Simple online learning with consistency oracle

Alexander Kozachinskiy[12], Tomasz Steifer[134]

[1]Instituto Milenio Fundamentos de los Datos, Chile
[2]Centro Nacional de Inteligencia Artificial, Chile
[3]Instituto de Ingeniería Matemática y Computacional, Universidad Católica de Chile
[4]Institute of Fundamental Technological Research, Polish Academy of Sciences

Abstract

We consider online learning in the model where a learning algorithm can access the class only via the *consistency oracle*—an oracle, that, at any moment, can give a function from the class that agrees with all examples seen so far. This model was recently considered by Assos et al. (COLT'23). It is motivated by the fact that standard methods of online learning rely on computing the Littlestone dimension of subclasses, a problem that is computationally intractable.

Assos et al. gave an online learning algorithm in this model that makes at most C^d mistakes on classes of Littlestone dimension d, for some absolute unspecified constant $C > 0$. We give a novel algorithm that makes at most $O(256^d)$ mistakes. Our proof is significantly simpler and uses only very basic properties of the Littlestone dimension. We also observe that there exists no algorithm in this model that makes at most $2^{d+1} - 2$ mistakes.

Our algorithm (as well as the algorithm of Assos et al.) solves an open problem by Hasrati and Ben-David (ALT'23). Namely, it demonstrates that every class of finite Littlestone dimension with recursively enumerable representation admits a computable online learner (that may be undefined on unrealizable samples).

1 Introduction

Online learning is a fundamental setting in machine learning theory and artificial intelligence. Online learning for binary classification can be modeled

by the following game, played between Learner and Adversary. It starts with a hypothesis class H, which is a set of functions from some domain X to $\{0,1\}$. The game goes on for infinitely many rounds. In each round, Adversary first gives Learner an element $x \in X$. Learner then makes a prediction $\widehat{y} \in \{0,1\}$ of the label of x. Finally, Adversary reveals the "real" label $y \in \{0,1\}$ of x. If $y \neq \widehat{y}$, this means that Learner made a mistake. Adversary has to maintain consistency of all its labels with some function from H. Learner's goal is to make as few mistakes as possible.

In his seminal paper [5], Littlestone gave a combinatorial characterization (now called the Littlestone dimension) of hypotheses classes that are learnable by a learner who always makes a uniformly bounded number of mistakes. The minimal possible mistake bound when playing on the class H exactly equals the Littlestone dimension of H. This bound is achieved by a learner, playing according to Littlestone's Standard Optimal Algorithm (SOA). However, SOA requires computing Littlestone dimension of subclasses of H, which in some settings is known to be computationally demanding [2, 7, 6, 4]. This motivates the search for new online learning algorithms that are computationally tractable even if SOA is not, possibly at the cost of increasing the bound on the number of mistakes.

Following a very recent paper by Assos et al. [1], we consider a scenario when Learner has access to the *consistency oracle* for the class H. Namely, at each round of the game, Learner can obtain a function from the class which is consistent with all Adversary's labels revealed so far. We assume that this is the only information about the class that is available to Learner.

We are aware of two frameworks where consistency oracle is efficiently computable while SOA is not. The first one is when the domain is finite and the hypothesis H is given as a truth table. Then consistency oracle can be computed in polynomial time (in the size of the table) while SOA, under standard complexity assumptions, requires quasi-polynomial time [2, 7, 6]. The other framework is concerned with *recursively enumerably representable* (RER) hypothesis classes, studied by Hasrati and Ben-David [4]. These are classes that can be given as recursively enumerable sets of programs that compute functions from the class. As Hasrati and Ben-David observed, there are RER classes of finite Littlestone dimension for which there exists no computable SOA. On the other hand, consistency oracle for a RER class is always computable. Indeed, we can start enumerating programs for functions of the class until we find one that agrees with all labels of Adversary so far. Since Adversary is constrained to be consistent with some function from the class, this procedure always halts.

