $i \in \{2, 3, \ldots, n\}$, there is precisely one execution of the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" that adds x_i to Q. Namely, there is precisely one point in time during the execution of Algorithm 2 in which $x_i = x_t \circ y$, $x_i \notin Q$, and the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" is executed resulting in $x_i \in Q$.

Consider a function f that maps $i \in \{2, 3, \ldots, n\}$ to the value of the index b' during the unique execution of the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" that adds x_i to Q. Namely, if b' had some value β when x_i was added to Q, then $f(i) = \beta$.

Notice that "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" is executed only if the condition $x_t \in \mathrm{path}(\mathcal{H}_{b'})$ is satisfied in the previous line. Furthermore, the line "$\mathcal{H}_{b'} \leftarrow \{h \in \mathcal{H}_b : h(x_t) = y\}$" ensures that the node $x_i = x_t \circ y$ being added to Q satisfies $x_t \circ y \in \mathrm{path}(\mathcal{H}_{b'})$, namely

$$\forall h \in \mathcal{H}_{b'} : \; x_i \in \mathrm{path}(h).$$

Consequently, $x_i \in \mathrm{path}(\mathcal{G})$ for any class $\mathcal{G}$ that is a subset of $\mathcal{H}_{b'}$; in particular, because the only way that $\mathcal{H}_{b'}$ might be modified later during the execution of Algorithm 2 is by removing elements, it follows that $x_i \in \mathrm{path}(\mathcal{H}_{b'})$ when the line "$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x_t \circ y\}$" is executed and in all subsequent times.

However, $|\mathrm{path}(\mathcal{G})| = d + 1$ for any class $\mathcal{G} \subseteq \{0,1\}^{T_d}$. This implies that f maps at most $(d+1)$ nodes to each bit string. In other words, for any bit string b, the size of the preimage satisfies $|f^{-1}(b)| \le d + 1$.

The condition "$|b| < M$" in Algorithm 2 ensures that $|b'| \le M$, namely, $b' \in \{0,1\}^k$ for $k \in \{0, 1, 2, \ldots, M\}$. Thus,

$$\begin{aligned}
n &= 1 + |\{2, 3, \ldots, n\}| \\
&= 1 + \sum_{\substack{b \in \{0,1\}^k \\ k \in \{0,\ldots,M\}}} |\{i \in \{2, 3, \ldots, n\} : \; f(i) = b\}| \\
&= 1 + \sum_{\substack{b \in \{0,1\}^k \\ k \in \{0,\ldots,M\}}} |f^{-1}(b)| \\
&\le 1 + \sum_{\substack{b \in \{0,1\}^k \\ k \in \{0,\ldots,M\}}} (d+1) \\
&\le 1 + (d+1) \cdot (2^{M+1} - 1). \\
&< (d+1) \cdot 2^{M+1},
\end{aligned}$$

as desired. □

Claim B.4. *Let $d \in \mathbb{N}$, let $M = \sqrt{d}/10$, and let $\mathcal{H} \subseteq \{0,1\}^{T_d}$ be a hypothesis class. Consider an execution of* TRANSDUCTIVEADVERSARY $(\mathcal{H})$ *as in Algorithm 1. Let*

$$\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n$$

be the sequence of hypothesis classes created by TRANSDUCTIVEADVERSARY*, let*

$$S = \left\{t \in [n] : \ r_t \in [\varepsilon, 1-\varepsilon]\right\}$$

be the set of indices where TRANSDUCTIVEADVERSARY *forces a mistake, and let*

$$\mathbb{H}_0, \mathbb{H}_1, \ldots, \mathbb{H}_n$$

be the sequence of collections created by the subroutine CONSTRUCTSEQUENCE *(Algorithm 2). If $|S| \leq M$ then*

$$\forall t \in \{0, 1, \ldots, n\} : \ \mathcal{H}_t \in \mathbb{H}_t.$$

Proof. Proceed by induction on $t \in \{0, 1, \ldots, n\}$. The base case $t = 0$ is satisfied, because $\mathcal{H}_0 = \mathcal{H} \in \{\mathcal{H}\} = \mathbb{H}_0$. For the induction step, assume that $\mathcal{H}_{i-1} \in \mathbb{H}_{i-1}$ for some $i \in [n]$. We prove that $\mathcal{H}_i \in \mathbb{H}_i$.

Let y_i be the label assigned to x_i by TRANSDUCTIVEADVERSARY. Then

$$\mathcal{H}_i = \{h \in \mathcal{H}_{i-1} : \ h(x_i) = y_i\}.$$

Consider the iteration of the while loop in CONSTRUCTSEQUENCE that starts with $t \leftarrow i$. By the induction hypothesis, $\mathcal{H}_{i-1} \in \mathbb{H}_{i-1}$. Therefore, in this iteration of the while loop, there will be an iteration of the "for $\mathcal{H}_b \in \mathbb{H}_{t-1}$" loop where $\mathcal{H}_b = \mathcal{H}_{i-1}$. In that iteration, $y_i \in \mathcal{Y}$ by construction of y_i and $\mathcal{Y}$. Therefore, in the iteration of the "for $y \in \mathcal{Y}$" loop in which $y = y_i$,

$$\mathcal{H}_{b'} = \{h \in \mathcal{H}_b : h(x_t) = y\} = \{h \in \mathcal{H}_{i-1} : h(x_i) = y_i\} = \mathcal{H}_i.$$

The class $\mathcal{H}_{b'}$ is then added to $\mathbb{H}_i = \mathbb{H}_t$ in the line "$\mathbb{H}_t \leftarrow \mathbb{H}_t \cup \{\mathcal{H}_{b'}\}$". Furthermore, no class is ever removed from $\mathbb{H}_t$. So $\mathcal{H}_i \in \mathbb{H}_i$, as desired. □

Claim B.5. *Let $d \in \mathbb{N}$, let $M = \sqrt{d}/10$, and let $\mathcal{H} \subseteq \{0,1\}^{T_d}$ be a hypothesis class. Consider an execution of* TRANSDUCTIVEADVERSARY $(\mathcal{H})$ *as in Algorithm 1 where the adversary constructs a sequence of nodes $x_1, x_2, \ldots, x_n \in T_d$ and a sequence of classes $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n \subseteq \{0,1\}^{T_d}$. Let*

$$S = \left\{t \in [n] : \ r_t \in [\varepsilon, 1-\varepsilon]\right\}$$

be the set of indices where TRANSDUCTIVEADVERSARY *forces a mistake, and assume that $|S| \leq M$. Then for all $k \in \{0, 1, \ldots, d\}$ there exists $i \in [n]$ such that*

1. *$|x_i| = k$, and*
2. *$x_i \in \mathrm{path}(\mathcal{H}_{i-1})$,*

Proof. Proceed by induction on k. For the base case $k = 0$, notice that $x_1 = \lambda$, $|\lambda| = 0$, and $\lambda \in \mathrm{path}(\mathcal{H}_{-1})$.

For the induction step, assume the claim holds for some $k \in \{0, 1, \ldots, d-1\}$, and take $i_k \in [n]$ such that $|x_{i_k}| = k$ and $x_{i_k} \in \mathrm{path}(\mathcal{H}_{i_k-1})$; we prove that the claim holds for $k+1$ as well.

Consider the iteration of the while loop in CONSTRUCTSEQUENCE in which x_{i_k} is added to the sequence (i.e., the iteration starting with $t \leftarrow i_k$). By Claim B.4 and the assumption $|S| \leq M$, $\mathcal{H}_{i_k-1} \in \mathbb{H}_{i_k-1}$. Hence, within this iteration of the while loop, there is an iteration of the "for $\mathcal{H}_b \in \mathbb{H}_{t-1}$" loop such that $\mathcal{H}_b = \mathcal{H}_{i_k-1}$. By construction, the set $\mathcal{Y}$ always contains the label predicted by the adversary, so $y_{i_k} \in \mathcal{Y}$. Consider the iteration of the "for $y \in \mathcal{Y}$" loop such that $y = y_{i_k}$. By the induction hypothesis, $x_i \in \mathrm{path}(\mathcal{H}_{i_k-1})$, and since $\mathcal{H}_{b'} \subseteq \mathcal{H}_b = \mathcal{H}_{i_k-1}$, it follows that $x_{i_k} \in \mathrm{path}(\mathcal{H}_{b'})$. Seeing as $|x_{i_k}| < d$, in the last line of this iteration of the "for $y \in \mathcal{Y}$" loop, the node $x_{i_{k+1}} := x_{i_k} \circ y_{i_k}$ is added to $\mathcal{Q}$. This guarantees that $x_{i_{k+1}}$ will eventually be popped from

$\mathcal{Q}$ and added to the sequence returned by CONSTRUCTSEQUENCE. Once a node has been added to the sequence, it is never removed.

Notice that $|x_{i_{k+1}}| = |x_{i_k}| + 1 = k + 1$, satisfying Item 1. Therefore, it remains to show Item 2, namely, to show that $x_{i_{k+1}} \in \mathrm{path}\big(\mathcal{H}_{i_{k+1}-1}\big)$.

Indeed, by the induction hypothesis, $x_i \in \mathrm{path}(\mathcal{H}_{i_k-1})$, and in the iteration of the "for $y \in \mathcal{Y}$" discussed above, $\mathcal{H}_b = \mathcal{H}_{i_k-1}$, $\mathcal{H}_{b'} = \mathcal{H}_{i_k}$, and $\mathcal{H}_{b'} = \{h \in \mathcal{H}_b : h(x_{i_k}) = y_{i_k}\}$. Hence,

$$\forall h \in \mathcal{H}_{i_k} : \; x_{i_k} \in \mathrm{path}(h) \;\wedge\; h(x_{i_k}) = y_{i_k}.$$

Seeing as $x_{i_{k+1}} = x_{i_k} \circ y_{i_k}$ This implies that

$$\forall h \in \mathcal{H}_{i_k} : \; x_{i_{k+1}} \in \mathrm{path}(h).$$

Item 2 follows from the inclusion $\mathcal{H}_{i_{k+1}-1} \subseteq \mathcal{H}_{i_k}$. □

B.3 Proof

Finally, we complete the proof of the lower bound.

