Improved Replicable Boosting with Majority-of-Majorities

Kasper Green Larsen Markus Engelund Mathiasen Clement Svendsen

Aarhus University

Abstract

We introduce a new replicable boosting algorithm which significantly improves the sample complexity compared to previous algorithms. The algorithm works by doing two layers of majority voting, using an improved version of the replicable boosting algorithm introduced by Impagliazzo et al. [2022] in the bottom layer.

1 Introduction

Replicability of an algorithm is a property introduced as a reaction to what is called the reproducibility crisis. Multiple Nature articles have pointed out the issue of researchers not being able to replicate findings [Baker, 2016, Ball, 2023]. As a supplement to implementing better research practices in order to ensure replicability, Impagliazzo et al. [2022] introduced the concept of replicability as a property of algorithms themselves. Informally, an algorithm is replicable if it, with high probability, outputs the same result when run with *different* input data drawn from the same distribution.

Definition 1.1 (Replicability [Impagliazzo et al., 2022])**.** Let $\mathcal{A}$ be a randomized algorithm. Then, $\mathcal{A}$ is said to be ρ-replicable if there is an $n \in \mathbb{N}$ such that for all distributions $\mathcal{D}$ on some space $\mathcal{X}$, it holds that

$$\mathbb{P}_{S_1,S_2,r}\left[\mathcal{A}(S_1;r) = \mathcal{A}(S_2;r)\right] \geq 1-\rho$$

where $S_1, S_2 \sim \mathcal{D}^n$ are independent, and r denotes the internal randomness used by $\mathcal{A}$.

As is evident from the definition, we require the algorithm to use the same internal randomness r in both runs. This turns out to be crucial - if we remove this requirement, we cannot solve simple tasks such as estimating the mean of a distribution replicably [Dixon et al., 2024]. Researchers who use replicable algorithms may then publish the random seed used in their run of the algorithm which lets other researchers use the same seed to replicate the results with high probability, assuming that the data they use comes from the same underlying distribution.

In this work, we consider replicability in the weak-to-strong learning setting. Specifically we improve the best known sample complexity of ρ-replicable boosting algorithms. Let $\mathcal{X}$ be an input domain, and let $f : \mathcal{X} \to \{-1, 1\}$ be the function we are trying to predict. An algorithm $\mathcal{W}$ is said to be a γ-weak learner for $\gamma \in (0, 1/2)$ if there exists an $m \in \mathbb{N}$ such that for any distribution $\mathcal{D}$ on $\mathcal{X}$ and any sequence of m labelled samples $S = \{(x_i, f(x_i))\}_{i=1}^m$ drawn i.i.d. from $\mathcal{D}$, it holds that $h \coloneqq \mathcal{W}(S) : \mathcal{X} \to \{-1, 1\}$ satisfies $\mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)] \leq 1/2 - \gamma$. We call γ the *advantage* of $\mathcal{W}$ and m the *sample complexity* of $\mathcal{W}$. A strong learner on the other hand, is a learning algorithm such that for any distribution $\mathcal{D}$ on $\mathcal{X}$, failure probability $\delta > 0$ and error $\varepsilon > 0$, there is an $m = m(\varepsilon, \delta) \in \mathbb{N}$ such that when applied to an i.i.d. sample $S \sim \mathcal{D}^m$, the algorithm outputs a classifier which has error at most ε over $\mathcal{D}$ with probability at least $1 - \delta$. We denote the error of a hypothesis $h : \mathcal{X} \to \{-1, 1\}$ with respect to a distribution $\mathcal{D}$ by $\mathrm{Er}_{\mathcal{D}}(h) = \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)]$.

Boosting algorithms were originally introduced to answer the following theoretical question posed by Kearns [1988], Kearns and Valiant [1994]: Is it possible to combine hypotheses produced by a weak learning algorithm into a strong learner? As shown by Schapire [1990], this turned out to be the case, and one of the most famous algorithms that solves this problem is the AdaBoost algorithm [Freund and Schapire, 1995]. In short, boosting works by running a number of iterations. In each iteration t, we update a distribution $\mathcal{D}_t$ on the samples and run the weak learner with this new distribution. After a sufficient number of iterations, we take a weighed majority vote among all the produced weak hypotheses.

1.1 Our contribution

Our main contribution is a replicable boosting algorithm called rMetaBoost which is inspired by an existing replicable boosting algorithm, rBoost [Impagliazzo et al., 2022] and the SmoothBoost algorithm [Servedio, 2001]. First, fix a distribution $\mathcal{D}$ on $\mathcal{X}$ and let $\mathcal{W}$ denote a replicable weak learner and for $\rho \in (0,1)$ let $m_{\mathcal{W}(\rho)}$ be the sample complexity of $\mathcal{W}$ when run with replicability parameter ρ. Our main result is the following:

Theorem 1.2 (rMetaBoost)**.** *For any $\rho, \varepsilon \in (0,1)$ and $\widetilde{\Theta}(\rho\gamma^2)$-replicable weak learner $\mathcal{W}$ with advantage γ, rMetaBoost is ρ-replicable, makes $O(\frac{\ln(1/\varepsilon)}{\gamma^2})$ calls to $\mathcal{W}$, and with probability at least $1-\rho$ outputs a hypothesis H with $\mathrm{Er}_{\mathcal{D}}(H) \leq \varepsilon$. Furthermore, its sample complexity is*

$$\widetilde{O}\left(\frac{m_{\mathcal{W}(\widetilde{\Theta}(\rho\gamma^2))}}{\varepsilon\gamma^2} + \frac{1}{\rho^2\varepsilon\gamma^3}\right).$$

Our algorithm significantly improves on the sample complexity of Impagliazzo et al. [2022] which is $\widetilde{O}\left(\frac{m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{1}{\rho^2\varepsilon^5\gamma^6}\right)$. Note that this sample complexity is not what is stated in their paper, but is in fact the correct sample complexity of their algorithm.[1] We improve the first term by a factor $1/\varepsilon$ and also remove a factor ε in the replicability parameter to the weak learner. Since the sample complexity of most replicable algorithms has a quadratic dependence on their replicability parameter, this will amount to an extra $1/\varepsilon^2$ improvement in this term. In the second term we shave off a factor $1/(\varepsilon^4\gamma^3)$. All improvements are up to logarithmic factors.

As a secondary contribution, we introduce an algorithm rThreshold for performing a replicable *threshold check*. This algorithm replicably checks if the expected value of a function φ is above a certain threshold z, and is used as a subroutine in rMetaBoost. We state the guarantees of the algorithm below.