Results. Assos et al. gave an online learning algorithm in the model with consistency oracle that makes at most C^d mistakes for hypothesis classes of Littlestone dimension d, where $C > 0$ is some absolute constant (unspecified in their paper). Our main contribution is a new, simpler algorithm that achieves the mistake bound of at most $O(256^d)$. Our result follows from an elementary proof and uses only basic facts about the Littlestone dimension. This contrasts with a rather complicated combinatorial proof given by Assos et al., which requires advanced results on threshold dimension and various notions of fat-shattering dimensions. Arguably, our algorithm is itself simpler. For any prediction, it just takes a simple majority vote over some functions, obtained from the consistency oracle on previous steps. At the same time, the algorithm of Assos et al. uses a weighted majority vote, with exponential weights that are updated in each round.

We complement this with a simple lower bound, showing that any online learning algorithm in the model with consistency oracle must make at least $2^{d+1} - 1$ mistakes for classes of Littlestone dimension d. This demonstrates that exponential number of mistakes in this model is inevitable. Our work leaves the following question.

Open Problem 1. *What is the minimal $C > 0$ such that for all d there exists an online learning algorithm in the model with consistency oracle, making at most $O(C^d)$ mistakes for classes of Littlestone dimension d?*

From our results, it follows that $2 \leq C \leq 256$.

Applications. Both our algorithm and the algorithm of Assos et al. are time-efficient – to produce a prediction, they need time which is linear in the number of mistakes so far. Thus, both algorithms imply efficient online learners for the two settings mentioned above. Namely, for the setting when H is given as a truth table, we get a polynomial-time online learning algorithm with at most $O(256^d)$ mistakes. In turn, for every RER class of finite Littlestone dimension, we get the existence of a computable online learner.

2 Preliminaries

Here we give necessary definitions, regarding hypothesis classes and Littlestone dimension. Fix an infinite set X called *the domain*. Non-empty sets $H \subseteq \{0,1\}^X$ will be called *hypothesis classes*. A *labeled tree* is a complete rooted binary tree of depth d, for some $d \in \mathbb{N}$, in which every non-leaf node is labeled by an element of X. We also assume that, for every non-leaf node v, if we consider two edges that go from v to its children, one of them is labeled

by 0 and the other one is labeled by 1. Correspondingly, one child of v will be called the 0-child of v and the other child will be called the 1-child of v. Every leaf l in such a tree can be understood as an assignment of elements of X, appearing on the path to l, to 0s and 1s. The idea is that when we descend from a node, labeled by $x \in X$, to one of the children of this node, namely, to its y-child for some $y \in \{0,1\}$, this can be understood as if we assign y to x. More specifically, any leaf l can be assigned a sequence of pairs $(x_1, y_1), \ldots, (x_d, y_d) \in X \times \{0,1\}$, where $x_i \in X$ is the label of the depth-$(i-1)$ ancestor of l (the depth-0 ancestor of l is the root), and $y_i \in \{0,1\}$ is the label of the edge, leading from the depth-$(i-1)$ ancestor of l to its depth-i ancestor. Next, we say that a leaf l is *consistent* with a function $f\colon X \to \{0,1\}$ if so is the corresponding partial assignment, that is, if

$$f(x_1) = y_1, \ldots, f(x_d) = y_d.$$

Now, we say that a hypothesis class H shatters a labeled tree T if every leaf l is consistent with some function in H.

The Littlestone dimension of a hypothesis class H, denoted by $\mathsf{Ldim}(H)$, is the maximal $d \geq 1$ such that H shatters some labeled tree of depth d. If there is no such $d \geq 1$, that is, if there is not even a depth-1 tree, shattered by H (this happens exactly when H has just one function), we set $\mathsf{Ldim}(H) = 0$. In turn, if for every $d \geq 1$ there exits a labeled tree of depth d, shattered by H, then we set $\mathsf{Ldim}(H) = +\infty$. We leave $\mathsf{Ldim}(\varnothing)$ undefined.

It is important to note again that by labeled trees we only mean complete trees, that is, if H shatters some incomplete tree of depth d (some leaves are at depth d but some are at smaller depth), this does not count.

We mention two standard properties of the Littlestone dimension that easily follow from the definition.