Proof of Theorem B.1. Fix $d_0 = 800$ and assume $d \geq d_0$. Seeing as $\mathsf{LD}(\mathcal{H}) = d$, $\mathcal{H}$ shatters the tree T_d. By replacing $\mathcal{H}$ with a suitable subset of $\mathcal{H}$ of cardinality 2^{d+1}, renaming the elements in the domain of $\mathcal{H}$ to nodes of T_d, and restricting the domain of each function in $\mathcal{H}$ to T_d, assume without loss of generality that $\mathcal{H} \subseteq \{0,1\}^{T_d}$, $|\mathcal{H}| = 2^{d+1}$, and $\mathcal{H}$ shatters T_d.

Consider the loop "for $t \in [n]$" in Algorithm 1, and let

$$S = \{s_1, s_2, \ldots, s_m\} = \big\{t \in [n] : \; r_t \in [\varepsilon, 1-\varepsilon]\big\}$$

be the set of indices where the adversary forces a mistake, such that the learner makes at least $m = |S|$ mistakes. Let $M = \sqrt{d}/10$, and assume for contradiction that $m \leq M$.

By Claim B.5, there exists $t \in [n]$ such that $|x_t| = d$ (i.e., x_t is a leaf in T_d) and $x_t \in \mathrm{path}(\mathcal{H}_{t-1})$, namely,

$$\forall h \in \mathcal{H}_{t-1} : \; x_t \in \mathrm{path}(h).$$

Seeing as x_t is a leaf,

$$\forall h \in \mathcal{H}_{t-1} : \; \mathrm{path}(x_t) = \mathrm{path}(h). \tag{5}$$

By construction,

$$\mathcal{H}_t \subseteq \Big\{h \in \mathcal{H} : \; (\forall i \in [t] : \; h(x_i) = y_i)\Big\},$$

and $\mathcal{H}_t$ is not empty. Fix some $h^* \in \mathcal{H}_t \subseteq \mathcal{H}_{t-1}$. By Item (a) in Claim B.3, $\mathrm{path}(x_t) = \mathrm{path}(h^*)$ is a subsequence of $x_1, x_2, \ldots, x_t$, so

$$\forall h \in \mathcal{H}_t \; \forall x \in \mathrm{path}(h^*) : \; h(x) = h^*(x).$$

Seeing as $\mathcal{H}$ shatters T_d and $|\mathcal{H}| = 2^{k+1}$, if two functions $h, h^* \in \mathcal{H}$ agree on the labels for all nodes in $\mathrm{path}(h^*)$, then $h = h^*$. We conclude that $\mathcal{H}_t = \{h^*\}$ and $|\mathcal{H}_t| = 1$.

Consider the loop "for $t \in [n]$" in Algorithm 1. For each $t \in [n]$,

$$|\mathcal{H}_t| \geq \begin{cases} \varepsilon \cdot |\mathcal{H}_{t-1}| & t \in S \\ (1-\varepsilon) \cdot |\mathcal{H}_{t-1}| & t \notin S. \end{cases}$$

Hence,

$$\begin{aligned}
1 &= |\mathcal{H}_t| \\
&\geq \varepsilon^m \cdot (1-\varepsilon)^{n-m} \cdot |\mathcal{H}_0| \\
&= \varepsilon^m \cdot (1-\varepsilon)^{n-m} \cdot 2^{d+1} \\
&\geq \varepsilon^m \cdot (1-\varepsilon)^{n} \cdot 2^{d+1} \\
&\geq \varepsilon^m \cdot (1-\varepsilon)^{n_d} \cdot 2^{d+1} && \text{(by Item (b) in Claim B.3.)} \\
&\geq \varepsilon^m \cdot 2^d = 2^{-m\sqrt{d}/2+d}, && (6)
\end{aligned}$$

where the final line holds because $\varepsilon = 2^{-\sqrt{d}/2}$, $n_d = (d+1) \cdot 2^{\sqrt{d}/10+1}$, and

$$(1-\varepsilon)^{n_d} = \left(1 - 2^{-\sqrt{d}/2}\right)^{(d+1)\cdot 2^{\sqrt{d}/10+1}} \geq \frac{1}{2}$$

for our choice of $d \geq 800$. Rearranging Eq. (6) yields

$$2\sqrt{d} \leq m.$$

This is a contradiction to the assumption $m \leq M = \sqrt{d}/10$. We conclude that an adversary A following Algorithm 1 satisfies

$$\inf_{L \in \mathcal{L}_n} M_{\mathsf{tr}}(\mathcal{H}, n, L, A) \geq m > M = \sqrt{d}/10, \tag{7}$$

as desired.

To establish the "furthermore" part of the theorem, fix a length $n \in \mathbb{N}$. Let k be the largest integer such that $2^{\lceil \sqrt{k}/10 \rceil} \leq n+1$ and $k \leq d$. By Eq. (7), there exists some sequence on which the adversary can force every learning rule to make at least $\left\lceil \sqrt{k}/10 \right\rceil$ mistakes. By Theorem C.2, this implies that there exists a sequence of length $2^{\lceil \sqrt{k}/10 \rceil} - 1 \leq n$ on which the adversary can force every learning rule to make at least $\left\lceil \sqrt{k}/10 \right\rceil = \min\left\{\left\lceil \sqrt{d}/10 \right\rceil, \lfloor \log(n+1) \rfloor\right\}$ mistakes. Namely,

$$M_{\mathsf{tr}}(\mathcal{H}, n) \geq \min\left\{\left\lceil \sqrt{d}/10 \right\rceil, \lfloor \log(n+1) \rfloor\right\},$$

as in Eq. (4). □

C Sequence Length

In this section, we show that if there exists a sequence on which the adversary can force M mistakes, then a sequence of length $2^M - 1$ is sufficient, and this upper bound is tight for some classes.[19]

Definition C.1 (Minimal sequence)**.** *Let $\mathcal{X}$ be a set, let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a class, and let $M \in \mathbb{N}$.*

The minimal sequence length for forcing M mistakes for the class $\mathcal{H}$, denoted $\mathsf{MinLen}(\mathcal{H}, M)$ is

$$\mathsf{MinLen}(\mathcal{H}, M) = \inf\{n \in \mathbb{N} : \ (\exists x \in \mathcal{X}^n : \ M_{\mathsf{tr}}(\mathcal{H}, x) \geq M)\}.$$

In words, $\mathsf{MinLen}(\mathcal{H}, M)$ is the smallest integer n for which there exists a sequence of length n on which the adversary can force at least n mistakes; if no such sequence exists, then $\mathsf{MinLen}(\mathcal{H}, M) = \infty$.

Theorem C.2 (Minimal sequence bound)**.** *Let $\mathcal{X}$ be a set, and fix $M \in \mathbb{N}$. Then for any class $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$, if $\mathsf{MinLen}(\mathcal{H}, M) < \infty$ then*

$$\mathsf{MinLen}(\mathcal{H}, M) \leq 2^M - 1.$$

Furthermore, there exists a class $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ for which $\mathsf{MinLen}(\mathcal{H}, M) = 2^M - 1$.

Theorem C.2 is a corollary of the tree rank characterization of M_{tr} from Ben-David et al. (1997). For completeness, we present a direct proof of Theorem C.2 that does not directly invoke that characterization. Roughly, given an adversary A_0 that forces every learner to make at least M mistakes on a (possibly long) sequence x, we apply two modifications to obtain new adversaries

$$A_0 \rightsquigarrow A_1 \rightsquigarrow A_2.$$

A_1 forces M mistakes and has a specific structure that we call 'rigidity', but it still uses the same (possibly long) sequence x. Capitalizing on the rigid structure, A_2 selects a subsequence of x of length at most $2^M - 1$, and forces M mistakes on that subsequence.

[19] Of course, there also exist classes for which a shorter sequence is sufficient. For instance, if the class shatters (in the VC sense) a subset of the domain of cardinality M, then a sequence of length M suffices.

C.1 Rigid Adversary

Definition C.3 (Rigid adversary)**.** *Let $n \in \mathbb{N}$, let $\mathcal{X}$ be a set, and let*

$$A : \left(\bigcup_{k=0}^{n-1} \{0,1\}^{2k} \right) \times \{0,1\} \to \{0,1\}$$

be an adversary strategy for some fixed sequence $x \in \mathcal{X}^n$. We say that A is rigid if there exists a function

$$f : \bigcup_{k=0}^{n-1} \{0,1\}^k \to \{0,1,\star\}$$

such that for all $k \in \{0,1,\ldots,n-1\}$ and all $y, \hat{y} \in \{0,1\}^k$,

$$A(\hat{y}_1, y_1, \ldots, \hat{y}_k, y_k, \hat{y}_{k+1}) = \begin{cases} f(y_1,\ldots,y_k) & f(y_1,\ldots,y_k) \in \{0,1\} \\ 1 - \hat{y}_{k+1} & f(y_1,\ldots,y_k) = \star \end{cases} .$$

Note that if an adversary is rigid, then the function f that witnesses this is uniquely determined.

Claim C.4 (Rigid adversary exists)**.** *Let $n, M \in \mathbb{N}$, let $\mathcal{X}$ be a set, let $x \in \mathcal{X}^n$, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a class. Let A be an adversary strategy that forces every learner to make at least M mistakes on x. Then there exists an adversary strategy A^* such that:*

1. *A^* forces every learner to make at least M mistakes on x and A^* is rigid.*
2. *Let f be the function that witnesses the rigidity of A^*. Then for every $y \in \{0,1\}^n$, the sequence*
$$f(y_{\leq 0}), f(y_{\leq 1}), f(y_{\leq 2}), \ldots, f(y),$$
has at least M members equal to $\star$.

Proof of Claim C.4. For Item 1, consider the adversary strategy A^* that simulates an execution of A, as in Algorithm 3. In broad strokes, A^* functions as a middle-man between the learner and A. As the learner makes a sequence of predictions $\hat{y} \in \{0,1\}^n$, the adversary A^* generates a sequence of (possibly different) predictions $\tilde{y} \in \{0,1\}^n$, and sends those to the adversary A. Adversary A sees only the predictions $\tilde{y}$, and assigns labels $y \in \{0,1\}^n$, which are relayed back to the learner by A^* with no modifications.

First, observe that A^* satisfies the realizability requirement. Indeed, A^* simulates an execution of A such that the sequence of labels $y_1, \ldots, y_n$ sent by A^* to the learner is exactly the sequence of labels selected by A. Seeing as A is realizable, every sequence of labels selected by A is realizable, and therefore every sequence of labels selected by A^* must be realizable as well.