Lemma 1.3 (rThreshold)**.** *Let $z, \rho \in (0,1)$, $\delta \in (0, \rho/8]$, let $\varphi : \mathcal{X} \to [0,1]$ and let $S = (x_1, \ldots, x_m)$ be samples drawn i.i.d. from distribution $\mathcal{D}$. Then there exists a constant c such that if $m \geq c\frac{\ln(1/\delta)}{\rho^2 z}$, rThreshold$(S, z, \varphi)$ is ρ-replicable and returns a bit b such that with probability at least $1-\delta$:*

- *If $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \leq z/2$, then $b = 0$.*
- *If $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \geq 2z$, then $b = 1$.*

We believe this algorithm is also of independent interest and can be applied in many scenarios as an alternative to statistical queries which were previously used for such applications. This is because our algorithm achieves a dependence of $1/z$ in the sample complexity, while using statistical queries for the same purpose comes with a factor $1/z^2$ in the sample complexity (Thm. 2.3, Impagliazzo et al. [2022]). While our approach to threshold checks is not neccesarily novel, it seems to have been overlooked in the context of replicable algorithms.

1.2 Related work

In recent years, replicable algorithms have been developed in a variety of settings. This includes e.g. learning half spaces, clustering, reinforcement learning and online learning [Kalavasis et al., 2024, Esfandiari et al., 2024, Eaton et al., 2024, Ahmadi et al., 2024].

There are also important connections to the field of differential privacy. Intuitively, a replicable algorithm does not depend heavily on the specific sample given to the algorithm. This is similar to the requirement in differential privacy where we demand that when the algorithm is run on two samples differing in only a single point, then the two distributions on the outputs are close in the sense of max divergence. Bun et al. [2023, Thm. 3.1] show that there is a reduction "without substantial blowup in runtime or sample complexity" from differential privacy to replicability. On the other hand, they also show that no computationally efficient transformation of differentially private algorithms to replicable ones can exist under standard cryptographic assumptions. However, if one does not care about computational efficiency, they do give a reduction from differential privacy to replicability with only a quadratic blowup in sample complexity. This means it would be possible to take an existing differentially private boosting algorithm and make it

[1] We have personally contacted the authors to make them aware, and they have acknowledged this error.

replicable. One example of a differentially private boosting algorithm is BoostingForPeople [Dwork et al., 2010]. However, using the reduction on this algorithm would incur a $1/\gamma^8$ and $1/\varepsilon^2$ dependence in the sample complexity.

Moving away from differential privacy, another candidate algorithm to be made replicable is the SmoothBoost algorithm [Servedio, 2001]. This algorithm differs from e.g. the well-known AdaBoost [Freund and Schapire, 1995] in that it maintains a *smoothness* across the distributions $\mathcal{D}_t$ over the data in every iteration t. Formally, this means that the distribution $\mathcal{D}_t$ satisfies $\max_x \mathcal{D}_t(x) \leq 1/(\varepsilon m)$ for some $\varepsilon > 0$ where m is the number of samples. This smoothness property ensures that no single example has too much influence on the distributions which is why smoothness is a desirable property when designing replicable boosting algorithms. In fact, the boosting algorithm by Impagliazzo et al. [2022] can be seen as a translation of SmoothBoost into the replicable setting.

The downside of using SmoothBoost is that it requires $O(\frac{1}{\varepsilon\gamma^2})$ invocations of the weak learner $\mathcal{W}$. We call this the *round complexity* of the algorithm. This should be compared to AdaBoost which has round complexity $O(\frac{\ln(1/\varepsilon)}{\gamma^2})$. In the replicable setting, we draw new samples for each invocation of $\mathcal{W}$, so the round complexity directly affects the number of samples used. This motivates looking at smooth boosting algorithms with fewer invocations of $\mathcal{W}$ such as the one presented by Barak et al. [2009]. This algorithm uses Bregman projections to maintain the smoothness property, and it matches the round complexity of AdaBoost. However, converting the algorithm to the replicable setting would require us to make replicable approximations of these Bregman projections which turns out to use more samples than we obtain in Theorem 1.2.

1.3 High-Level Ideas

We will now explain the very high-level idea behind our new boosting algorithm rMetaBoost. The first step towards constructing this improved replicable boosting algorithm is to make slight modifications to the algorithm rBoost of Impagliazzo et al. [2022] to improve its sample complexity. We will refer to this modified version as rBoost* which can be found in Algorithm 1. Remark that the functions g_t, μ_t are functions over the entire domain $\mathcal{X}$ and not just the samples that we see. This means that we cannot afford to update these functions explicitly for every point, so instead we update the description of the functions. To distinguish this from normal assignments in the pseudocode, we use the $\stackrel{\text{def}}{=}$ operator for assignments to these functions and the $\leftarrow$ operator for normal assignments.

In this algorithm $\mu_t : \mathcal{X} \to [0,1]$ is a function which determines the reweighing of the data distribution $\mathcal{D}$ in iteration t. The reweighed distribution is then $\mathcal{D}_{\mu_t}(x) = \mu_t(x)\mathcal{D}(x)/d(\mu_t)$ where $d(\mu_t) = \mathbb{E}_{x\sim\mathcal{D}}[\mu_t(x)]$ is the normalization factor which we call the *density* of μ. The subroutine RejectionSampler then lets us sample from the distribution $\mathcal{D}_{\mu_t}$ when given access to μ_t and samples from $\mathcal{D}$ (see Lemma 2.1 for formal guarantee). We also note without proof that large density of μ_t actually implies smoothness of the reweighed distribution $\mathcal{D}_{\mu_t}$ with respect to the original distribution $\mathcal{D}$. More precisely, if $d(\mu_t) \geq \varepsilon$, for some $\varepsilon > 0$, then $\mathcal{D}_{\mu_t}(x) \leq \mathcal{D}(x)/\varepsilon$ for all $x \in \mathcal{X}$. These samples from $\mathcal{D}_{\mu_t}$ are then given to the replicable weak learner. We will not go into further detail with how or why the original rBoost works but instead refer to Impagliazzo et al. [2022], Servedio [2001].