Proposition 1. *For any hypothesis class H we have $\mathsf{Ldim}(H) \leq \log_2(|H|)$.*

Proposition 2. *Consider any hypothesis class H and any $x \in X$. Define $H_0 = \{f \in H \mid f(x) = 0\}$ and $H_1 = \{f \in H \mid f(x) = 1\}$. Assume that both H_0 and H_1 are non-empty. Then*

$$\mathsf{Ldim}(H) \geq \min\{\mathsf{Ldim}(H_0), \mathsf{Ldim}(H_1)\} + 1.$$

3 The model and the lower bound

We now proceed to a formal definition of the model of online learning with consistency oracle for classes of Littlestone dimension d. It is the following perfect information game, played between Learner and Adversary. The game

proceeds in infinitely many rounds, indexed by $r = 1, 2, 3, \ldots$ In the round number r, the following happens:

- Adversary names $x_r \in \mathbb{N}$;
- Learner names $\widehat{y_r} \in \{0, 1\}$.
- Adversary names a bit $y_r \in \{0, 1\}$ and a function $f_r \colon X \to \{0, 1\}$ such that, first,
$$f_r(x_1) = y_1, \ldots, f_r(x_r) = y_r$$
(the function f_r must be consistent with all assignments of Adversary we have so far), and second,
$$\mathsf{Ldim}(\{f_1, \ldots, f_r\}) \leq d$$
(Adversary makes sure that its functions come from some class of Littlestone dimension at most d).

Remark 1. *Note that both players always have at least one legal move from any reachable position. Adversary, for example, can just use the same function as in the previous round.*

The *number of mistakes* along an infinite play is the number of rounds r such that $\widehat{y_r} \neq y_r$ (in some plays, of course, it might be infinite).

An online learning algorithm with consistency oracle that makes at most n mistakes for classes of Littlestone dimension d is a strategy of Learner in the above game, ensuring that the number of mistakes along any play is at most n.

Remark 2. *Fix a bound on the number of mistakes n. We obtain a game with the following winning condition: Learner wins if the number of mistakes along the play does not exceed n. Since the game is infinite, it requires some argument to show that the game is* ***determined****, meaning that for every d and n, either Learner or Adversary has a winning strategy. This is because the winning condition for Adversary is open, that is, it can be represented as a union of all infinite extensions of some set of finite plays (those where Learner makes more than n mistakes). It is well-known that all infinite games with open winning conditions are determined [3].*

We observe that there is a simple strategy of Adversary, enforcing Learner to make at least $2^{d+1} - 1$ mistakes.

Proposition 3. *For every $d \geq 1$, there exists no online learning algorithm for classes of Littlestone dimension d in the model with consistency oracle that makes at most $2^{d+1} - 2$ mistakes.*

Proof. Adversary picks any $n = 2^{d+1} - 1$ elements of the domain and gives them, in any order, to Learner. Each time Learner predicts some $\widehat{y_r}$, Adversary plays the opposite label $y_r = \neg\widehat{y_r}$, and an *arbitrary function* f_r, agreeing with the current history. As a result, Learner makes a mistake in each round. After n rounds, we have $n = 2^{d+1} - 1$ mistakes, but the set of Adversary's function still has Littlestone dimension at most d. This is simply because there are at most $2^{d+1} - 1$ different functions so far, so the Littlestone dimension, by Proposition 1, is bounded by $\log_2(2^{d+1} - 1) < d + 1$. □

Our algorithm will use oracle access to functions, provided by Adversary, rather than keeping the whole description of these functions (that might be infinite).

Theorem 1. *There exists an online learning algorithm in the model with consistency oracle that for classes of Littlestone dimension d makes at most $O(256^d)$ mistakes.*

Producing each prediction takes linear time in the number of mistakes made so far, assuming that it takes constant time to evaluate a function, coming from the consistency oracle, on a given $x \in X$.

We present our algorithm in the next section.