Second, observe that A^* forces every leaner to make at least M mistakes. To see this, notice that in Algorithm 3,

$$\sum_{t \in [n]} \mathbb{1}(\tilde{y}_t \neq y_t) \geq M. \tag{8}$$

Indeed, A forces every learner to make at least M mistakes, and in particular this applies to a learner that makes predictions $\tilde{y}$ as in the simulation. Furthermore, observe that A^* only alters the predictions it receives from the learner in cases when it selects a label that is accepted by A, namely,

$$\forall t \in [n] : \ \tilde{y}_t \neq \hat{y}_t \implies \tilde{y}_t = y_t. \tag{9}$$

Therefore, if $E = \{t \in [n] : \ \tilde{y}_t = \hat{y}_t\}$, then

$$\begin{aligned} \sum_{t \in [n]} \mathbb{1}(\tilde{y}_t \neq y_t) &= \sum_{t \in E} \mathbb{1}(\tilde{y}_t \neq y_t) + \sum_{t \in [n] \setminus E} \mathbb{1}(\tilde{y}_t \neq y_t) \\ &= \sum_{t \in E} \mathbb{1}(\tilde{y}_t \neq y_t) + 0 && \text{(By Eq. (9))} \\ &= \sum_{t \in E} \mathbb{1}(\hat{y}_t \neq y_t) && \text{(Defintion of } E\text{)} \end{aligned}$$

Assumptions:

- $n \in \mathbb{N}$, $\mathcal{X}$ is a set, $x \in \mathcal{X}^n$ is a fixed sequence of instances.
- $A : \left(\bigcup_{k=0}^{n-1} \{0,1\}^{2k}\right) \times \{0,1\} \to \{0,1\}$ is an adversary labeling strategy for x.

RIGIDADVERSARY:

send $x_1, \ldots, x_n$ to the learner

for $t = 1, 2, \ldots, n$:

receive prediction $\hat{y}_t$ from learner

if $A(\tilde{y}_1, y_1, \ldots, \tilde{y}_{t-1}, y_{t-1}, 0) = 0$:
$\tilde{y}_t \leftarrow 0$
else if $A(\tilde{y}_1, y_1, \ldots, \tilde{y}_{t-1}, y_{t-1}, 1) = 1$:
$\tilde{y}_t \leftarrow 1$
else:
$\tilde{y}_t \leftarrow \hat{y}_t$

send prediction $\tilde{y}_t$ to A

receive label y_t from A

send label y_t to learner

Algorithm 3: Construction of a rigid adversary, by simulating a given adversary A.

$$\leq \sum_{t \in [n]} \mathbb{1}(\hat{y}_t \neq y_t). \tag{10}$$

Combining Eqs. (8) and (10) implies that A forces at least M mistakes.

Third, we show that A^* is rigid. We claim that there exists a function $g : \{0,1\}^{\leq n-1} \to \{0,1\}^{\leq n-1}$ such that for every $t \in \{0, 1, 2, \ldots, n-1\}$,

$$(\tilde{y}_1, \ldots, \tilde{y}_t) = g(y_1, \ldots, y_t).$$

Proceed by induction on t. For the base case $t = 0$ there is nothing to prove. For the induction step, we assume the claim holds for some $t = k < n - 1$, and show that it holds for $t = k + 1$. From Algorithm 3, $\tilde{y}_{k+1}$ satisfies

$$\tilde{y}_{k+1} = \begin{cases} 0 & A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 0) = 0 \\ 1 & A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 0) = A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 1) = 1 \\ 1 - y_{k+1} & \text{otherwise} \end{cases}. \tag{11}$$

The first two cases in Eq. (11) are immediate from Algorithm 3, and the remaining case occurs when A forces a mistake at time $k + 1$, namely, when A selects $y_{k+1} = 1 - \tilde{y}_{k+1}$. Thus, $\tilde{y}_{k+1}$ is a function of $y_{\leq k+1}$ and $\tilde{y}_{\leq k}$. By the induction hypothesis, $\tilde{y}_{\leq k} = g(y_{\leq k})$, so $\tilde{y}_{k+1}$ is simply a function of $y_{\leq k+1}$. This establishes the existence of the desired function g.

Hence, A^* is rigid, as witnessed by the function

$$f(y_1, \ldots, y_k) = \begin{cases} 0 & A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 0) = 0 \\ 1 & A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 0) = A(\tilde{y}_1, y_1, \ldots, \tilde{y}_k, y_k, 1) = 1 \\ \star & \text{otherwise} \end{cases},$$

where f is a well-defined function because $\tilde{y}_{\leq k} = g(y_{\leq k})$.

We have seen that A^* is a valid (realizable) adversary that forces every learner to make at least M mistakes, and it is rigid. This concludes the proof of Item 1.

Finally, For Item 2, note that $\tilde{y}_t \neq y_t$ only if A forces a mistake at time t in the sense that A selects $y_t = 1 - b$ for any prediction $b \in \{0,1\}$ provided at time t. If A forces a mistake at time t, then A^* forces a mistake at time t as well. Therefore, if $\tilde{y}_t \neq y_t$, then $f(y_{<t}) = \star$, namely, $\tilde{y}_t$ makes mistakes only when the value of f is $\star$. By Eq. (8), $\tilde{y}_t$ makes at least M mistakes throughout the game, so there must be at least M rounds where f outputs $\star$, as desired. □

C.2 Essential Indices

Definition C.5. *Let $n, M \in \mathbb{N}$, let $\mathcal{X}$ be a set, let $x \in \mathcal{X}^n$, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a class. Let A be a rigid adversary strategy witnessed by function f. We say that an index $t \in [n]$ is essential for A for forcing M mistakes on x if there exists a sequence $y \in \{0,1\}^{t-1}$ such that $f(y) = \star$ and the sequence*

$$f(y_{\leq 0}), f(y_{\leq 1}), f(y_{\leq 2}), \ldots, f(y_{\leq t-1})$$

contains at most $M-1$ members equal to $\star$.

Claim C.6. *Let $n, M \in \mathbb{N}$, let $\mathcal{X}$ be a set, let $x \in \mathcal{X}^n$, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a class. Let A be a rigid adversary strategy. Then $[n]$ contains at most $2^M - 1$ indices that are essential for A for forcing M mistakes on x.*

Proof. For each essential index $t \in [n]$, there exists a label sequence $y \in \{0,1\}^{t-1}$ that witnesses that t is essential, as in Definition C.5. Each label sequence y is a witness for at most one index (the index $|y|+1$), so it suffices to show that the set $Y \subseteq \{0,1\}^{\leq n-1}$ of all witness label sequences is of cardinality at most $2^M - 1$.

Think of Y as a collection of nodes in the binary tree T_{n-1} (Definition A.4). By Definition C.5, if $y \in Y$, then the collection of all ancestors of y in Y has cardinality

$$\left|\left\{y_{\leq i} : \ i \in \{0,1,2,\ldots,|y|-1\}\right\} \cap Y\right| \leq M-1.$$

Namely, Y is a subtree of depth at most $d = M-1$ in the binary tree T_{n-1}.[20] Hence, the number of nodes in Y is at most

$$2^{d+1} - 1 = 2^M - 1,$$

as desired. □

C.3 Proof

Proof of Theorem C.2. If $\mathsf{MinLen}(\mathcal{H}, M) < \infty$, then there exist a sequence $x \in \mathcal{X}^n$, and an adversary A_0 that forces every learner to make at least M mistakes on x. By Claim C.4, there exists a rigid adversary A_1 that causes every learner to make at least M mistakes on x,[21] and also satisfies Item 2 in Claim C.4. Let f be the function that witnesses the rigidity of A_1. By Claim C.6, the set $I \subseteq [n]$ of indices that are essential for A_1 for forcing M mistakes on x has cardinality $k = |I| \leq 2^M - 1$.

Algorithm 4 defines a new adversary, A_2, which forces every learner to make at least M mistakes on a sequence of length k. A_2 is realizable, because A_1 is realizable.[22]

To see that adversary A_2 forces every learner to make at least M mistakes, let $y_1, \ldots, y_n$ be the sequence of labels assigned by A_2. Seeing as A_2 assigns the same labels as A_1, and A_1 satisfies Item 2 in Claim C.4, it follows that there are at least M indices $j \in [n]$ such that $f(y_{\leq j-1}) = \star$. Fix $J \subseteq [n]$ to be the first M such indices. Then $J \subseteq I$, namely, all the indices in J are essential for A_1 for forcing M mistakes on x (Definition C.5).

Therefore, for each $j \in J$, A_2 includes the instance x_j in the sequence of length k sent to the learner. Then, in round j of the n rounds simulated by A_2:

- The leaner makes a prediction $\hat{y}_j \in \{0,1\}$ corresponding to instance x_j.
- Adversary A_2 sends prediction $\hat{y}_j$ to adversary A_1. Because $f(y_{\leq j-1}) = \star$, adversary A_1 assigns the label $y_j = 1 - \hat{y}_j$. Adversary A_2 then sends that label y_j to the learner. So the learner makes a mistake on x_j.

Hence, the learner makes at least $|J| = M$ mistakes, as desired. □

[20] The depth of a subtree is s if the longest root-to-node path contains $s+1$ nodes from the subtree.

[21] This is Item 1 in Claim C.4.

[22] The argument for realizability is the same as in the proof of Claim C.4.

Assumptions:

- $n, M \in \mathbb{N}$, $\mathcal{X}$ is a set, $x \in \mathcal{X}^n$ is a fixed sequence of instances.
- $A_1 : \left(\bigcup_{k=0}^{n-1} \{0,1\}^{2k}\right) \times \{0,1\} \to \{0,1\}$ is a rigid adversary labeling strategy for x that forces every learner to make at least M mistakes on the sequence x, and satisfies Items 1 and 2 in Claim C.4.
- $I = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ is the set of indices that are essential for A for forcing M mistakes on x, and $i_1 \leq i_2 \leq \cdots \leq i_k$. By Claim C.6, $k \leq 2^M - 1$.

MINIMALADVERSARY:

send $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ to the learner
for $t = 1, 2, \ldots, n$:
  if $t \in I$:
    receive prediction $\hat{y}_t$ from learner
    send prediction $\hat{y}_t$ to A_1
    receive label y_t from A_1
    send label y_t to learner
  else:
    send prediction $\hat{y}_t = 0$ to A_1
    receive label y_t from A_1

Algorithm 4: Construction of an adversary that forces M mistakes using a sequence x of length at most $2^M - 1$. In the proof of Theorem C.2, this adversary is A_2. Internally, it simulates a rigid adversary A_1.

D Upper Bound

D.1 Statement

The following result states that the lower bound of Theorem B.1 is tight for some classes.