In total, we have made two modifications in rBoost*. The first modification is that we have changed the termination condition in line 14 to use our rThreshold algorithm instead of the statistical query algorithm they used. This accomplishes exactly the same thing, but uses a factor $1/\varepsilon$ fewer samples for each call. The second modification is the introduction of the if-statement in line 12. It turns out that this check only makes the algorithm run for a constant factor more iterations. However, this allows us to shave off a factor $1/\gamma$ in the number of calls to rThreshold. Since the replicability parameter of rThreshold needs to be ρ divided by the number of calls to rThreshold, this is a great improvement. This is because the sample complexity of rThreshold is inversely proportional to the square of its replicability parameter, so it will need a factor $1/\gamma^2$ fewer samples for each invocation of rThreshold. Since it is now only called every $1/\gamma$ iteration, we shave off a factor $1/\gamma^3$ in total by introducing this check. In Section 3, we will explain in more detail why these modifications preserve correctness, but for now we will just state the guarantees of rBoost*.

Algorithm 1 $\text{rBoost}^*_{\rho,\varepsilon}(S, \mathcal{W})$

Input: Samples S i.i.d. from $\mathcal{D}$, replicable γ-weak learner $\mathcal{W}$, replicability ρ, error ε.
Output: Hypothesis $H : \mathcal{X} \to \{-1, 1\}$.

$g_0(x) \stackrel{\text{def}}{=} 0$
$\mu_1(x) \stackrel{\text{def}}{=} 1$
$t \leftarrow 0$
while true do
  $t \leftarrow t + 1$
  $\mathcal{D}_{\mu_t}(x) \stackrel{\text{def}}{=} \mu_t(x)\mathcal{D}(x)/d(\mu_t)$
  $S_1 \leftarrow \widetilde{O}(m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}/\varepsilon)$ fresh samples from S
  $S_{\mathcal{W}} \leftarrow \text{RejectionSampler}(S_1, m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}, \mu_t)$
  $h_t \leftarrow$ Run $\mathcal{W}(S_{\mathcal{W}})$ with replicability $\Theta(\rho\varepsilon\gamma^2)$
  $g_t(x) \stackrel{\text{def}}{=} g_{t-1}(x) + h_t(x)f(x) - \gamma/(2+\gamma)$
  $\mu_{t+1}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } g_t(x) \le 0 \\ (1-\gamma)^{g_t(x)/2}, & \text{if } g_t(x) > 0 \end{cases}$
  if $\lfloor \frac{1}{\gamma} \rfloor$ divides t **then**
    $S_2 \leftarrow \widetilde{O}\left(\frac{1}{\rho^2\varepsilon^3\gamma^2}\right)$ fresh samples from S
    if $\text{rThreshold}(S_2, \varepsilon/2, \mu_t) = 0$ **then**
      Exit while loop
Return: $H \leftarrow \text{sign}(\sum_t h_t)$

Theorem 1.4 (rBoost*)**.** *For any $\rho, \varepsilon \in (0,1)$ and $\Theta(\rho\varepsilon\gamma^2)$-replicable weak learner $\mathcal{W}$ with advantage γ, rBoost* is ρ-replicable, makes $O(\frac{1}{\varepsilon\gamma^2})$ calls to $\mathcal{W}$, and with probability at least $1-\rho$ outputs a hypothesis H with $\text{Er}_{\mathcal{D}}(H) \le \varepsilon$. Furthermore, its sample complexity is*

$$m_{\text{rBoost}^*}(\rho, \varepsilon) = O\left(\frac{\ln(\frac{1}{\rho\varepsilon\gamma^2}) m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{\ln(\frac{1}{\rho\varepsilon\gamma})}{\rho^2\varepsilon^4\gamma^3}\right)$$
$$= \widetilde{O}\left(\frac{m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{1}{\rho^2\varepsilon^4\gamma^3}\right).$$

Remember that the original version of rBoost had sample complexity $\widetilde{O}(\frac{m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{1}{\rho^2\varepsilon^5\gamma^6})$. The dependence on γ has therefore improved greatly. However, we are still not happy with the dependence on ε. The main idea of our algorithm is therefore to use rBoost* as a subroutine and only call it with constant error parameter $\varepsilon_0 = 1/16$. Our algorithm can be seen as a meta boosting algorithm where in each iteration, we call rBoost* to get a hypothesis with constant advantage. We then perform exponential weight updates similar to AdaBoost in order to make our algorithm only run for $T = O(\ln(1/\varepsilon))$ iterations. Remark that this entirely removes the problem of rBoost* having a bad dependence on ε, since we only invoke it with a constant error parameter. This is the main insight needed to understand how our algorithm works.

2 Our Replicable Boosting Algorithm

In this section, we will present our new ρ-replicable boosting algorithm which can be found in Algorithm 2.

The algorithm runs for T iterations while maintaining functions N_t, M_t, μ_t. In each iteration the algorithm performs rejection sampling to get samples S_2 drawn from distribution $\mathcal{D}_{\mu_t}$. It then gets a hypothesis h_t from rBoost* which has constant error of at most ε_0 with respect to $\mathcal{D}_{\mu_t}$. One can interpret $N_t(x)$ as a lower bound for counting how many of the first $t-1$ hypotheses that misclassify element x. However, in order to ensure high density of the updated reweighing function μ_t, we check if the points sharing the largest count have a total probability mass of at least $\varepsilon/16$

by using RTHRESHOLD. If not, we subtract 1 from the largest count which suffices to ensure high density of μ_t (see Lemma 2.4). The capped values are stored in M_t, and will be used in subsequent iterations. The value c_t can be interpreted as a bound for the largest allowed count in iteration t, that is $c_t \geq M_t(x)$ for all $x \in \mathcal{X}$.

Algorithm 2 RMETABOOST$_{\rho,\varepsilon}(S, \mathcal{W})$

Input: Samples S i.i.d. from $\mathcal{D}$, replicable γ-weak learner $\mathcal{W}$, replicability ρ, error ε.
Output: Hypothesis $H : \mathcal{X} \to \{-1, 1\}$.