4 The algorithm

For our algorithm, it is not necessary to run the consistency oracle every round and on all examples, revealed by Adversary so far. Instead, it will be enough to run it only after rounds with mistakes, and only for examples from these rounds. More precisely, the algorithm will maintain a sample $S = (x_1, y_1), \ldots, (x_e, y_e)$, where e is the number of mistakes so far. Whenever we make a mistake, we add a new pair to the end of S and we run the consistency oracle on the updated S. As a result, we also keep an ordered list of functions $f_1, \ldots, f_e$, given to us by the consistency oracle after rounds with mistakes. To produce a prediction on a given $x \in X$, we just need to evaluate $f_1(x), \ldots, f_e(x)$, and then the prediction takes $O(e)$-time.

For the sake of clarity, we first present a recursive version of the algorithm, which allows us to give a straightforward inductive proof of the claimed mistake bound. We then give a direct non-recursive implementation of the algorithm.

The goal of Learner is to guarantee that $\mathsf{Ldim}\{f_1, \ldots, f_e\}$ grows as $\log_{256}(e) - \Omega(1)$. Then for classes of Littlestone dimension d, the number of mistakes will be bounded by $O(256^d)$.

For effective implementation of our algorithm, we will mark some functions among $f_1, \ldots, f_e$ as *active*. Active functions will be kept in the ordered list. The size of the list will be denoted by L, and g_i will denote the $(i+1)$st function in the current list (the first function in the list will be g_0). The algorithm will employ two operations with the list of active functions: (a) add a new function to the end of the list; (b) delete some functions from the list, without changing the order of the remaining ones. Both of them are easily realizable in linear time using the list data structure. Note that after deletion, g_i might refer to a different function.

We will add a new active function only in the case when we used the last active function for prediction (that is, given $x \in X$, we predicted $g_{L-1}(x)$) and we made a mistake. In this way, we ensure that the list of active functions never has repetitions. Indeed, consider a moment we put some function g on the list. We show that we can never put the same function again. Indeed, currently, g is the last one in the list. Whatever deletions we do, g stays at the end of the list. Now, a new function comes when we made a prediction $\widehat{y} = g(x)$ on some x, and it was a mistake. At this moment, we add $(x, 1-\widehat{y})$ to S. This means that all future functions (including one that is about to be added) will be equal to $1-\widehat{y}$ on x as they come from the consistency oracle called on the updated S.

At each round of the game, the algorithm predicts according to the majority vote over the last 2^k active functions, for some $k \geq 0$. If there is no mistake, the algorithm keeps doing the same thing, without changing k or the list of active functions. If a mistake happens, the algorithm modifies the list of active functions (and, potentially, k, according to the rules to be defined later). If $k = 0$ (when just the last active function is used for prediction), the algorithm runs the consistency oracle on S, updated after the new mistake, and adds the resulting function to the end of the list of active functions. If $k > 0$, the algorithm does not add a new active function, and this ensures, as we discuss below, that the list of active functions never has repetitions. Instead, when $k > 0$, deletes some active functions: among 2^k active functions it used for voting, it selects any 2^{k-1} of them that agree with its prediction (and, thus, disagree with the label that Adversary gave to us after our prediction) and removes other 2^{k-1} from the list of active functions. This is formally defined as a subroutine `VoteAndUpdate`(k) below.

Technically, if there are not enough active functions to do the requested majority vote, our algorithm just predicts $\widehat{y} = 0$. We will make sure that this can only happen for $k = 0$. Let us also remark that we exit `VoteAndUpdate`(k) exactly after 1 mistake since the start of the subroutine (and if we never make mistakes inside it, we never exit it).

We now describe the recursive implementation of our algorithm. The

Algorithm 1: `VoteAndUpdate`(k)

receive $x \in X$ to predict;

if $L \geq 2^k$ **then**
  predict $\widehat{y} = \mathsf{MAJORITY}(g_{L-2^k}(x), \ldots, g_{L-1}(x))$;
else
  predict $\widehat{y} = 0$;
end if

receive $y \in \{0, 1\}$;

if $y = \widehat{y}$ **then**
  go to 1;
else
  add (x, y) to S;
  if $k = 0$ *or* $L < 2^k$ **then**
    $L := L + 1$;
    $g_L := \texttt{ConsOracle}(S)$;
  else
    among $g_{L-2^k}, \ldots, g_{L-1}$, select any 2^{k-1}, agreeing with $(x, \widehat{y})$, and delete the other 2^{k-1};
  end if
end if

idea is to construct a procedure that halts whenever $O(256^d)$ mistakes are made, and if it halts, the set of active functions is guaranteed to have the Littlestone dimension larger than d. Such a procedure cannot halt over a class of Littlestone dimension d (all active functions come from the class) and hence it makes less than $O(256^d)$ mistakes.