Theorem D.1 (Upper bound, and separation between standard and transductive online learning)**.** *For every integer $d \geq 43$, there exists a hypothesis class $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ with a domain $\mathcal{X}$ of size $|\mathcal{X}| = 2^d - 1$ such that $\mathsf{LD}(\mathcal{H}) = d$ and the following two conditions hold for all $n \in \mathbb{N}$:*

1. $M_{\mathsf{tr}}(\mathcal{H}, n) \leq 48 \cdot \sqrt{d}$.
2. $M_{\mathsf{std}}(\mathcal{H}, n) = \min\{n, d\}$.

D.2 Hypothesis Class

In this section we construct the hypothesis class for Theorem D.1.

Lemma D.2. *Let $d \in \mathbb{N}$, $d \geq 42$. Let T_d be a perfect binary tree of depth d, as in Definition A.4. Then there exists a collection of functions $\mathcal{H} \subseteq \{0,1\}^{T_d}$ such that $\mathsf{LD}(\mathcal{H}) = d + 1$ and the following two conditions hold for all $H \subseteq \mathcal{H}$ and all $X \subseteq T_d$:*

1. *If $\forall h \in H\ \forall x \in X :\ x \notin \mathrm{path}(h)\ \wedge\ h(x) = 0$, then $\min\{|H|, |X|\} < 2^{2\sqrt{d}}$.*
2. *If $\forall h \in H\ \forall x \in X :\ x \notin \mathrm{path}(h)\ \wedge\ h(x) = 1$, then $|H| < 2^{2\sqrt{d}}$ or $|X| < 3\sqrt{d}$.*

The proof employs the probabilistic method, showing that a hypothesis class sampled randomly from a suitable distribution has the desired properties with very high probability.

Proof. Let $\mathcal{P}$ be a probability distribution over hypothesis classes. Formally, $\mathcal{P} \in \Delta\Big((\{0,1\}^{T_d})^{2^{d+1}}\Big)$ is a distribution over vectors of hypotheses. Each vector $\mathcal{H} \in \operatorname{supp}(\mathcal{P})$ consists of 2^{d+1} hypotheses,

$$\mathcal{H} = (h_b)_{b \in \{0,1\}^{d+1}},$$

where for each $b \in \{0,1\}^{d+1}$, hypothesis h_b is a function $h_b : T_d \to \{0,1\}$ sampled independently as follows:

- For each $i \in [d] \cup \{0\}$: $h_b(b_{\le i}) = b_{i+1}$. (In particular, with probability 1, $\operatorname{path}(h_b) = (b_{\le 0}, b_{\le 1}, \ldots, b_{\le d})$, each entry in the vector $\mathcal{H}$ is unique, and $\mathcal{H}$ shatters T_d.)
- For each $x \in T_d \backslash \operatorname{path}(h_b)$, the bit $h_b(x) \in \{0,1\}$ is sampled $\operatorname{Ber}\Big(2^{-\sqrt{d}}\Big)$ independently of all other bits in $\mathcal{H}$, i.e., $\mathbb{P}[h_b(x) = 1] = \mathbb{P}[h_b(x) = 1 \mid \{h_{b'}\}_{b' \neq b}, \{h_b(x')\}_{x' \neq x}] = 2^{-\sqrt{d}}$.

In words, for all nodes on the path in the tree corresponding to b, the function h_b assigns a label according to b, and for all other nodes, h_b assigns a label of 1 with probability $2^{-\sqrt{d}}$, and a label of 0 otherwise. In particular, the collection $\mathcal{H}$ Littlestone-shatters the tree T_d.

Fix $B \subseteq \{0,1\}^{d+1}$ and $X \subseteq T_d$, and let $E(B, X, y)$ denote the event

$$\{\forall b \in B \; \forall x \in X : \; x \notin \operatorname{path}(h_b) \; \wedge \; h_b(x) = y\}. \tag{12}$$

Seeing as each off-path label $h_b(x) \in \{0,1\}$ is sampled independently,

$$\begin{aligned}
\mathbb{P}_{\mathcal{H}\sim\mathcal{P}}[E(B, X, 0)] &= \prod_{(b,x) \in B \times X} \mathbb{P}_{\mathcal{H}\sim\mathcal{P}}[x \notin \operatorname{path}(h_b) \; \wedge \; h_b(x) = 0] \\
&\le (1 - 2^{-\sqrt{d}})^{|B \times X|}.
\end{aligned} \tag{13}$$

Hence,

$$\begin{aligned}
&\mathbb{P}_{\mathcal{H}\sim\mathcal{P}}\Big[\exists B \subseteq \{0,1\}^{d+1} \; \exists \, X \subseteq T_d : \; E(B, X, 0) \; \wedge \; \min\{|B|, |X|\} \ge 2^{2\sqrt{d}}\Big] \\
&\quad = \mathbb{P}_{\mathcal{H}\sim\mathcal{P}}\Big[\exists B \subseteq \{0,1\}^{d+1} \; \exists \, X \subseteq T_d : \; E(B, X, 0) \; \wedge \; |B| = |X| = \Big\lceil 2^{2\sqrt{d}} \Big\rceil\Big] \\
&\quad \le \binom{|\{0,1\}^{d+1}|}{\lceil 2^{2\sqrt{d}} \rceil} \binom{|T_d|}{\lceil 2^{2\sqrt{d}} \rceil} (1 - 2^{-\sqrt{d}})^{2^{4\sqrt{d}}} && \text{(union bound, Eq. (13))} \\
&\quad < \binom{2^{d+1}}{2^{2\sqrt{d}} + 1}^2 \cdot (1 - 2^{-\sqrt{d}})^{2^{4\sqrt{d}}} && \text{($2^{2\sqrt{d}}$ may not be an integer; formally, we use the generalized binomial coefficient, or simply skip to the next line)} \\
&\quad < 2^{2 \cdot (d+1) \cdot \left(2^{2\sqrt{d}} + 1\right)} \cdot e^{-2^{-\sqrt{d}} \cdot 2^{4\sqrt{d}}} && \left(\tbinom{n}{k} < n^k \text{ for } k \ge e;\; 1 + x \le e^x \text{ for } x \in \mathbb{R}\right) \\
&\quad < 2^{2 \cdot (d+2) \cdot 2^{2\sqrt{d}}} \cdot 2^{-2^{-\sqrt{d}} \cdot 2^{4\sqrt{d}}} && (2^{2\sqrt{d}} \ge d + 1 \text{ for } d \ge 0) \\
&\quad = 2^{2^{2\sqrt{d}} \cdot (2d + 4 - 2^{\sqrt{d}})} \\
&\quad < 2^{-2^{2\sqrt{d}}}. && (2d + 4 - 2^{\sqrt{d}} < -1 \text{ for } d \ge 42)
\end{aligned} \tag{14}$$

Similarly,

$$\mathbb{P}_{\mathcal{H}\sim\mathcal{P}}[\forall b \in B \; \forall x \in X : \; x \notin \operatorname{path}(h_b) \; \wedge \; h_b(x) = 1] \le 2^{-\sqrt{d} \cdot |B \times X|}, \tag{15}$$

so

$$\mathbb{P}_{\mathcal{H}\sim\mathcal{P}}\Big[\exists B \subseteq \{0,1\}^{d+1} \; \exists \, X \subseteq T_d : \; E(B, X, 0) \; \wedge \; |H| \ge 2^{2\sqrt{d}} \; \wedge \; |X| \ge 3\sqrt{d}\Big]$$

$$
\begin{aligned}
&\leq \binom{|\{0,1\}^{d+1}|}{\left\lceil 2^{2\sqrt{d}} \right\rceil} \binom{|T_d|}{\left\lceil 3\sqrt{d} \right\rceil} \cdot 2^{-\sqrt{d}\cdot 2^{2\sqrt{d}}\cdot 3\sqrt{d}} && \text{(union bound, Eq. (15))} \\
&\leq \binom{2^{d+1}}{2^{2\sqrt{d}}+1} \binom{2^{d+1}}{3\sqrt{d}+1} \cdot 2^{-3d\cdot 2^{2\sqrt{d}}} \\
&< 2^{(d+1)\cdot\left(2^{2\sqrt{d}}+1\right)} \cdot 2^{(d+1)\cdot\left(3\sqrt{d}+1\right)} \cdot 2^{-3d\cdot 2^{2\sqrt{d}}} && \left(\binom{n}{k} < n^k \text{ for } k \geq e\right) \\
&< 2^{(d+1)\cdot\left(2^{2\sqrt{d}}+3\sqrt{d}+2\right)} \cdot 2^{-3d\cdot 2^{2\sqrt{d}}} \\
&< 2^{2d\cdot 2^{2\sqrt{d}}} \cdot 2^{-3d\cdot 2^{2\sqrt{d}}} && \text{(for } d \geq 4\text{)} \\
&< 2^{-d2^{\sqrt{d}}}. && (16)
\end{aligned}
$$

Applying a union bound to Eqs. (14) and (16) gives

$$\mathbb{P}_{\mathcal{H}\sim\mathcal{P}}[\mathcal{H} \text{ satisfies Items 1 and 2}] \geq 1 - 2^{-2^{2\sqrt{d}}} - 2^{-d2^{\sqrt{d}}} \geq 1 - 10^{-100}.$$

In particular, there exists a collection $\mathcal{H}$ that satisfies Items 1 and 2. Furthermore, this collection has $\mathsf{LD}(\mathcal{H}) = d+1$ (namely, $\mathsf{LD}(\mathcal{H}) \geq d+1$ because it shatters T_d; and $\mathsf{LD}(\mathcal{H}) \leq d+1$ because $|\mathcal{H}| = 2^{d+1}$). □

D.3 Algorithm

In this section we describe Algorithms 5, 6a, and 6c, which together constitute the learning algorithm that achieves the $O\left(\sqrt{d}\right)$ mistake upper bound in the transductive setting, as in Theorem D.1. See Section 2.3 for a general overview of these algorithms.

D.3.1 How Experts Work

We start with some preliminary remarks about experts in Algorithms 5, 6a, and 6c.

