CS7545, Spring 2023: Machine Learning Theory - Homework #3

Jacob Abernethy, Zihao Hu, and Guanghui Wang Due: Tuesday, March 19 at 11:59 p.m.

Homework Policy: *The due is at Sunday 3/19, but everyone gets a free extension to 3/28 (late submissions are not guarantee to be graded before exam).* Working in groups is fine, but *every student* must submit their own writeup. Please write the members of your group on your solutions. There is no strict limit to the size of the group but we may find it a bit suspicious if there are more than 4 to a team. Questions labelled with **(Challenge)** are not strictly required, but you'll get some participation credit if you have something interesting to add, even if it's only a partial answer.

1) **Tuning Parameters.** We are going to imagine we have some algorithm $\mathcal{A}$ with a performance bound that depends on some input values (which can not be adjusted) and some tuning parameters (which can be optimized). We will use greek letters (α, η, ζ, etc.) for the tuning parameters and capital letters (T, D, N, etc.) for inputs. We would like the bound to be the tightest possible, up to multiplicative constants. For each of the following, tune the parameters to obtain the optimal bound. Using *big-Oh* notation is fine to hide constants, but you *must not* ignore the dependence on the input parameters. For example, assuming $M, T > 0$, imagine we have a performance guarantee of the form:

$$\text{Performance}(\mathcal{A}; M, T, \epsilon) \leq M\epsilon + \frac{T}{\epsilon} \tag{1}$$

and we know $\epsilon > 0$. Then by optimizing the above expression with respect to the free parameter we can set $\epsilon = \sqrt{\frac{T}{M}}$. With this value we obtain $\text{Performance}(\mathcal{A}; M, T, \epsilon) = O(\sqrt{MT})$

NOTE: I didn't have to make up this problem, I actually pulled all the bounds below from different papers!

(a) $\text{Performance}(\mathcal{A}; T, N, \eta) \leq \frac{\log N + \eta T}{1 - \exp(-\eta)}$

(b) $\text{Performance}(\mathcal{A}; T, \eta) \leq \max(T\eta, \eta^{-2})$

(c) $\text{Performance}(\mathcal{A}; T, \eta) \leq \frac{T}{\eta} + \exp(\eta)$. (Note: you needn't obtain the optimal choice of η here or the tightest possible bound, but try to tune in order to get a bound that is $o(T)$ – i.e. the bound should grow strictly slower than linear in T.)

(d) $\text{Performance}(\mathcal{A}; N, T, \eta, \epsilon) \leq \frac{T\epsilon}{\eta} + \frac{N}{\epsilon} + T\eta$

2) **The Doubling Trick.** Let's imagine that an online learning algorithm that runs in T rounds has the bound in Eq. (1). By optimally tuning ϵ we obtain a bound of the form $O(\sqrt{TM})$. The problem with this approach is that it requires us to know T in advance. Is there a way around this issue?

Imagine constructing a modified algorithm $\mathcal{A}'$ that does the following iterative procedure. $\mathcal{A}'$ starts with an initial parameter ϵ_1, implements algorithm $\mathcal{A}$ using this parameter for T_1 rounds, then adjusts the parameter to ϵ_2, and runs $\mathcal{A}$ for T_2 rounds, and so forth. Let's say ϵ gets updated k times, where $T_1 + T_2 + \ldots + T_k = T$. You can also assume that $\text{Performance}(\mathcal{A}'; T, M) = \sum_{i=1}^{k} \text{Performance}(\mathcal{A}; T_i, M, \epsilon_i)$.

Can you construct a schedule for the sequence of (ϵ_i, T_i) that achieves the same bound (up to a multiplicative constant factor) as the optimally tuned bound (namely, $O(\sqrt{MT})$), even though T is unknown in advance? In other words, you want to choose the sequence of $T_1, T_2, \ldots$, with the associated $\epsilon_1, \epsilon_2, \ldots$ so that whenever the game truly ends, at a previously unknown T, the bound Performance$(\mathcal{A}'; T, M) = O(\sqrt{MT})$ will always hold. (Note: this is referred to as a "doubling trick".)

3) **Dynamic Regret.** In Lecture 16, we learned how to get the regret guarantee of convex functions by OGD. In this problem, we consider a generalized notation of regret: dynamic regret, which is defined as

$$\text{Regret }_T := \sum_{t=1}^{T} f_t\left(\mathbf{w}_t\right) - \sum_{t=1}^{T} f_t(\mathbf{w}_t^*)$$

where $\mathbf{w}_t^* \in K$ and $\sum_{t=2}^{T} \|\mathbf{w}_t^* - \mathbf{w}_{t-1}^*\|_2 \leq P_T$. For convenience we assume $0 \in K$. Can you get the regret guarantee in terms of G, D, T, and P_T? What is the optimal η in this scenario?