$N_1(x) \stackrel{\text{def}}{=} 0$
$M_1(x) \stackrel{\text{def}}{=} 0$
$\mu_1(x) \stackrel{\text{def}}{=} 1$
$c_1 \leftarrow 0$
for $t = 1$ to T **do** ▷ $T = O(1/\varepsilon)$
  $\mathcal{D}_{\mu_t}(x) \stackrel{\text{def}}{=} \mu_t(x)\mathcal{D}(x)/d(\mu_t)$
  $S_1 \leftarrow \widetilde{O}(m_{\text{RBOOST}^*}(\rho_0, \varepsilon_0)/\varepsilon)$ fresh samples from S ▷ $\rho_0 = \rho/(6T)$, $\varepsilon_0 = 1/16$
  $S_2 \leftarrow \text{REJECTIONSAMPLER}(S_1, m_{\text{RBOOST}^*}(\rho_0, \varepsilon_0), \mu_t)$
  $h_t \leftarrow \text{RBOOST}^*_{\rho_0,\varepsilon_0}(S_2, \mathcal{W})$
  $N_{t+1}(x) \stackrel{\text{def}}{=} M_t(x) + \mathbb{1}\{h_t(x) \neq f(x)\}$
  $S_3 \leftarrow \widetilde{O}(\frac{1}{\rho^2\varepsilon})$ fresh samples from S
  $b_{t+1} \leftarrow \text{RTHRESHOLD}(S_3, \varepsilon/16, \varphi)$ ▷ $\varphi(x) = \mathbb{1}\{N_{t+1}(x) = c_t + 1\}$
  $c_{t+1} \leftarrow c_t + b_{t+1}$
  $M_{t+1}(x) \stackrel{\text{def}}{=} \min(N_{t+1}(x), c_{t+1})$
  $\mu_{t+1}(x) \stackrel{\text{def}}{=} \exp(M_{t+1}(x) - c_{t+1})$
Return $H = \text{sign}\left(\sum_{t=1}^{T} h_t\right)$

Now, before going into the analysis of the algorithm, we will present the guarantees of the REJECTIONSAMPLER which we use to draw samples from $\mathcal{D}_\mu$. The pseudocode and proofs of the below guarantees are described by Impagliazzo et al. [2022], so we will not repeat those here.

Lemma 2.1 (Rejection Sampling [Impagliazzo et al., 2022])**.** *For any $\varepsilon \in (0, 1]$, if μ has density $d(\mu) \geq \varepsilon$ and $S \sim \mathcal{D}^m$ where $m \geq 8\ln(1/\delta)m_{target}/\varepsilon$, then REJECTIONSAMPLER$(S, m_{target}, \mu)$ outputs a sample $S_{out} \sim \mathcal{D}_\mu^{m_{target}}$ with probability at least $1 - \delta$.*

Lemma 2.2 (Composing Replicable Algorithms with Rejection Sampling [Impagliazzo et al., 2022])**.** *Let $\mathcal{A}(S)$ be a ρ-replicable algorithm with sample complexity m. Let $\mu : \mathcal{X} \to [0, 1]$. Then let $\mathcal{B}$ be the algorithm that runs $\mathcal{A}$ with samples drawn from $\mathcal{D}_\mu$ using rejection sampling. Let q be the failure probability of REJECTIONSAMPLER. Then $\mathcal{B}$ is $(2q + 2\rho)$-replicable.*

2.1 Analysis of RMETABOOST

We are now ready to analyze RMETABOOST. To make it easier to follow the analysis, we will split it into four parts.

1. Correctness,
2. Replicability,
3. Sample complexity,
4. Failure probability.

We will start with correctness. However, before going into the formal details, we will give an explanation of the high level ideas in the proof. First, observe that if we did not cap the weights N_t, the multiplicative weight updates would be very similar to the updates made in ADABOOST. Recall that for any $t \in [T]$, $M_t(x)$ is exactly the number

of misclassifications of x minus the amount of times we have capped the weight so far. Hence, if we did not cap the weights by c_t each iteration, x would be misclassified by the final hypothesis H only if $M_{T+1}(x) \geq T/2$. We will now take the capping into account. We first show using an argument similar to the standard analysis of AdaBoost that the probability of drawing an x from $\mathcal{D}$ for which $M_{T+1}(x) \geq T/4$ is at most $\varepsilon/2$. What remains is to argue that the probability of drawing an x which is misclassified but simultaneously satisfies $M_{T+1}(x) < T/4$ is small. The only way this can happen is if there were at least $T/4$ iterations in which we capped down the value of $N_{t+1}(x)$ when calculating $M_{t+1}(x)$, since in such iterations we would not increment $M_{t+1}(x)$ even though h_t misclassified x. Observe that due to the threshold check, the total probability mass (w.r.t. $\mathcal{D}$) of points whose value of N_{t+1} were capped in a single iteration cannot exceed $\varepsilon/8$. Therefore, after T iterations, the total probability mass of points, whose value of N_{t+1} were capped $T/4$ times is at most $T(\varepsilon/8)/(T/4) = \varepsilon/2$. So in total, the probability mass of all the misclassified points is at most ε. We now prove this formally.

Lemma 2.3 (Correctness)**.** *Put $T \geq 8\ln(2/\varepsilon)$ and $\varepsilon_0 = 1/16$. Assuming that all subroutines of the algorithm succeed in every iteration, we achieve an error of at most ε over the distribution $\mathcal{D}$, i.e. $\mathrm{Er}_{\mathcal{D}}(H) \leq \varepsilon$.*

Proof. By definition of M_{T+1}, N_{T+1} and μ_T

$$\begin{aligned}
&\mathbb{E}[\exp(M_{T+1}(X))] \leq \mathbb{E}[\exp(N_{T+1}(X))] \\
&= \mathbb{E}[\exp(M_T(X))\exp(\mathbb{1}\{h_T(X) \neq f(X)\})] \\
&= e^{c_T}\mathbb{E}[\mu_T(X)\exp(\mathbb{1}\{h_T(X) \neq f(X)\})] \\
&= e^{c_T}\big(\mathbb{E}[\mu_T(X)\mathbb{1}\{h_T(X) = f(X)\}] \\
&\quad + e\mathbb{E}[\mu_T(X)\mathbb{1}\{h_T(X) \neq f(X)\}]\big). \qquad (1)
\end{aligned}$$

Now, since $\mathcal{D}_{\mu_T} = \frac{\mu_T \cdot \mathcal{D}}{d(\mu_T)}$, we can rewrite the above to an expectation involving $Y \sim \mathcal{D}_{\mu_T}$ such that (1) is equal to

$$\begin{aligned}
&e^{c_T} d(\mu_T)\big(\mathbb{E}[\mathbb{1}\{h_T(Y) = f(Y)\}] \\
&\quad + e\mathbb{E}[\mathbb{1}\{h_T(Y) \neq f(Y)\}]\big) \\
&= e^{c_T} d(\mu_T)\Big(\mathrm{Er}_{\mathcal{D}_{\mu_T}}(h_T)(e-1)+1\Big) \\
&\leq e^{c_T} d(\mu_T)\exp\Big((e-1)\,\mathrm{Er}_{\mathcal{D}_{\mu_T}}(h_T)\Big) \\
&\leq e^{c_T} d(\mu_T)\exp(2\varepsilon_0), \qquad (2)
\end{aligned}$$

where the final inequality follows since h_t has error at most ε_0 under $\mathcal{D}_{\mu_T}$ by Theorem 1.4. Now, note that

$$\begin{aligned}
e^{c_T} d(\mu_T) &= e^{c_T}\mathbb{E}[\mu_T(X)] = e^{c_T}\mathbb{E}[\exp(M_T(X) - c_T)] \\
&= \mathbb{E}[\exp(M_T(X))].
\end{aligned}$$