Experts. A tuple $e = (S, u, H)$ defines an expert that can make predictions using the procedure EXPERT.PREDICT$(e, \cdot)$. The tuple e reflects two kinds of information:

1. *Knowledge.* Information that the expert *knows* with certainty. Specifically, this reflects the labels $y_1, y_2, \ldots$ sent by the adversary so far. All experts see the labels sent by the adversary, so this knowledge is the same for all experts.
2. *Assumptions.* At certain times, experts make *assumptions* about things that are not known for certain. Specifically, experts assume that certain nodes x are on-path ($x \in \mathrm{path}(h)$) or off-path ($x \notin \mathrm{path}(h)$) with respect to the correct labeling function $h : T_d \to \{0,1\}$. Assumptions are simply guesses that may be wrong, and therefore when an expert needs to make such an assumption, it splits into two experts (as described below), with one expert assuming $x \in \mathrm{path}(h)$, and the other expert assuming $x \notin \mathrm{path}(h)$. This ensures that there always exists an expert for which all assumptions are correct.

In greater detail, the contents of the state tuple $e = (S, u, H)$ represents the knowledge and assumptions of the expert as follows:

- $u \in T_d$ – This single node encodes everything the expert knows and assumes about which of the nodes labeled so far are on-path. Observe that if $v_1, v_2, \ldots, v_k \in T_d$ are nodes that are assumed to be on-path (and all these assumptions are consistent), then these k assumptions can be represented succinctly by assigning $u = v_{i^*}$ where v_{i^*} is the deepest node among $v_1, v_2, \ldots, v_k$. Therefore, u simply holds the deepest node in the tree that is known or assumed to be on-path. At the start of the algorithm, this value is initialized to be $u = \lambda$, because the root is known to be on-path regardless of the target function.
- $S \subseteq T_d$ – the 'danger zone', as described in Section 2.3.4. This is a collection that contains all nodes in the prefix $x_{\leq t_{\mathsf{max}}} = (x_1, x_2, \ldots, x_{t_{\mathsf{max}}})$ of the sequence to be classified that have not been labeled yet and *might* be on-path for the true labeling function h given what

the expert knows and assumes so far. However, S is not required to contain ancestors of nodes that are assumed to be on-path. Initially, S equals the prefix $x_{\leq t_{\mathsf{max}}}$. As information accumulates, nodes that cannot be on-path are removed from S. For instance, if $x_i \in T_d$ is assigned label $y_i \in \{0,1\}$ by the adversary, then any $(1-y_i)$-descendant of x_i (including x_i itself) may safely be removed from S.

- $H \subseteq \{0,1\}^{T_d}$ – the version space of the experts, i.e., the collection of all functions that could be the correct labeling function given everything that the expert knows and assumes. Initially, H contains all functions in $\mathcal{H}$. As information accumulates, some functions are ruled out. Specifically, a function h can be removed from H for two reasons: (i) the adversary assigns a label $y \neq h(x)$ to some node $x \in T_d$; (ii) the expert makes an assumption that some $x \in T_d$ is on-path for the correct labeling function but $x \notin \mathrm{path}(h)$, or vice versa, the expert assumes that x is off-path for the correct labeling function but $x \in \mathrm{path}(h)$.

Updates and splits. An expert can be modified using the procedure EXPERT.EXTENDEDUPDATE$(e,\cdot,\cdot)$. This procedure either returns a single modified tuple (S,u,H) (in the first two return statements in the procedure), in which case we think of the expert as being *updated*; or alternatively, the procedure returns two tuples $e_\in = (S_\in, u_\in, H_\in)$ and $e_\notin = (S_\notin, u_\notin, H_\notin)$ (in the third return statement), in which case we think of the expert as being *split* into two experts. The expert $e_\in$ corresponds to adding an assumption that the most recently presented node x_t is on-path for the correct labeling function, and $e_\notin$ corresponds to adding the opposite assumption.

Ancestry. At the end of each iteration of the outer 'for' loop in Algorithm 5, for each expert $e \in E_{t+1}$ there exists a unique *ancestry* sequence $\mathrm{ancestry}(e) = (e_1, e_2, \ldots, e_{t+1})$ such that $e_1 = (\{x_1, \ldots, x_{t_{\mathsf{max}}}\}, \lambda, \mathcal{H})$ is the initial single expert that was created before the start of the outer 'for' loop, $e_{t+1} = e$ is the latest version of the expert, and for each $i \in [t]$, the expert e_{i+1} was created by an execution of EXPERT.BASICUPDATE$(e_i,\cdot,\cdot)$ possibly followed by an execution of EXPERT.EXTENDEDUPDATE.[23]

D.4 Analysis

In this section we prove our main result, Theorem D.1.

D.4.1 Assumption-Consistent Expert

Occasionally, when an expert is updated, it makes an assumption about whether the most-recently presented node x_t is on-path or off-path with respect to the true labeling function h. In these updates, the expert is split into two: one expert assumes that $x_t \in \mathrm{path}(h)$, and the other assumes $x_t \notin \mathrm{path}(h)$. Clearly, by splitting into two in this manner, we preserve the invariant that the set of experts always contains a 'vindicated' expert e^* such that all the assumptions made by e^* are correct. This simple observation is made formal in the following definition and claim.

Definition D.3 (Assumption consistency)**.** *For an expert $e \in E_{t+1}$ with $\mathrm{ancestry}(e) = (e_1, e_2, \ldots, e_{t+1})$, and an index $i \in [t]$, we say that the $i \to (i+1)$ update of e was assumption-consistent with a function $h : T_d \to \{0,1\}$ if one of the following conditions holds:*

- $e_{i+1} =$ EXPERT.BASICUPDATE(e_i, x_i, y_i)*; or*

[23] Note that in this paper, we use genealogical metaphors in two distinct contexts that should not be confused. First, as is customary, we use "child", "parent", "ancestor" and "descendant" to describe relations between nodes in the binary tree T_d, which constitutes the domain of our hypothesis class. Separately from that, we use "ancestor" and "descendant" to describe relations between experts.

This overlap in terminology can partially be excused by the fact that the history of experts also forms a binary tree. Indeed, initially there is a single expert (the root of the tree), and experts can split into two, corresponding to a node having two children as in a binary tree. Seeing as experts cannot merge, the expert history corresponds precisely to a binary tree. (However, the domain T_d is a *perfect* binary tree, whereas the binary tree corresponding to expert genealogy need not be balanced).

To reduce confusion, we use $\mathrm{path}(\cdot)$ only for nodes in T_d, and $\mathrm{ancestry}(\cdot)$ only for experts, even though these operators are mathematically equivalent (however, $\mathrm{path}(\cdot)$ is defined not only for nodes in T_d but also for functions $T_d \to \{0,1\}$).

Assumptions:

- $d, n \in \mathbb{N}$, λ is the empty string.
- $\mathcal{H} \subseteq \{0,1\}^{T_d}$ is the class that exists by Lemma D.2.
- $x_1, x_2, \ldots, x_n \in T_d$ are points to be classified.

TRANSDUCTIVELEARNER($\mathcal{H}$, d, $(x_1, x_2, \ldots, x_n)$):

$t \leftarrow 0$, $t_{\mathsf{max}} \leftarrow 2^{4\sqrt{d}}$
$e \leftarrow (\{x_1, \ldots, x_{t_{\mathsf{max}}}\}, \lambda, \mathcal{H})$ ▷ The initial expert. An expert is defined by a 3-tuple.
$w(e) \leftarrow 1$ ▷ Assign the initial expert a weight of 1.
$E_1 \leftarrow \{e\}$ ▷ E_t is the set of experts used for predicting $\hat{y}_t$.
$E_2, \ldots, E_n, E_{n+1} \leftarrow \varnothing$

for $t \leftarrow 1, 2, \ldots, n$:

$\hat{y}_t \leftarrow \mathbb{1}\left(\sum_{e \in E_t} w(S) \cdot \text{EXPERT.PREDICT}(e, x_t) \geq \frac{1}{2}\right)$ ▷ A weighted majority, using Algorithm 6a.

send prediction $\hat{y}_t$ to adversary

receive correct label $y_t \in \{0,1\}$ from adversary

for $e \in E_t$: ▷ Update the experts.
$e \leftarrow \text{EXPERT.BASICUPDATE}(e, x_t, y_t)$ ▷ Remove functions that disagree with the label y_t from the version space.
if $\text{EXPERT.PREDICT}(e, x_t) = y_t$:
$E_{t+1} \leftarrow E_{t+1} \cup \{e\}$ ▷ If expert e made a correct prediction, no further update is needed.
else:
$U \leftarrow \text{EXPERT.EXTENDEDUPDATE}(e, x_t, y_t)$ ▷ If e made a mistake, update e using Algorithm 6c. This might cause e to be split into two experts.
for $e' \in U$:
$E_{t+1} \leftarrow E_{t+1} \cup \{e'\}$ ▷ Add updated expert(s) to E_{t+1}.
$w(e') \leftarrow w(e)/(2 \cdot |U|)$ ▷ When e makes a mistake, its weight is decreased by a factor of 2 and then split equally between its descendants.

Algorithm 5: A transductive online learning algorithm that makes at most $O\left(\sqrt{d}\right)$ mistakes. It is a variant of the multiplicative weights algorithm that employs splitting experts. Namely, we start with a single expert, and when an expert makes a mistake it may split into two experts. The behavior of the experts is defined in Algorithms 6a and 6c.

- *e_{i+1} was the single expert returned when executing* $\text{EXPERT.EXTENDEDUPDATE}(e'_i, x_i, y_i)$ *for* $e'_i = \text{EXPERT.BASICUPDATE}(e_i, x_i, y_i)$*; or*
- *Executing* $\text{EXPERT.EXTENDEDUPDATE}(e'_i, x_i, y_i)$ *with* $e'_i = \text{EXPERT.BASICUPDATE}(e_i, x_i, y_i)$ *returned two experts* $(S_\in, u_\in, H_\in)$ *and* $(S_\notin, u_\notin, H_\notin)$ *(as in the third return statement), and furthermore,*

$$e_{i+1} = \begin{cases} (S_\in, u_\in, H_\in) & x_i \in \text{path}(h) \\ (S_\notin, u_\notin, H_\notin) & x_i \notin \text{path}(h). \end{cases} \tag{17}$$

Assumptions:

- $d \in \mathbb{N}$, $x \in T_d$.
- $e = (S, u, H)$ is a tuple that defines an expert:
  - $S \subseteq T_d$ – a collection of nodes that could be on-path for the true labeling function given what the expert knows and assumes.
  - $u \in T_d$ – the deepest node known or assumed to be on-path by the expert.
  - $H \subseteq \{0,1\}^{T_d}$ – the collection of all functions that could be the correct labeling function given what the expert knows and assumes.