Hint: you can start from

$$\text{Regret}_T \leq \sum_{t=1}^{T} \frac{\eta}{2} G^2 + \sum_{t=1}^{T} \frac{\|\mathbf{w}_t - \mathbf{w}_t^*\|_2^2 - \|\mathbf{w}_{t+1} - \mathbf{w}_t^*\|_2^2}{2\eta},$$

and consider how to use the definitions of D and P_T to finish the telescoping sum.

4) **Exponential Weights Algorithm with a Prior.** The Exponential Weights Algorithm had the initial weights $w_1^1, \ldots, w_n^1$ all set to 1. What if instead we imagine an algorithm $\mathcal{A}'$ where we set these weights according to $w_i^1 = p_i$ where $\vec{p}$ is some distribution (i.e. $p_i \geq 0$ for all i and $\sum_i p_i = 1$). We will do the same multiplicative update rule as before.

Prove that with this modified algorithm we achieve the following bound: for any expert i we have that

$$\text{Loss}_T(\mathcal{A}) \leq \frac{\log p_i^{-1} + \eta \text{Loss}_T(\text{expert } i)}{1 - e^{-\eta}}$$

5) **Online Non-Convex Optimization.** Sometimes our nice assumptions don't always hold. But maybe things will still work out just fine. For the rest of this problem assume that $X \subset \mathbb{R}^n$ is the learner's decision set, and the learner observes a sequence of functions $f_1, f_2, \ldots, f_T$ mapping $X \to \mathbb{R}$. The regret of an algorithm choosing a sequence of $\mathbf{x}_1, \mathbf{x}_2, \ldots$ is defined in the usual way:

$$\text{Regret }_T := \sum_{t=1}^{T} f_t\left(\mathbf{x}_t\right) - \min_{\mathbf{x} \in X} \sum_{t=1}^{T} f_t(\mathbf{x})$$

Wouldn't it ruin your lovely day if the functions f_t were not convex? Maybe the only two conditions you can guarantee is that the functions f_t are bounded (say in $[0,1]$) and are 1-Lipschitz: they satisfy that $|f_t(\mathbf{x}) - f_t\left(\mathbf{x}'\right)| \leq \|\mathbf{x} - \mathbf{x}'\|_2$. Prove that, assuming X is convex and bounded, there exists a randomized algorithm with a reasonable expected-regret bound. Something like $\mathbb{E}\left[\text{Regret}_T\right] \leq \sqrt{nT \log T}$ would be admirable. (Hint: Always good to ask the experts for ideas. And you needn't worry about efficiency.)

6) **Subsets as Experts.** **(Challenge)** We saw that when we wanted to do "predictive sorting" it's not easy to apply the halving algorithm to the class of all permutations (rankings) as experts. But this isn't the case for all classes of "complex experts".

Imagine a setting where we have N experts but our goal is not to choose one but $k < N$ of them on each round! We can imagine having a "hyperexpert" for each subset $S \subset [N]$, with $|S| = k$, of which there are clearly $\binom{N}{k}$. Let $\mathcal{S}_k^N$ be the set of all k-sized subsets of $[N]$. On round t, each "base" expert i suffers loss ℓ_i^t which implies that the hyperexpert S suffers loss

$$\text{loss}_t(S) := \sum_{i \in S} \ell_i^t,$$

that is, the hyperexpert loss is additive across the base experts in the subset. Our aim is to run the Exponential Weights Algorithm on these subset hyperexperts. With this well-defined loss on each hyperexpert and a given parameter η, we can define the weight update $w_S^{t+1} = w_S^t \exp(-\eta \text{loss}_t(S))$.

In many scenarios in which we are dealing with hyperexperts, it's suitable to compute the "projected" weights for each base expert i. That is, assume we can run our algorithm by simply knowing $u_i^t := \sum_{S \in \mathcal{S}_k^N : i \in S} w_S^t$. In other words, if our weights define a distribution over $\mathcal{S}_k^N$, then the value u_i^t corresponds to the (unnormalized) marginal probability of i being in a randomly drawn subset. Can we obtain these values efficiently? If so, how?

Hint: Given a vector of positive values $v_{1:n} := \langle v_1, \ldots, v_n \rangle$, we can define

$$\text{SumProd}(v_{1:n}, k) := \textstyle\sum_{S \in \mathcal{S}_k^n} \prod_{i \in S} v_i$$

Naively this requires $O(n^k)$ computation, but perhaps there is a faster way to compute SumProd?