Ideally, to facilitate an inductive proof, the algorithm, given parameter d, would make a constant number of recursive calls to its instance with parameter $d-1$ and so on. However, to make the induction work, we need a stronger property than just "having Littlestone dimension $> d$". This property is given in the next definition.

Definition 1. *Take any real $\gamma \geq 1$. A non-empty set $T \subseteq \{0,1\}^X$ is γ-**advanced** if for every non-empty $A \subseteq T$ we have*

$$\mathsf{Ldim}(A) \geq \gamma + \log_{16}\left(\frac{|A|}{|T|}\right).$$

By definition, any γ-advanced set has Littlestone dimension at least γ (use the definition for $A = T$). Correspondingly, our plan is to write a subroutine `CreateAdv`(k) that, if it halts, "creates" a $(1+k/2)$-advanced set of active functions. For classes of Littlestone dimension d, it will be enough to run `CreateAdv`$(2d-1)$.

Algorithm 2: `CreateAdv`(k)

```
if k = 0 then
    for i := 1 to 16 do
        VoteAndUpdate(0)
    end for
else
    for i := 1 to 16 do
        CreateAdv(k − 1);
        VoteAndUpdate(3k + 1)
    end for
end if
```

The following proposition summarizes the properties of `CreateAdv`(k).

Proposition 4. *For every $k \geq 0$, procedure* `CreateAdv`(k) *halts exactly after $R_k = 16+16^2+\ldots+16^{k+1}$ mistakes since the start of the procedure. Moreover, if it halts, it attaches to the list of active function exactly $2 \cdot 8^{k+1}$ new active functions, forming a $(1+k/2)$-advanced set (and it does not delete active functions that were in the list before the start of the procedure).*

By this proposition, $\texttt{CreateAdv}(2d-1)$ halts after precisely $16+16^2+16^{2d} = O(256^d)$ mistakes. On the other hand, it cannot halt for classes of Littlestone dimension d. Thus, running $\texttt{CreateAdv}(2d-1)$, we get an online learning algorithm with at most $O(256^d)$ mistakes for classes of Littlestone dimension d.

Proof of Proposition 4. We prove this proposition by induction on k. We start with the mistake bound. For $k = 0$, the procedure $\texttt{CreateAdv}(k)$ executes $\texttt{VoteAndUpdate}(0)$ 16 times. Each $\texttt{VoteAndUpdate}(0)$ halts exactly after 1 mistake, so if all of them halt, we had exactly $R_0 = 16$ mistakes. We now establish the inductive step. Assume that the mistake bound is proved for $\texttt{CreateAdv}(k-1)$. Now, $\texttt{CreateAdv}(k)$ runs 16 times $\texttt{CreateAdv}(k-1)$ and 16 times $\texttt{VoteAndUpdate}(3k+1)$. Thus, overall, it halts after $16(R_{k-1}+1) = 16(1+16+\ldots 16^k) = R_k$ mistakes, as required.

We now establish the second part of the proposition. We start with the induction base. We have to show that $\texttt{CreateAdv}(0)$, if it halts, attaches 16 new active functions, forming a 1-advanced set. Note that $\texttt{VoteAndUpdate}(0)$ always adds a new active function without touching the previous ones. Thus, $\texttt{CreateAdv}(0)$ halts by adding 16 new active functions. As we remarked in the beginning of the description of our algorithm, the list of active functions never has repetitions. Therefore, it remains to establish that any set of 16 different functions is 1-advanced.