EXPERT.PREDICT(e, x):

$(S, u, H) \leftarrow e$ ▷ Unpack the state that defines the expert.

if $|H| \leq 2^{2\sqrt{d}}$:
  return HALVING.PREDICT(H, x) ▷ Once H becomes small enough, simulate the Halving algorithm (Algorithm 7). [Case I]

if $x \preccurlyeq u$:
  return $b \in \{0,1\}$ such that $x \preccurlyeq_b u$ ▷ u is assumed to be on-path. If u is a b-decendant of x, then the correct label for x must be b. [Case II]

return $\mathbb{1}(|\{x' \in S : x \preccurlyeq_1 x'\}| > |S|/3)$ ▷ Output some $b \in \{0,1\}$ such that more than $1/3$ of suspected on-path nodes are b-decendants of x, if such a b exists. Otherwise (when at least $1/3$ of S are non-descendants of x), output 0. [Cases III to VI]

Algorithm 6a: A subroutine of Algorithm 5 that defines how an expert makes predictions.

Assumptions:

- x, e, S, u, H – as in Algorithm 6a.
- y – the correct label for x, as selected by the adversary.

EXPERT.BASICUPDATE(e, x, y):

$(S, u, H) \leftarrow e$ ▷ Unpack the state that defines the expert.

$H \leftarrow$ HALVING.UPDATE(H, x, y) ▷ Update the version space, as in the Halving algorithm (Algorithm 7).

return (S, u, H)

Algorithm 6b: A subroutine of Algorithm 5 that defines how an expert is updated each time that a label is selected by the adversary.

We say that an expert $e \in E_{t+1}$ is assumption-consistent with h if for all $i \in [t]$, the $i \to (i+1)$ update of e was assumption-consistent with h.

Claim D.4 (Existence of assumption-consistent expert)**.** *Let $d, n, t \in \mathbb{N}$, $t \leq n$, let $\mathcal{H} \subseteq \{0,1\}^{T_d}$, let $x_1, \ldots, x_n \in T_d$, and let $h : T_d \to \{0,1\}$. Consider an execution of*

$$\text{TRANSDUCTIVELEARNER}(\mathcal{H}, d, (x_1, x_2, \ldots, x_n))$$

as in Algorithm 5. Then, at the end of the t-th iteration of the outer 'for' loop in TRANSDUCTIVELEARNER*, there exists a unique expert $e^*_{t+1} \in E_{t+1}$ that is assumption-consistent with h.*

Assumptions:

- d, x, e, S, u, H – as in Algorithm 6a.
- y – the correct label for x, as selected by the adversary.

EXPERT.EXTENDEDUPDATE(e, x, y):

$(S, u, H) \leftarrow e$ ▷ Unpack the state that defines the expert.

if $|H| \leq 2^{2\sqrt{d}}$:
  return $\{(S, u, H)\}$ ▷ If the version space is small, we just simulate the Halving algorithm, so the update is complete. [Case III]

for $b \in \{0, 1\}$:
  $S_b \leftarrow \{x' \in S : x \preccurlyeq_b x'\}$ ▷ Set of suspected on-path nodes that are b-descendant of x.

if $|S_{(1-y)}| > |S|/3$:
  $S' \leftarrow S \setminus S_{(1-y)}$ ▷ At least $1/3$ of suspected on-path nodes were b-decendants of x, and therefore the expert predicted label $\hat{y} = b$. But the correct label was $y = 1 - b$. Remove all b-descendants of x from S. [Case IV]
  return $\{(S', u, H)\}$

else:
  $S_\notin \leftarrow S$; $u_\notin \leftarrow u$ ▷ Split e in two. First, construct $e_\notin$ to be an updated version of e after adding the assumption that $x \notin \text{path}(h)$ for the correct labeling function h.
  $H_\notin = \{h \in H : x \notin \text{path}(h)\}$
  $e_\notin \leftarrow (S_\notin, u_\notin, H_\notin)$
  $S_\in \leftarrow S_0 \cup S_1$ ▷ Next, construct $e_\in$ to be an updated version of e adding the assumption $x \in \text{path}(h)$. $S_\in$ contains only nodes that are descendants of x.
  $u_\in \leftarrow u$ ▷ $u_\in$ represents updating the prior assumption that u is on path by adding that x is also on path.
  if $u_\in \preccurlyeq x$:
    $u_\in \leftarrow x$
  $H_\in = \{h \in H : x \in \text{path}(h)\}$ ▷ $H_\in$ is obtained by updating the version space to include only function where x is on path.
  $e_\in \leftarrow (S_\in, u_\in, H_\in)$
  return $\{e_\notin, e_\in\}$ ▷ [Cases V and VI]

Algorithm 6c: A subroutine of Algorithm 5 that defines how an expert is updated (and possibly split into two) when it makes a mistake.

Proof. We prove by induction that, for all $s \in [t+1]$, E_s contains a unique expert that is assumption-consistent with h. The base case $s = 1$ is clear, because E_1 contains only a single expert that was never modified. For the induction step, let e_s^* be the unique assumption-consistent expert in E_s, and consider the $s \to (s+1)$ update. Notice that by Definition D.3,

- For all $e \in E_s \setminus \{e_s^*\}$, every expert $e' \in E_{s+1}$ such that e' was created from e by executing EXPERT.BASICUPDATE(e_s, x_s, y_s) possibly followed by an execution of EXPERT.EXTENDEDUPDATE is not assumption-consistent with h; and
- Either EXPERT.BASICUPDATE$(e_s^*, x_s, y_s) \in E_{s+1}$ and EXPERT.EXTENDEDUPDATE(e_s^*, x_s, y_s) is not executed (e_s^* is added to E_{s+1} with just a basic update), or precisely one of the experts that were created from e_s^* by executing EXPERT.EXTENDEDUPDATE and added to E_{s+1} is assumption-consistent with h.

Assumptions:

- $\mathcal{X}$ a set, $k \in \mathbb{N}$.
- $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ is a finite hypothesis class.
- $x, x_1, \ldots, x_k \in \mathcal{X}$, $y \in \{0,1\}$.

HALVING($\mathcal{H}$, $(x_1, x_2, \ldots, x_k)$):

$\mathcal{H}_1 \leftarrow \mathcal{H}$

for $i \in [k]$:

$\hat{y}_i \leftarrow$ HALVING.PREDICT($\mathcal{H}, x_i$)

send prediction $\hat{y}_i$ to adversary

receive correct label $y_i \in \{0,1\}$ from adversary

$\mathcal{H}_{i+1} \leftarrow$ HALVING.UPDATE($\mathcal{H}_i, x_i, y_i$)

HALVING.PREDICT($\mathcal{H}$, x):

return $\mathbb{1}\left(\frac{1}{|\mathcal{H}|}\sum_{h\in\mathcal{H}} h(x) \geq \frac{1}{2}\right)$

HALVING.UPDATE($\mathcal{H}$, x, y):

return $\left\{h \in \mathcal{H} : \ h(x) = y\right\}$

Algorithm 7: This is the well-known halving algorithm. The experts in Algorithms 6a and 6c simulate this algorithm once their version space becomes small enough.

Seeing as the $s \rightarrow (s+1)$ update executes EXPERT.BASICUPDATE and EXPERT.EXTENDEDUPDATE at most once for each $e \in E_s$, it follows that E_{s+1} contains precisely one expert that is assumption-consistent with h. □

An expert $e = (S, u, H)$ that is assumption-consistent with the correct labeling function enjoys two simple properties. The first property is that the node u in the expert encodes correct information about which previously seen nodes are on-path for the correct labeling function.

The second property is that the set S contains all future nodes that are on-path for the correct labeling function and are also deeper in the tree than all nodes assumed to be on-path so far. These two properties are formalized in the following claim.

Claim D.5 (Properties of assumption-consistent expert)**.** *Let $d, n, t \in \mathbb{N}$, $t \leq n+1$, let $\mathcal{H} \subseteq \{0,1\}^{T_d}$, let $x_1, \ldots, x_n \in T_d$. Consider an execution of*

$$\text{TRANSDUCTIVELEARNER}(\mathcal{H}, d, (x_1, x_2, \ldots, x_n))$$

as in Algorithm 5. Assume that the adversary selects labels $y_1, y_2, \ldots, y_n \in \{0,1\}$ that are consistent with some function $h : T_d \to \{0,1\}$. Let $e_t^* = (S_t^*, u_t^*, H_t^*) \in E_t$ be the unique expert in E_t that is assumption-consistent with h.[24] *Then the following two properties hold:*

1. $u_t^* \in \text{path}(h)$.
2. $\{x \in \{x_t, x_{t+1}, \ldots, x_{t_{\max}}\} : \ x \in \text{path}(h) \ \wedge \ x \not\preccurlyeq u_t^*\} \subseteq S_t^*$.

Proof of Claim D.5. The proof proceeds by induction on t. For the base case $t = 1$, E_1 contains a single expert $e_1^* = (S_1^*, u_1^*, H_1^*)$ where $u_1^* = \lambda$ is the root of T_d. Indeed, $\lambda \in \text{path}(h)$ for

[24] Recall that e_t^* exists by Claim D.4.

any function $h : T_d \to \{0,1\}$. This establishes the base case for Item 1. Additionally, $S_1^* = \{x_1, x_2, \ldots, x_{t_{\mathsf{max}}}\}$, satisfying the base case for Item 2.

For the induction step, we assume that the claim holds for some integer $t = i$, and show that it holds for $t = i+1$ as well. First, we establish Item 1. If $e_{i+1}^* = \text{EXPERT.BASICUPDATE}(e_i^*, x_i, y_i)$, then the claim is immediate because $u_{i+1}^* = u_i^* \in \text{path}(h)$. Otherwise, by Definition D.3 and the first first two return statements in EXPERT.EXTENDEDUPDATE, either $e_{i+1}^* = (S_{i+1}^*, u_{i+1}^*, H_{i+1}^*)$ has $u_{i+1}^* = u_i^* \in \text{path}(h)$, in which case the claim is immediate, or else e_{i+1}^* satisfies Eq. (17), namely,

$$e_{i+1}^* = \begin{cases} (S_\in, u_\in, H_\in) & x_i \in \text{path}(h) \\ (S_\notin, u_\notin, H_\notin) & x_i \notin \text{path}(h). \end{cases}$$

As defined in EXPERT.EXTENDEDUPDATE, $u_\in$ is equal either to u_i^* or to x_i, so if $x_i \in \text{path}(h)$ then

$$u_{i+1}^* = u_\in \in \{u_i^*, x_i\} \subseteq \text{path}(h).$$

On the other hand, if $x_i \notin \text{path}(h)$ then we get $u_{i+1}^* = u_\notin = u_i^* \in \text{path}(h)$. We see that in all cases, $u_{i+1}^* \in \text{path}(h)$ as desired. This concludes the proof of Item 1.