Plugging this into (2), we recursively get

$$\begin{aligned}
\mathbb{E}[\exp(M_{T+1}(X))] &\leq \mathbb{E}[\exp(M_T(X))]\exp(2\varepsilon_0) \\
&\leq \cdots \leq \exp(2T\varepsilon_0)
\end{aligned}$$

Now, define the sets $A = \{x : M_{T+1}(x) \geq T/4\}$ and $B = A^c \cap \{x : H(x) \neq f(x)\}$ and note that

$$\mathrm{Er}_{\mathcal{D}}(H) \leq \mathbb{P}(X \in A) + \mathbb{P}(X \in B).$$

For bounding $\mathbb{P}(X \in A)$, we have

$$\begin{aligned}
\mathbb{E}\left[\exp(M_{T+1}(X))\right] &\geq \mathbb{E}\left[\exp(M_{T+1}(X))\mathbb{1}_A(X)\right] \\
&\geq \exp(T/4)\mathbb{P}(X \in A),
\end{aligned}$$

and hence

$$\begin{aligned}\mathbb{P}(X \in A) &\leq \exp(-T/4)\mathbb{E}\left[\exp(M_{T+1}(X))\right] \\ &\leq \exp(T(2\varepsilon_0 - 1/4)) \\ &= \exp(-T/8) \leq \varepsilon/2.\end{aligned}$$

Bounding $\mathbb{P}(X \in B)$:
First, observe that $0 \leq M_{t+1}(x) - M_t(x) \leq 1$ for all $t \in [T], x \in \mathcal{X}$. Now, let $x \in B$. Since H is a majority classifier, we have

$$\begin{aligned}T/2 &\leq \sum_{t=1}^{T} \mathbb{1}\{h_t(x) \neq f(x)\} \\ &= \sum_{t=1}^{T} \mathbb{1}\{N_{t+1}(x) > c_{t+1}\} + M_{T+1}(x).\end{aligned}$$

Taking the expectation over the event $\{X \in B\}$ on both ends of the above then yields

$$\begin{aligned}&T\mathbb{P}(X \in B)/2 = T\mathbb{E}[\mathbb{1}\{X \in B\}]/2 \\ &\leq \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}\{N_{t+1}(X) > c_{t+1}\}\mathbb{1}\{X \in B\}] \\ &\quad + \mathbb{E}[M_{T+1}(X)\mathbb{1}\{X \in B\}] \\ &\leq \sum_{t=1}^{T} \mathbb{P}[N_{t+1}(X) > c_{t+1}] + \mathbb{E}[M_{T+1}(X)\mathbb{1}\{X \in B\}] \\ &\leq \sum_{t=1}^{T} \mathbb{P}[N_{t+1}(X) > c_{t+1}] + T\mathbb{P}(X \in B)/4.\end{aligned}$$

Due to the threshold check in line 12 and Lemma 1.3, we know that if $b_t = 0$, then we must have $\mathbb{P}[N_{t+1}(X) = c_t + 1] \leq \varepsilon/8$. Furthermore, in this case $c_{t+1} = c_t$. Hence,

$$\begin{aligned}\mathbb{P}[N_{t+1}(X) > c_{t+1}] &= \mathbb{P}[N_{t+1}(X) > c_t] \\ &= \mathbb{P}[N_{t+1}(X) = c_t + 1] \leq \varepsilon/8.\end{aligned}$$

If instead $b_t = 1$, then

$$N_{t+1}(x) \leq M_t(x) + 1 \leq c_t + 1 = c_{t+1},$$

which implies

$$\mathbb{P}[N_{t+1}(X) > c_{t+1}] = 0.$$

Hence, we get the bound

$$\begin{aligned}&T\mathbb{P}(X \in B)/2 \\ &\leq \sum_{t=1}^{T} \mathbb{P}[N_{t+1}(X) > c_{t+1}] + T\mathbb{P}(X \in B)/4 \\ &\leq T\varepsilon/8 + T\mathbb{P}(X \in B)/4.\end{aligned}$$

Rearranging gives $\mathbb{P}(X \in B) \leq \varepsilon/2$, meaning we in total have

$$\mathrm{Er}_{\mathcal{D}}(H) \leq \mathbb{P}(X \in A) + \mathbb{P}(X \in B) = \varepsilon/2 + \varepsilon/2 = \varepsilon. \qquad \square$$

For the remaining parts, we need the guarantee of Lemma 2.1 that rejection sampling fails with low probability when μ_t has large density. Hence, we first show that the density is indeed large.

Lemma 2.4 (High density of μ_t)**.** *Assume that* rThreshold *succeeds in every iteration in* rMetaBoost*. Then for any $t \in [T]$, μ_t has density $d(\mu_t) \geq \varepsilon/32$.*

Proof. Let $X \sim \mathcal{D}$. Then using the law of total expectation and the definition of μ_t we have

$$\begin{aligned} d(\mu_t) &= \mathbb{E}[\mu_t(X)] \\ &\geq \mathbb{E}[\mu_t(X)|M_t(X) \geq c_t]\mathbb{P}[M_t(x) \geq c_t] \\ &= \mathbb{E}[\exp(M_t(X) - c_t)|M_t(X) \geq c_t]\mathbb{P}[M_t(x) \geq c_t] \\ &\geq \mathbb{P}[M_t(X) \geq c_t]. \end{aligned}$$