Lemma 1. *Any $T \subseteq \{0,1\}^X$ of size 16 is 1-advanced.*

Proof. Take any non-empty $A \subseteq T$. If $|A| = 1$, then $\mathsf{Ldim}(A) = 0 = 1 + \log_{16}(|A|/|T|)$, as required. If $|A| \geq 2$, then $\mathsf{Ldim}(A) \geq 1$, meaning that $\mathsf{Ldim}(A) = 1 \geq 1 + \log_{16}(|A|/|T|)$, as required. □

We are now proving the induction step. Assuming that the statement is proved for $k-1$, we establish it for k. Consider any run of $\texttt{CreateAdv}(k)$ that halts. At the beginning, it runs $\texttt{CreateAdv}(k-1)$. By the induction hypothesis, if it halts, it attaches a $(1+(k-1)/2)$-advanced set $T_1 \subseteq \{0,1\}^X$ of size $2\cdot 8^k$ to the list of active functions. Then we run $\texttt{VoteAndUpdate}(3k+1)$. At this moment, for prediction we use the majority vote over the last $2^{3k+1} = 2\cdot 8^k$ active functions, that is, over T_1. We exit $\texttt{VoteAndUpdate}(3k+1)$ when our majority vote made a mistake on some $x = x_1$. More specifically, the majority of functions from T_1 were equal to some $\widehat{y}_1 \in \{0,1\}$ on x_1, and then we obtained the opposite label $y_1 = 1-\widehat{y}_1$ from Adversary. We choose some subset $\Gamma_1 \subseteq T_1$ of size $(1/2)|T_1| = 8^k$ where all functions are equal to $\widehat{y}_1$ on x_1, and we delete the rest of functions of T_1 from the list of active function. At this point, the pair $(x_1, y_1) = (x_1, 1-\widehat{y}_1)$ is added to the sample on which

we call the consistency oracle, meaning that all new active functions will be equal to $1 - \widehat{y}_1$ on x_1, opposite to functions from Γ_1.

We then repeat $\texttt{CreateAdv}(k-1)$ and $\texttt{VoteAndUpdate}(3k+1)$ 15 more times. Each time, a set Γ_i with the same properties as Γ_1 is attached to the list of active functions. More precisely, $\texttt{CreateAdv}(k)$ ends up attaching $\Gamma_1, \ldots, \Gamma_{16}$, where for every $i = 1, \ldots, 16$ we have the following:

- Γ_i is a set of functions of size 8^k, which is a subset of some $(1+(k-1)/2)$-advanced set T_i of size $2 \cdot 8^k$;
- there exists $x_i \in X$ and $\widehat{y}_i \in \{0, 1\}$ such that, first, all functions from Γ_i are equal to $\widehat{y}_i$ on x_i, and second, all functions from $\Gamma_{i+1} \cup \ldots \cup \Gamma_{16}$ are equal to $1 - \widehat{y}_i$ on x_i.

Overall, we attached $16 \cdot 8^k = 2 \cdot 8^{k+1}$ new active functions. It remains to show that $T = \Gamma_1 \cup \ldots \cup \Gamma_{16}$ is $(1 + k/2)$-advanced.

Take any non-empty subset $A \subseteq T$. Our goal is to show that $\mathsf{Ldim}(A) \geq 1 + k/2 + \log_{16}\left(\frac{|A|}{|T|}\right)$. Once again, the list of active functions never has repetitions, which means that $\Gamma_1, \ldots, \Gamma_{16}$ are disjoint. In particular, A is a disjoint union of $\Gamma_1 \cap A, \ldots, \Gamma_{16} \cap A$. Denote

$$\alpha = \frac{|A|}{|T|}, \qquad \alpha_i = \frac{|A \cap \Gamma_i|}{|\Gamma_i|}.$$

Note that our goal is to show that $\mathsf{Ldim}(A) \geq 1 + k/2 + \log_{16}(\alpha)$. Since A is non-empty, we have $\alpha > 0$. Each Γ_i is 16 times smaller than T, which means that

$$\alpha = \frac{|\Gamma_1 \cap A| + \ldots + |\Gamma_{16} \cap A|}{|T|} = \frac{\alpha_1 + \ldots + \alpha_{16}}{16}. \tag{1}$$