For Item 2, again, if $e_{i+1}^* = \text{EXPERT.BASICUPDATE}(e_i^*, x_i, y_i)$, then the claim is immediate because $S_{i+1}^* = S_i^*$ and $u_{i+1}^* = u_i^*$. Otherwise, consider the various ways in which u_{i+1}^* and S_{i+1}^* can be assigned by EXPERT.EXTENDEDUPDATE. In the first return statement, $u_{i+1}^* = u_i^*$ and $S_{i+1}^* = S_i^*$, and the claim is immediate.

The second return statement assigns $u_{i+1}^* = u_i^*$ and $S_{i+1}^* = S_i^* \setminus S_{1-y_i}$, where S_{1-y_i} is the set of $(1-y_i)$-descendants of x_i (including x_i itself). Notice that regardless of whether x_i is on-path for the correct labeling function h or not, none of the $(1-y_i)$-descendants of x_i (except possibly x_i itself) can be on-path for h, because h assigns a label y_i to x_i. And seeing as Item 2 only requires that S_{i+1}^* contain nodes from $\{x_{i+1}, x_{i+2}, \ldots, x_{t_{\mathsf{max}}}\}$, it is also safe to remove x_i. Therefore, removing S_{1-y_i} preserves Item 2.

For the third return statement, there are two possibilities. The first possibility is that $u_{i+1}^* = u_\notin = u_i^*$ and $S_{i+1}^* = S_\notin = S_i^*$, in which case the claim is immediate. The second possibility assigns $u_{i+1}^* = u_\in$, and $S_{i+1}^* = S_\in = S_0 \cup S_1$, namely, S_{i+1}^* is constructed by removing the non-descendants of x_i from S_i^*. By Eq. (17), this happens when $x_i \in \text{path}(h)$, so all non-descendants of x_i or either off-path for h, or they are ancestors of x_i. Seeing as $x_i \in \text{path}(h)$ and $u_i^* \in \text{path}(h)$, and $u_\in$ is the deeper node between these two, any node that is an ancestor of x_i is also an ancestor of $u_{i+1}^* = u_\in$. Thus, all the nodes removed or either off-path for h, or they are ancestors of u_{i+1}^*, satisfying Item 2. (Similarly, any node that is an ancestor of u_i^* is also an ancestor of u_{i+1}^*, so we do not need to add any new nodes to S_{i+1}^* that are not included in S_i^*.)

We see that in all cases, Item 2 is preserved, as desired. □

D.4.2 Transition to Halving

Claim D.6. *Let $d, n, t \in \mathbb{N}$, $d \geq 16$, let $\mathcal{H} \subseteq \{0,1\}^{T_d}$, and let $x_1, \ldots, x_n \in T_d$. Consider an execution of*

$$\text{TRANSDUCTIVELEARNER}(\mathcal{H}, (x_1, x_2, \ldots, x_n))$$

as in Algorithm 5. Let $t > t_{\mathsf{max}} = 2^{4\sqrt{d}}$ and let $e = (S, u, H) \in E_t$ be an expert. Then

$$|H| \leq 2^{2\sqrt{d}}.$$

Proof of Claim D.6. Assume for contradiction that $|H| > 2^{2\sqrt{d}}$. Let $H' \subseteq H$ be an arbitrary subset of size $2^{2\sqrt{d}} + 1$. Let

$$P = \cup_{h \in H'} \text{path}(h).$$

Seeing as each root-to-leaf path contains $d+1$ nodes,

$$|P| \leq |H'| \cdot (d+1) \leq \left(2^{2\sqrt{d}} + 1\right) \cdot (d+1) \leq d2^{2\sqrt{d}+1}. \tag{18}$$

Let $y_1, y_2, \ldots, y_t$ be the labels provided by the adversary in the first t_{max} iterations. The line in EXPERT.BASICUPDATE constructing H using $\text{HALVING.UPDATE}(H, x, y)$ ensures that

$$\forall h \in H \; \forall i \in [t_{\mathsf{max}}] : \; h(x_i) = y_i. \tag{19}$$

Consider two cases:

• **Case I.** $\sum_{i=1}^{t_{\mathsf{max}}} y_i \le t_{\mathsf{max}}/2$. Then the set

$$X_0 = \{x_i : i \in [t_{\mathsf{max}}] \ \wedge \ y_i = 0\}$$

has cardinality $|X_0| \ge t_{\mathsf{max}}/2$. Let $X_0' = X_0 \setminus P$. By Eq. (18),

$$|X_0'| \ge \frac{t_{\mathsf{max}}}{2} - d2^{2\sqrt{d}+1} = 2^{4\sqrt{d}} - d2^{2\sqrt{d}+1}. \tag{20}$$

From the choice of X_0', the inclusion $H' \subseteq H$, and Eq. (19),

$$\forall h \in H' \, \forall x \in X_0' : \ x \notin \mathrm{path}(h) \ \wedge \ h(x) = 0. \tag{21}$$

Seeing as $|H'| > 2^{2\sqrt{d}}$, Eq. (21) and Item 1 from Lemma D.2 imply that

$$|X_0'| \le 2^{2\sqrt{d}}. \tag{22}$$

Combining Eqs. (20) and (22) yields

$$\begin{aligned} 2^{2\sqrt{d}} \ge |X_0'| &\ge 2^{4\sqrt{d}} - d2^{2\sqrt{d}+1} \\ &\ge 2^{4\sqrt{d}-1} \qquad (d \ge 16), \end{aligned}$$

which is a contradiction.

• **Case II.** $\sum_{i=1}^{t_{\mathsf{max}}} y_i > t_{\mathsf{max}}/2$. A similar argument gives a contradiction by defining

$$X_1 = \{x_i : i \in [t_{\mathsf{max}}] \ \wedge \ y_i = 1\}, \ \text{ and } \ X_1' = X_1 \setminus P.$$

As before,

$$|X_1'| \ge \frac{t_{\mathsf{max}}}{2} - d2^{2\sqrt{d}+1} \ge 2^{4\sqrt{d}} - d2^{2\sqrt{d}+1}. \tag{23}$$

for all $d \in \mathbb{N}$. However, $|H'| > 2^{2\sqrt{d}}$ and Item 2 imply that

$$|X_1'| < 3\sqrt{d}, \tag{24}$$

which is a contradiction. □

D.4.3 Performance of Best Expert

Claim D.7 (Existence of expert with large weight)**.** *Let $d, n \in \mathbb{N}$, $d \ge 16$, let $\mathcal{H} \subseteq \{0,1\}^{T_d}$, and let $x_1, \ldots, x_n \in T_d$. Consider an execution of*

$$\textsc{TransductiveLearner}(\mathcal{H}, (x_1, x_2, \ldots, x_n))$$

as in Algorithm 5. Then, at the end of the execution, there exists $e \in E_{n+1}$ such that

$$w(e) \ge 2^{-48\sqrt{d}}. \tag{25}$$

Note that the lower bound in Eq. (25) does not depend on n.

Proof. Fix a hypothesis $h \in \mathcal{H}$ such that $h(x_t) = y_t$ for all $t \in [n]$ (such an h exists because the adversary must always select a realizable label).

By Claim D.4, there exists $e_{n+1}^* \in E_{n+1}$ that is assumption-consistent with h. Let $\mathrm{ancestry}(e_{n+1}^*) = (e_1^*, e_2^*, \ldots, e_{n+1}^*)$. We argue that this ancestry sequence makes few mistakes. Specifically, for each $t \in [n]$, let $\hat{y}_t^* = \textsc{Expert.Predict}(e_t^*, x_t)$. We claim that

$$m := \sum_{t=1}^{n} \mathbb{1}(\hat{y}_t^* \ne y_t) \le 24\sqrt{d}.$$

Indeed, let $B = \{t \in [n] : \ \hat{y}_t^* \ne y_t\}$ be the set of m indices where a mistake was made. For each $t \in B$, let $e_t^* = (S, u, H)$, and note that each $t \in B$ has a corresponding execution of $\textsc{Expert.Predict}(e_t^*, x_t)$, and an execution of $e_t' = \textsc{Expert.BasicUpdate}(e_t^*, x_t, y_t)$ followed by $\textsc{Expert.ExtendedUpdate}(e_t', x_t, y_t)$ that produces e_{t+1}^* ($\textsc{Expert.ExtendedUpdate}$ is executed because $t \in B$, i.e., a mistake was made). We partition the indices in B into six cases (six disjoint sets), and bound the number of indices that fall in each.

- **Case I.** *The execution of* EXPERT.PREDICT(e_t^*, x_t) *exited via the first return statement in that procedure.* This happens once $|H| \le 2^{2\sqrt{d}}$, and from that point on, the expert and all subsequent experts in the ancestry are exactly simulating the HALVING algorithm (Algorithm 7) in both predictions and updates. Hence, by Fact E.1, B contains at most $m_{\text{I}} = 2\sqrt{d}$ such indices.

- **Case II.** *The execution of* EXPERT.PREDICT(e_t^*, x_t) *exited via the second return statement in that procedure.* In particular $x \preccurlyeq u$, and the predicted label was $\hat{y}_t^* = b \in \{0, 1\}$ such that $x_t \preccurlyeq_b u$. Because e_t^* is assumption-consistent with h, Item 1 in Claim D.5 implies that $u \in \text{path}(h)$. Namely, we see that u is a b-descendant of x_t and $u \in \text{path}(h)$. It follows that $\hat{y}_t^* = b = h(x_t) = y_t$. So no mistakes are made in Case II, and the number of indices $t \in B$ that belong to Case II is simply $m_{\text{II}} = 0$.

In the remaining cases, we assume that EXPERT.PREDICT(e_t^*, x_t) exited via the third return statement in that procedure, so the prediction was

$$\hat{y}_t^* = \mathbb{1}(|S_1| > |S|/3), \tag{26}$$

where $S_1 = \{x' \in S : \ x_t \preccurlyeq_1 x'\}$. These cases are as follows.

- **Case III.** *The execution of* EXPERT.EXTENDEDUPDATE(e_t', x_t, y_t) *exited via the first return statement in that procedure.* Namely, after the update, the resulting expert e_{t+1}^* has $|H| \le 2^{2\sqrt{d}}$. However, because we are not in Case I, at the beginning of the iteration expert e_t^* had $|H| > 2^{2\sqrt{d}}$. Seeing as the cardinality of H decreases monotonically throughout the ancestry $e_1^*, \ldots, e_{n+1}^*$, this type of mistake can happen at most $m_{\text{III}} = 1$ times.