We now show by induction in t that $\mathbb{P}[M_t(X) \geq c_t] > \varepsilon/32$. For $t = 1$, we have $M_1(X) = c_1 = 0$, and hence $\mathbb{P}[M_t(X) \geq c_t] = 1$. Now, assume the claim holds for t. We will then show that it holds for $t+1$ by case analysis on b_{t+1}. If $b_{t+1} = 0$, we have $c_{t+1} = c_t$ and

$$\begin{aligned} \mathbb{P}[M_{t+1}(X) \geq c_{t+1}] &= \mathbb{P}[M_{t+1}(X) \geq c_t] \\ &\geq \mathbb{P}[M_t(X) \geq c_t] > \varepsilon/32 \end{aligned}$$

using the induction hypothesis and the fact that $M_{t+1}(X) \geq M_t(X)$. Now assume $b_{t+1} = 1$. Then we know by Lemma 1.3 that $\mathbb{P}\left[N_{t+1}(X) = c_t + 1\right] > \varepsilon/32$. Since $b_{t+1} = 1$, then $c_{t+1} = c_t + 1$ which then implies that

$$\begin{aligned} &\mathbb{P}[M_{t+1}(X) \geq c_{t+1}] = \mathbb{P}[M_{t+1}(X) \geq c_t + 1] \\ &= \mathbb{P}[\min(N_{t+1}(X), c_{t+1}) \geq c_t + 1] \\ &= \mathbb{P}[N_{t+1}(X) \geq c_t + 1] > \varepsilon/32. \end{aligned}$$

□

Lemma 2.5 (Replicability)**.** *rMetaBoost is ρ-replicable.*

Proof. Let S_1, S_2 be two independent samples with distribution $\mathcal{D}^m$ for some $m = \widetilde{O}\Big(\frac{m_{\mathcal{W}(\tilde{\Theta}(\rho\gamma^2))}}{\varepsilon\gamma^2} + \frac{1}{\rho^2\varepsilon\gamma^3}\Big)$. Assume that in iterations $1, \ldots, t-1$, the algorithm has produced the same objects, i.e. that the reweighing functions and hypotheses associated with S_1 and S_2 are the same. Then, for iteration t to be replicable, we need the following:

1. rBoost* outputs the same hypothesis for both samples.
2. rThreshold outputs the same bit for both samples.

When these conditions hold, the rest of the quantities appearing in the algorithm will be the same for both samples and hence ensure replicability. We call rBoost* with replicability parameter $\rho_0 = \rho/(6T)$ and call RejectionSampler with at least $8\ln(6T/\rho)/\varepsilon$ samples. Since Lemma 2.4 tells us that the density of μ_t satisfies $d(\mu_t) > \varepsilon/32$, we can use Lemmas 2.1 and 2.2 to conclude that rBoost* combined with RejectionSampler is $2\rho/(6T) + 2\rho/(6T) = 2\rho/(3T)$-replicable. Finally, by Lemma 1.3, rThreshold is $\rho/(3T)$-replicable. Hence, by a union bound over the conditions, each iteration is ρ/T-replicable and union bounding over all T iterations, the entire algorithm is ρ-replicable. □

Lemma 2.6 (Sample complexity)**.** *rMetaBoost uses $m = \widetilde{O}\Big(\frac{m_{\mathcal{W}(\tilde{\Theta}(\rho\gamma^2))}}{\varepsilon\gamma^2} + \frac{1}{\rho^2\varepsilon\gamma^3}\Big)$ samples.*

Proof. For the sample complexity of a single iteration, we simply add up the sample complexities of all the subroutines:

- rBoost*: Since we give it parameters $\rho_0 = \rho/(6T)$,$\varepsilon_0 = 1/16$, we get from Theorem 1.4 that the sample complexity of rBoost* in a single iteration is

$$m_{\text{rBoost}^*}(\rho_0, \varepsilon_0) = O\left(\frac{\ln(\frac{T}{\rho\gamma^2})m_{\mathcal{W}(\Theta(\frac{\rho\gamma^2}{T}))}}{\gamma^2} + \frac{\ln(\frac{T}{\rho\gamma})T^2}{\rho^2\gamma^3}\right)$$

  Remark that the choice of constant ε_0 removes all the dependence on ε.

- RejectionSampler: To invoke Lemma 2.1 with failure probability $\rho/(6T)$ the number of samples used in each iteration is
$$O\left(\frac{\ln(\frac{T}{\rho})m_{\text{rBoost}^*}(\rho_0,\varepsilon_0)}{\varepsilon}\right).$$
- rThreshold: To invoke Lemma 1.3 with replicability parameter $\rho/(3T)$ and failure probability $\rho/(24T)$ the number of samples used in each iteration is
$$O(\frac{\ln(\frac{T}{\rho})T^2}{\rho^2\varepsilon}).$$

Remembering that the number of iterations is $T = O(\ln(1/\varepsilon))$ we get the total sample complexity to be

$$O\left(T\left(\frac{\ln(\frac{T}{\rho})\ln(\frac{T}{\rho\gamma^2})m_{\mathcal{W}(\Theta(\frac{\rho\gamma^2}{T}))}}{\varepsilon\gamma^2}+\frac{\ln(\frac{T}{\rho})\ln(\frac{T}{\rho\gamma})T^2}{\rho^2\varepsilon\gamma^3}+\frac{\ln(\frac{T}{\rho})T^2}{\rho^2\varepsilon}\right)\right)$$
$$=\widetilde{O}\left(\frac{m_{\mathcal{W}(\widetilde{\Theta}(\rho\gamma^2))}}{\varepsilon\gamma^2}+\frac{1}{\rho^2\varepsilon\gamma^3}\right)$$ □

Lemma 2.7 (Failure probability). *rMetaBoost fails with probability at most $9\rho/24 \leq \rho$.*

Proof. The only sources of failure are the three subroutines. RejectionSampler fails with probability $\rho/(6T)$ in each iteration. rThreshold fails with probability $\rho/(24T)$ in each iteration. rBoost* fails with probability at most $\rho/(6T)$ in each iteration. Hence, the total failure probability of the algorithm is at most $9\rho/24$. □

3 Subroutines

In this section, we will present the replicable subroutines that the boosting algorithm uses. This includes rThreshold and rBoost*. As mentioned earlier, we will not present RejectionSampler as we have made no changes to it, so we refer to Impagliazzo et al. [2022] for the description of this subroutine. We now move on to describe the two other subroutines.

3.1 rThreshold

In this section, we will describe rThreshold in more detail. The pseudocode can be found in Algorithm 3. The purpose of this algorithm is to make a replicable test to see if $\mathbb{E}[\varphi(X)] > z$ for some threshold $z \in (0,1)$ and $\varphi : \mathcal{X} \to [0,1]$. In the original version of rBoost, this is done by replicably simulating a statistical query for estimating $\mathbb{E}[\varphi(X)]$, with an additive error of order z. However, for a threshold check it suffices to have a multiplicative error when estimating $\mathbb{E}[\varphi(X)]$, which means we can do a better analysis by using a Chernoff bound.