First, consider the case when there exists $i = 1, \ldots, 16$ such that $\alpha_i \geq 8\alpha$. We will bound $\mathsf{Ldim}(A)$ from below by just $\mathsf{Ldim}(A \cap \Gamma_i)$, using the fact that $A \cap \Gamma_i \subseteq \Gamma_i \subseteq T_i$ is a sufficiently large subset of a $(1 + (k-1)/2)$-advanced set T_i. Moreover, the improvement of the parameter will be achieved due to the fact that $A \cap \Gamma_i$ is 4 times larger w.r.t. T_i than A w.r.t. T.

First of all, $A \cap \Gamma_i$ is non-empty because its size is the α_i-fraction of the size of Γ_i, and $\alpha_i \geq 8\alpha > 0$. Again, the size of $A \cap \Gamma_i$ is the α_i-fraction of $|\Gamma_i|$ while the latter is half of $|T_i|$. Hence, the size of $A \cap \Gamma_i$ is the $\alpha_i/2$-fraction of $|T_i|$. Applying the definition for T_i, we get:

$$\begin{aligned}
\mathsf{Ldim}(A) &\geq \mathsf{Ldim}(A \cap \Gamma_i) \\
&\geq (1 + (k-1)/2) + \log_{16}(\alpha_i/2) \\
&\geq (1 + (k-1)/2) + \log_{16}(4\alpha) \\
&= 1 + k/2 + \log_{16}(\alpha),
\end{aligned}$$

as required.

Now, consider the case when $\alpha_i < 8\alpha$ for every $i = 1, \ldots, 16$. This time, we will bound $\mathsf{Ldim}(A)$ from below using Proposition 2, showing that as long as for some $x \in X$, both subsets $A_0 = \{f \in A \mid f(x) = 0\}$ and $A_1 = \{f \in A \mid f(x) = 1\}$ have Littlestone dimension at least d, the set A itself has Littlestone dimension at least $d + 1$. A crucial observation here is that for any $i < j$, we have that all functions from $A \cap \Gamma_i$ are equal to $\widehat{y}_i$ on x_i while all functions from $A \cap \Gamma_j$ are equal to $1 - \widehat{y}_i$ on x_i. By Proposition 2, for every $i < j$, as long as both $A \cap \Gamma_i, A \cap \Gamma_j$ are non-empty, we get:

$$\mathsf{Ldim}(A) \geq \min\{\mathsf{Ldim}(A \cap \Gamma_i), \mathsf{Ldim}(A \cap \Gamma_j)\} + 1.$$

Observe that we get the required bound $\mathsf{Ldim}(A) \geq 1 + k/2 + \log_{16}(\alpha)$ as long as we have:

$$\mathsf{Ldim}(A \cap \Gamma_i) \geq k/2 + \log_{16}(\alpha)$$

for at least two different i.

Once again, $\mathsf{Ldim}(A \cap \Gamma_i)$ can be bounded from below using the fact that $A \cap \Gamma_i$ is a subset of T_i whose size is the $(\alpha_i/2)$-fraction of $|T_i|$. For every i, as long as $\alpha_i > 0$, we get

$$\begin{aligned}\mathsf{Ldim}(A \cap \Gamma_i) &\geq (1 + (k-1)/2) + \log_{16}(\alpha_i/2) \\ &= k/2 + \log_{16}(2\alpha_i).\end{aligned}$$

Thus, our job is done as long as $\alpha_i \geq \alpha/2$ for at least 2 different i. To show this, among $\alpha_1, \ldots, \alpha_{16}$, take two largest numbers. Namely, let it be $\alpha_i \geq \alpha_j$ for some $i \neq j$. We claim that $\alpha_i \geq \alpha_j \geq \alpha/2$. Indeed, note that the sum $\alpha_1 + \ldots + \alpha_{16}$ can be bounded from above by $16\alpha_j + \alpha_i$ (all numbers, except α_i, are at most α_j). Thus, from (1), we get:

$$\alpha_j + \frac{\alpha_i}{16} \geq \alpha.$$

Remembering that $\alpha_i < 8\alpha$, we obtain $\alpha_j \geq \alpha/2$. □

Direct implementation

Finally, we give a simple non-recursive implementation of our algorithm. Let us first observe that it is possible to give an implementation that does not depend on d. Indeed, for every $k \geq 0$, if we get rid of the recursion, the procedure `CreateAdv`(k) makes calls the VoteAndUpdate procedures in some fixed order. Moreover, `CreateAdv`(k) starts with a call to `CreateAdv`$(k-1)$, which in turn starts with a call to `CreateAdv`$(k-2)$, and so on. This means that

there exists an infinite sequence of the VoteAndUpdate procedures such that, for every $k \geq 0$, some prefix of this sequence is a realization of $\texttt{CreateAdv}(k)$.

It remains to explicitly define this sequence. For a moment,

$$\texttt{Procedure}(0) = \texttt{VoteAndUpdate}(0),$$
$$\texttt{Procedure}(k) = \texttt{VoteAndUpdate}(3k+1), \text{ for } k \geq 1.$$

Note that $\texttt{CreateAdv}(0)$ consists of 16 repetitions of $\texttt{Procedure}(0)$, while $\texttt{CreateAdv}(k)$ for $k > 0$ consists of 16 repetitions of $\texttt{CreateAdv}(k-1)$; $\texttt{Procedure}(k)$.

For $k > 0$, we call $\texttt{Procedure}(k)$ always just after executing $\texttt{CreateAdv}(k-1)$. Now, each call to $\texttt{CreateAdv}(k-1)$ has exactly 16 calls to $\texttt{Procedure}(k-1)$, with the last one of these 16 calls happening right at the end of $\texttt{CreateAdv}(k-1)$. In other words, for $k > 0$, the call to $\texttt{Procedure}(k)$ happens exactly at moments when the last called procedure is $\texttt{Procedure}(k-1)$, and the total number of calls to $\texttt{Procedure}(k-1)$ is a multiple of 16.

This gives the following rule. If the last procedure so far is $\texttt{Procedure}(k)$ for some $k \geq 0$, then if the total number of calls to $\texttt{Procedure}(k)$ is a multiple of 16, we run $\texttt{Procedure}(k+1)$, otherwise, we run $\texttt{Procedure}(0)$.

This order is modeled by the following algorithm.

Algorithm 3: `Learner`

```
1 N := 1;
2 i := maximal i ≥ 0 such that 16^i divides N;
3 run Procedure(0) = VoteAndUpdate(0);
4 for j := 1 to i do
5 |   run Procedure(j) = VoteAndUpdate(3j + 1);
6 end for
7 N := N + 1;
8 go to 2;
```

In this algorithm, we run `Procedure`(0) for all N, `Procedure`(1) for all N that are multiples of 16, `Procedure`(2) for all N that are multiples of 16^2, and so on. In this way, every call to `Procedure`(k) is preceded by (and follows right after) exactly 16 calls to `Procedure`($k-1$), as required.

References

[1] Assos, A., Attias, I., Dagan, Y., Daskalakis, C., and Fishelson, M. K. Online learning and solving infinite games with an erm oracle. In *The Thirty Sixth Annual Conference on Learning Theory* (2023), PMLR, pp. 274–324.

[2] Frances, M., and Litman, A. Optimal mistake bound learning is hard. *Information and Computation 144*, 1 (1998), 66–82.

[3] Gale, D., and Stewart, F. M. Infinite games with perfect information. *Contributions to the Theory of Games 2*, 245-266 (1953), 2–16.

[4] Hasrati, N., and Ben-David, S. On computable online learning. In *International Conference on Algorithmic Learning Theory* (2023), PMLR, pp. 707–725.

[5] Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning 2* (1988), 285–318.

[6] Manurangsi, P. Improved inapproximability of vc dimension and littlestone's dimension via (unbalanced) biclique. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)* (2023), Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[7] Manurangsi, P., and Rubinstein, A. Inapproximability of vc dimension and littlestone's dimension. In *Conference on Learning Theory* (2017), PMLR, pp. 1432–1460.