- **Case IV.** *The execution of* EXPERT.EXTENDEDUPDATE(e_t', x_t, y_t) *exited via the second return statement in that procedure.* In this case, $|S_{(1-y_t)}| > |S|/3$, and $e_{t+1}^* = (S', u, H)$ with $S' = S \setminus S_{1-y_t}$. So $|S'| < 2|S|/3$. Namely, the update causes the cardinality of the set S to be multiplied by a factor of at most $2/3$ and it strictly decreases. Seeing as the initial cardinality is t_{max}, and cardinalities are integers, the number of times this can happen is at most

$$m_{\text{IV}} = \frac{\log(t_{\mathsf{max}})}{\log(3/2)} + 1 = \frac{4\sqrt{d}}{\log(3/2)} + 1. \tag{27}$$

In the remaining cases, we assume that the execution of EXPERT.EXTENDEDUPDATE(e_t^*, x_t, y_t) exited via the third return statement in that procedure. This implies that

$$|S_{\hat{y}_t^*}| \le |S|/3 \tag{28}$$

Combining this with Eq. (26), it follows $\hat{y}_t^* = 0$ and therefore $y_t = 1$. The remaining cases are as follows.

- **Case V.** $x_t \in \text{path}(h)$. Let $e_t^* = (S, u, H)$. Seeing as $|H| > 2^{2\sqrt{d}}$ (because we are not in Case I), Claim D.6 (with the assumption $d \ge 16$) implies that $t \le t_{\mathsf{max}}$. By Item 2 of Claim D.5, the facts $x_t \not\preccurlyeq u$ (we are not in Case II) and $x_t \in \text{path}(h)$ imply that $x_t \in S$. In particular, S is not empty.

  Because the $t \to (t+1)$ update of e_{t+1}^* was assumption-consistent with h, Eq. (17) implies that $e_{t+1}^* = (S_\in, u_\in, H_\in)$, with $S_\in = S_0 \cup S_1$. Observe that

  - $|S_0| \le |S|/3$ (plugging $\hat{y}_t^* = 0$ into Eq. (28)); and
  - $|S_1| \le |S|/3$ (because otherwise, by Eq. (26), the prediction would have been $\hat{y}_t^* = 1$).

  Therefore,

$$|S_\in| \le |S_0| + |S_1| \le 2|S|/3. \tag{29}$$

  As in Case IV, combining Eq. (29) and the fact that S is not empty imply an upper bound m_{V} on the number of times Case V can happen, with the bound being the same number $m_{\text{V}} = m_{\text{IV}}$ as in Eq. (27).

- **Case VI.** $x_t \notin \mathrm{path}(h)$. So (x_t, y_t) is a pair such that $x_t \notin \mathrm{path}(h)$ and $y_t = 1$. Assume for contradiction that this type of mistake can happen strictly more than

$$m_{\text{VI}} = 3\sqrt{d}$$

times. Let $t_1, t_2, \ldots, t_{m_{\text{VI}}}$ be the indices of the first m_{VI} iterations of the outer 'for' loop of TRANSDUCTIVELEARNER in which this type of mistake happened. Note that if at the end of iteration $t_{m_{\text{VI}}}$, we had expert $e^*_{t_{m_{\text{VI}}}+1} = (S_{t_{m_{\text{VI}}}+1}, u_{t_{m_{\text{VI}}}+1}, H_{t_{m_{\text{VI}}}+1})$ such that $|H_{t_{m_{\text{VI}}}+1}| \leq 2^{2\sqrt{d}}$, then from that point onwards, the expert would be simulating the halving algorithm, and in particular, it would not make any further mistake of the type in Case VI (all subsequent mistakes would belong to Case I). Hence, by the assumption that strictly more than m_{VI} mistakes were made, it follows that $|H_{t_{m_{\text{VI}}}+1}| > 2^{2\sqrt{d}}$. Let

$$H^* = \big\{h' \in \mathcal{H} : \ (\forall t \in [m_{\text{VI}}] : \ h'(x_{t_t}) = 1 \ \wedge \ x_t \notin \mathrm{path}(h'))\big\}.$$

Because $e^*_{t_{m_{\text{VI}}}+1}$ is assumption-consistent with h, and from the construction of $H_{t_{m_{\text{VI}}}+1}$ using $H_{\in}$ and $H_{\notin}$ in EXPERT.EXTENDEDUPDATE, it follows that $H_{t_{m_{\text{VI}}}+1} \subseteq H^*$. So there exist collections $H^* \subseteq \mathcal{H}$ and $X = \{x_{t_t} : \ t \in [m_{\text{VI}}]\} \subseteq T_d$ such that

- $|H^*| \geq |H_{t_{m_{\text{VI}}}+1}| > 2^{2\sqrt{d}}$,
- $|X| = m_{\text{VI}} = 3\sqrt{d}$,
- $\forall h' \in H^* \ \forall x \in X : \ h'(x) = 1$.
- $\forall h' \in H^* \ \forall x \in X : \ x \notin \mathrm{path}(h')$.

This is a contradiction to the choice of $\mathcal{H}$, specifically, to Item 2 in Lemma D.2.

Thus, combining the analyses of all cases, we see that the number of mistakes made by the $\mathrm{ancestry}\big(e^*_{n+1}\big)$ is at most

$$\begin{aligned} m &\leq m_{\text{I}} + m_{\text{II}} + m_{\text{III}} + m_{\text{IV}} + m_{\text{V}} + m_{\text{VI}} \\ &\leq 2\sqrt{d} + 0 + 1 + \left(\frac{4\sqrt{d}}{\log(3/2)} + 1\right) + \left(\frac{4\sqrt{d}}{\log(3/2)} + 1\right) + 3\sqrt{d} \\ &\leq 24\sqrt{d}. \end{aligned}$$

The weights satisfy

$$w(e^*_{t+1}) \begin{cases} = w(e^*_t) & \hat{y}^*_t = y_t \\ \geq \frac{1}{4} \cdot w(e^*_t) & \hat{y}^*_t \neq y_t. \end{cases}$$

This implies that $w(e^*_{n+1}) \geq w(e^*_1) \cdot \prod_{t=1}^{n} 4^{-\mathbb{1}(\hat{y}_i \neq y_i)} = w(e^*_1) \cdot 4^{-m} \geq 4^{-24\sqrt{d}} = 2^{-48\sqrt{d}}$, as desired. □

D.4.4 Multiplicative Weights Mistake Bound

Claim D.8 (Mistake bound for multiplicative weights)**.** *Let $d, n \in \mathbb{N}$, let $\alpha > 0$, let $\mathcal{H} \subseteq \{0,1\}^{T_d}$, and let $x_1, \ldots, x_n \in T_d$. Consider an execution of*

$$\text{TRANSDUCTIVELEARNER}(\mathcal{H}, (x_1, x_2, \ldots, x_n))$$

as in Algorithm 5. Assume that at the end of the execution, there exists $e^* \in E_{n+1}$ such that

$$w(e^*) \geq 2^{-\alpha}.$$

Then TRANSDUCTIVELEARNER *makes at most α mistakes.*

Proof of Claim D.8. For all $i \in [n+1]$, let $w(E_i) = \sum_{e \in E_i} w(e)$. For each $i \in [n]$, if $\hat{y}_i \neq y_i$, then $w(E_{i+1}) \leq w(E_i)/2$. Hence, if TRANSDUCTIVELEARNER makes m mistakes, then by induction

$$w(E_{n+1}) \leq w(E_1) \cdot \prod_{t=1}^{n} 2^{-\mathbb{1}(\hat{y}_i \neq y_i)} = 2^{-m} \cdot w(E_1).$$

So

$$2^{-\alpha} \leq w(e^*) \leq \sum_{e \in E_{n+1}} w(e) = w(E_{n+1}) \leq 2^{-m} \cdot w(E_1) = 2^{-m}.$$

We conclude that

$$m \leq \alpha,$$

as desired. □

D.5 Proof

Proof of Theorem D.1. Fix an integer $d \geq 43$. Let $\mathcal{H} \subseteq \{0,1\}^{T_{d-1}}$ be the class constructed by invoking Lemma D.2 for the integer $d - 1 \geq 42$. We argue that this class satisfies the requirements of Theorem D.1.

By construction, $\mathcal{H}$ is a class of Littlestone dimension precisely d. By Theorem A.7, this implies the equality in Item 2.

We now show the upper bound in Item 1. We argue that TRANSDUCTIVELEARNER (Algorithm 5) satisfies this upper bound. By Claim D.7, at the end of the execution of TRANSDUCTIVELEARNER there exists an expert $e \in E_{n+1}$ such that $w(e) \geq 2^{-48\sqrt{d}}$. By Claim D.8, this implies that the number of mistakes made by TRANSDUCTIVELEARNER is at most $48\sqrt{d}$, as desired. □

E Halving

Fact E.1. *Let $\mathcal{X}$ be a set, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a hypothesis class. Then for all $n \in \mathbb{N}$, all sequences $x \in \mathcal{X}^n$, and all realizable adversaries,* HALVING *(Algorithm 7) makes at most* $\log(|\mathcal{H}|)$ *mistakes in the transductive online learning (Game 2).*[25] *Namely,*

$$\sup_{n \in \mathbb{N}} \sup_{A \in \mathcal{A}_n} M_{\mathsf{tr}}(\mathcal{H}, n, \text{HALVING}, A) \leq \log(|\mathcal{H}|).$$

[25] With the suitable syntactic modification, it also makes at most $\log(|\mathcal{H}|)$ mistakes in the standard online learning (Game 1).

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

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

2. **Limitations**

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

   Answer: [NA]

   Justification: Purely rigorous mathematical results. We explain precisely what our proofs imply (and therefore also what they do not imply).

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: For each theoretical result, the paper provides the full set of assumptions and a complete (and correct) proof.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [NA]

   Justification: The paper has no experiments.

   Guidelines:

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

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

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

Answer: [NA]

Justification: The paper does not include experiments requiring code.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [NA]

   Justification: The paper does not include experiments.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [No]

    Justification: The work is purely theoretical with no immediate direct societal impacts forseeable.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [NA]

    Justification: The paper does not use existing assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

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

14. **Crowdsourcing and research with human subjects**

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development in this research does not involve LLMS as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.