Algorithm 3 rThreshold(S, z, φ)

Input: Samples $S = (x_1, \ldots, x_m)$ drawn from $\mathcal{D}$, threshold z, function $\varphi : \mathcal{X} \to [0,1]$.
Output: Bit b being a guess, whether $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] > z$.

1: $z_0 \leftarrow_r [\frac{3}{4}z, \frac{3}{2}z]$ ▷ Chosen uniformly at random
2: $\overline{\varphi(S)} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\varphi(x_i)$
3: **Return:** $b = \mathbb{1}\left\{\overline{\varphi(S)} > z_0\right\}$

We will now prove the guarantee of rThreshold. For convenience, we restate the guarantee here.

Lemma 1.3 Restated. *Let $z, \rho \in (0,1)$, $\delta \in (0, \rho/8]$, let $\varphi : \mathcal{X} \to [0,1]$ and let $S = (x_1, \ldots, x_m)$ be samples drawn i.i.d. from distribution $\mathcal{D}$. Then there exists a constant c such that if $m \geq c\frac{\ln(1/\delta)}{\rho^2 z}$, rThreshold$(S, z, \varphi)$ is ρ-replicable and returns a bit b such that with probability at least $1-\delta$:*

- *If* $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \leq z/2$*, then* $b = 0$.
- *If* $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \geq 2z$*, then* $b = 1$.

Proof. It is sufficient to set $m \geq \frac{700\ln(1/\delta)}{z\rho^2}$. We will first prove the first bullet point. Hence, assume that $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \leq z/2$. We then bound the following probability:

$$\begin{aligned}\mathbb{P}\left[b=1\right] &= \mathbb{P}\left[\overline{\varphi(S)} > z_0\right] \leq \mathbb{P}\left[\overline{\varphi(S)} > \frac{3}{4}z\right] \\ &= \mathbb{P}\left[\sum_{i=1}^{m}\varphi(x_i) > (1+\frac{1}{2})m(z/2)\right].\end{aligned}$$

Since the assumption states that $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \leq z/2$, we can use a Chernoff bound to bound the above probability by

$$\exp(-mz/24) \leq \exp(-\ln(1/\delta)/\rho^2) = \delta^{1/\rho^2} \leq \delta.$$

We move on to the second bullet point. Hence, we now assume $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \geq 2z$ and bound the following probability:

$$\begin{aligned}\mathbb{P}\left[b=0\right] &= \mathbb{P}\left[\overline{\varphi(S)} \leq z_0\right] \leq \mathbb{P}\left[\overline{\varphi(S)} \leq \frac{3}{2}z\right] \\ &= \mathbb{P}\left[\sum_{i=1}^{m}\varphi(x_i) \leq (1-\frac{1}{4})m(2z)\right].\end{aligned}$$

Again, since $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \geq 2z$, we can use a Chernoff bound to bound the above probability by

$$\exp(-mz/16) \leq \exp(-\ln(1/\delta)/\rho^2) = \delta^{1/\rho^2} \leq \delta.$$

We will now show that rThreshold is ρ-replicable by considering two different runs of the algorithm with common randomness. Let $S_1, S_2 \sim \mathcal{D}^m$ be the two sequences of samples used in the two runs. Assuming that neither of the runs fail, they will always output the same bit if $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \notin [z/2, 2z]$, so we can safely assume that this is not the case. Now, we will bound the probability that $\overline{\varphi(S_i)}$ deviates too much from $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)]$. So we bound the following probabilities using Chernoff bounds and the assumption that $\mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \in [z/2, 2z]$:

$$\begin{aligned}&\mathbb{P}\left[\overline{\varphi(S_i)} - \mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \geq 3z\rho/16\right] \\ &\leq \exp(-3z\rho^2 m/2048) \leq \delta \leq \rho/8\end{aligned}$$

and

$$\begin{aligned}&\mathbb{P}\left[\overline{\varphi(S_i)} - \mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)] \leq -3z\rho/16\right] \\ &\leq \exp(-9z\rho^2 m/4096) \leq \delta \leq \rho/8.\end{aligned}$$

Using a union bound, we can conclude that

$$\mathbb{P}\left[\left|\overline{\varphi(S_i)} - \mathbb{E}_{x\sim\mathcal{D}}[\varphi(x)]\right| \geq 3z\rho/16\right] \leq \rho/4.$$

We can therefore conclude that with high probability, the two estimates will be close to each other. That is,

$$\mathbb{P}\left[\left|\overline{\varphi(S_1)} - \overline{\varphi(S_2)}\right| \geq 3z\rho/8\right] \leq \rho/2.$$

So assume for now, that the two estimates are within $3z\rho/8$ of each other. Then the two runs will only give different outputs if the random split z_0 is chosen between them. The probability of this happening is at most the distance between the estimates divided by the total range of z_0, which is at most

$$\frac{3z\rho/8}{3z/2 - 3z/4} = \rho/2.$$

So the probability that the two runs output different bits is at most $\rho/2 + \rho/2 = \rho$. Therefore, the algorithm is ρ-replicable. □

3.2 rBoost*

We now move on to discuss in more detail why the modifications in rBoost* preserve correctness. The modified version can be seen in Algorithm 1. The only two modifications can be found in line 12 and 14. In line 14 we have substituted a statistical query with our threshold check, and in line 12 we have inserted an if-statement to only do the threshold check every $1/\gamma$ iteration. In this algorithm, the threshold check gives the same guarantees as the statistical query, and it will therefore not affect correctness. However, the introduction of the if-statement could lead to two kinds of errors, since we do not check the value of $d(\mu_t)$ in every iteration. First, it could be that RejectionSampler fails, since it needs $d(\mu_t)$ to be large. Second, it could be that the number of iterations is increased, since the algorithm does not detect immediately when the density becomes small. To show that these events are not problematic, we first show that the densities do not decrease too much over $1/\gamma$ iterations.

Lemma 3.1. *Let T_0 denote the number of iterations that rBoost* runs for. Then for all $t \in [T_0]$ and $k \leq \max\{\lfloor 1/\gamma \rfloor, T_0 - t\}$ that $d(\mu_{t+k}) \geq d(\mu_t)/2$.*

Proof. Let $x \in \mathcal{X}$. Then by the recursive definition of the g_t's, we have $\mu_{t+1}(x) \geq (1-\gamma)^{1/2}\mu_t(x)$. Inductively, we get

$$\begin{aligned}\mu_{t+k}(x) &\geq \mu_t(x)(1-\gamma)^{k/2} \geq \mu_t(1-\gamma)^{\lfloor \frac{1}{\gamma} \rfloor/2} \\ &\geq \mu_t(x)\left(1 - \gamma\lfloor\frac{1}{\gamma}\rfloor/2\right) \geq \frac{1}{2}\mu_t(x),\end{aligned}$$

where we use Bernoulli's inequality which applies since $-\gamma > -1$. Taking the expectation with respect to $\mathcal{D}$ on both sides yields the desired conclusion. □

Theorem 1.4 Restated. *For any $\rho, \varepsilon \in (0,1)$ and $\Theta(\rho\varepsilon\gamma^2)$-replicable weak learner $\mathcal{W}$ with advantage γ, rBoost* is ρ-replicable, makes $O(\frac{1}{\varepsilon\gamma^2})$ calls to $\mathcal{W}$, and with probability at least $1-\rho$ outputs a hypothesis H with $\mathrm{Er}_{\mathcal{D}}(H) \leq \varepsilon$. Furthermore, its sample complexity is*

$$\begin{aligned}m_{\text{rBoost*}}(\rho,\varepsilon) &= O\left(\frac{\ln(\frac{1}{\rho\varepsilon\gamma^2})m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{\ln(\frac{1}{\rho\varepsilon\gamma})}{\rho^2\varepsilon^4\gamma^3}\right) \\ &= \widetilde{O}\left(\frac{m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{1}{\rho^2\varepsilon^4\gamma^3}\right).\end{aligned}$$

Proof. First, we argue that the RejectionSampler succeeds with high probability. Observe that due to Lemma 3.1, the densities are at most halved in rBoost* compared to the original version of rBoost. Due to Lemma 2.1 we therefore only need to use twice as many samples in the rejection sampler for it to still succeed.

Next, we will argue that the number of iterations remains the same as in rBoost up to constant factors. The number of iterations is bounded in Servedio [2001] by showing that for any $\kappa > 0$, there is some t within the first $O(\frac{1}{\kappa\gamma^2})$ iterations such that $d(\mu_t) < \kappa$. This result also applies to rBoost*. However, we will not repeat the proof here.

We can then conclude that $d(\mu_t)$ will fall below $\varepsilon/8$ within the first $O(\frac{1}{\varepsilon\gamma^2})$ iterations. Since the threshold check in line 14 always realizes when $d(\mu_t) \leq \varepsilon/4$ (see Lemma 1.3), and it takes $1/\gamma$ iterations to further decrease the density from $\varepsilon/4$ to $\varepsilon/8$, rThreshold will always have terminated the loop before reaching density $\varepsilon/8$. Hence, the number of iterations in rBoost* is still $T_0 = O(\frac{1}{\varepsilon\gamma^2})$.

We now argue, that replicability is preserved. Since this argument is almost identical to the proof of Lemma 2.5, we will only describe what differs in this analysis. First, the weak learner is called T_0 times, and therefore it needs replicability parameter $\rho/(6T_0) = \Theta(\rho\varepsilon\gamma^2)$. Second, rThreshold is called γT_0 times and therefore needs replicability parameter $\rho/(6\gamma T_0) = \rho\varepsilon\gamma/6$, which it achieves due to Lemma 1.3. Thus, our modifications preserve replicability.

Finally, we calculate the sample complexity. The RejectionSampler uses $O(\ln(\frac{T_0}{\rho})m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}/\varepsilon)$ samples for each call, and it is called T_0 times. Meanwhile, rThreshold uses $O(\frac{\ln(\gamma T_0/\rho)}{\rho^2\varepsilon^3\gamma^2})$ samples for each call, and is called

γT_0 times. Hence, the total sample complexity is

$$
\begin{aligned}
&O\left(T_0 \frac{\ln(\frac{T_0}{\rho}) m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon} + \gamma T_0 \frac{\ln(\frac{\gamma T_0}{\rho})}{\rho^2 \varepsilon^3 \gamma^2}\right) \\
&= O\left(\frac{\ln(\frac{1}{\rho\varepsilon\gamma^2}) m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{\ln(\frac{1}{\rho\varepsilon\gamma})}{\rho^2\varepsilon^4\gamma^3}\right) \\
&= \widetilde{O}\left(\frac{m_{\mathcal{W}(\Theta(\rho\varepsilon\gamma^2))}}{\varepsilon^2\gamma^2} + \frac{1}{\rho^2\varepsilon^4\gamma^3}\right).
\end{aligned}
$$

□

References

S. Ahmadi, S. Bhandari, and A. Blum. Replicable online learning. *arXiv preprint arXiv:2411.13730*, 2024.

M. Baker. Reproducibility crisis. *nature*, 533(26):353–66, 2016.

P. Ball. Is ai leading to a reproducibility crisis in science? *Nature*, 624(7990):22–25, 2023.

B. Barak, M. Hardt, and S. Kale. The uniform hardcore lemma via approximate bregman projections. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1193–1200. SIAM, 2009.

M. Bun, M. Gaboardi, M. Hopkins, R. Impagliazzo, R. Lei, T. Pitassi, S. Sivakumar, and J. Sorrell. Stability is stable: Connections between replicability, privacy, and adaptive generalization. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 520–527, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399135. doi: 10.1145/3564246.3585246. URL https://doi.org/10.1145/3564246.3585246.

P. Dixon, J. Vander Woude, and N. Vinodchandran. List and certificate complexities in replicable learning. *Advances in Neural Information Processing Systems*, 36, 2024.

C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *2010 IEEE 51st annual symposium on foundations of computer science*, pages 51–60. IEEE, 2010.

E. Eaton, M. Hussing, M. Kearns, and J. Sorrell. Replicable reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

H. Esfandiari, A. Karbasi, V. Mirrokni, G. Velegkas, and F. Zhou. Replicable clustering. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49195-8.

R. Impagliazzo, R. Lei, T. Pitassi, and J. Sorrell. Reproducibility in learning. In *Proceedings of the 54th annual ACM SIGACT symposium on theory of computing*, pages 818–831, 2022.

A. Kalavasis, A. Karbasi, K. G. Larsen, G. Velegkas, and F. Zhou. Replicable learning of large-margin halfspaces. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 22861–22878. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/kalavasis24a.html.

M. Kearns. Learning boolean formulae or finite automata is as hard as factoring. *Technical Report TR-14-88 Harvard University Aikem Computation Laboratory*, 1988.

M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

R. E. Schapire. The strength of weak learnability. *Machine learning*, 5:197–227, 1990.

R. A. Servedio. Smooth boosting and learning with malicious noise. In D. Helmbold and B. Williamson, editors, *Computational Learning Theory*, pages 473–489, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44581-4.