

Online Learning

So far: we assumed we could see the future (e.g., scheduling, caching).

What if we can't see the future? This is called an *online* setting, not like the internet, but as if the input is waiting *on line*.

An Online Problem

1. The input arrives “one piece at a time.”
2. An algorithm makes an irrevocable decision each time it receives a new piece of the input.

Online Decision Making

Choose from expert predictions every time step. An adversary decides who predicts well/poorly in response to your strategy.

Online Decision-Making

At each time step $t = 1, 2, \dots, T$:

a decision-maker picks a probability distribution \mathbf{p}^t over her experts or actions $i = 1, \dots, N$

an adversary picks a **loss** vector $\ell^t : A \rightarrow [0, 1]$

an action i^t is chosen according to the distribution \mathbf{p}^t , and the decision-maker receives loss ℓ_i^t

the decision-maker learns ℓ^t , the entire **loss** vector

The input arrives “one piece at a time.”

What should we compare to?

Thus far, we've been trying to achieve optimal solutions, or comparing to optimal solutions assuming we know full information about the future and what is optimal. Does that still make sense?

Example 1 (Comparing to the Best Action Sequence). Suppose your set of experts (or actions) is $A = \{1, 2\}$. Each day t , the adversary chooses the loss vector ℓ^t as follows: if the algorithm chooses a distribution \mathbf{p}^t for which the probability on action 1 is at least $\frac{1}{2}$, then ℓ^t is set to the vector $(1, 0)$. Otherwise, the adversary sets ℓ^t equal to $(0, 1)$.

This adversary forces the expected cumulative loss of the algorithm to be at least $T/2$, while ensuring that the loss of the best action sequence in hindsight is 0. Thus, the algorithm's approximation is $0/T/2$ —no approximation at all.

Example 1 tells us that we should not be trying to compare to the Best Action Sequence—this is too strong of a goal. Instead, we compare it to the loss incurred by the best *fixed action* in hindsight. In words, we change our benchmark from

$$\sum_{i=1}^T \min_{i=1}^N \ell_i^t \quad \text{to} \quad \min_{i=1}^N \sum_{t=1}^T \ell_i^t.$$

Definition 1 (Regret). Fix loss¹ vectors ℓ^1, \dots, ℓ^T . The regret of the action sequence a^1, \dots, a^T is

$$\underbrace{\sum_{t=1}^T \ell^t(a^t)}_{\text{our algorithm}} - \underbrace{\min_{i=1}^N \sum_{t=1}^T \ell_i^t}_{\text{best fixed action}}. \quad (1)$$

Specifically, our goal is to minimize *regret*.

¹Note that for the rewards setting, the definition of regret would instead be $\max_{i=1}^N \sum_{t=1}^T r_i^t - \sum_{t=1}^T r^t(a^t)$, still minimizing the difference between the algorithm and the best fixed action, but now the maximum reward for the best fixed action will be larger than the algorithm instead of the minimum loss being smaller.

Example 2 (Randomization Is Necessary for No Regret). Fix a deterministic online decision-making algorithm. At each time step t , the algorithm commits to a single action a_t . The obvious strategy for the adversary is to set the loss of action a_t to 1, and the loss of every other action to 0. Then, the cumulative loss of the algorithm is

while the cumulative loss of the best action in hindsight is at most

So the regret of any deterministic algorithm is at least

Even when there are only 2 actions, for arbitrarily large T , the worst-case regret of the algorithm is at least

Example 3 (Regret Lower Bound). Suppose there are $n = 2$ actions, and that we choose each loss vector ℓ^t independently and equally likely to be $(0, 1)$ or $(1, 0)$. No matter how smart or dumb an online decision-making algorithm is, with respect to this random choice of loss vectors, its expected loss at each time step is exactly

and its expected cumulative loss is thus

The expected cumulative loss of the best fixed action in hindsight is

This follows from the fact that if a fair coin is flipped T times, then the expected number of heads is $\frac{T}{2}$ and the standard deviation is $\frac{1}{2}\sqrt{T}$.

What if there are n actions? A similar argument shows that, with n actions, the expected regret of an online decision-making algorithm cannot grow more slowly than $b\sqrt{T \ln n}$, where $b > 0$ is some constant independent of n and T .

Multiplicative Weight Update

Formally, we want an algorithm that works in the following framework:

1. In rounds $1, \dots, T$, the algorithm chooses some expert i^t .
2. Each expert i experiences a loss $\ell_i^t \in [0, 1]$. The algorithm experiences the loss of the expert it chooses: $\ell_A^t = \ell_{i^t}^t$.
3. The total loss of expert i is $L_i^T = \sum_{t=1}^T \ell_i^t$, and the total loss of the algorithm is $\mathbb{E}_{\mathbf{p}} [L_{MWU}] = \mathbb{E}_{\mathbf{p}} \left[\sum_{t=1}^T \ell_{i^t}^t \right]$. The goal of the algorithm is to obtain loss not much worse than that of the best expert: $\min_i L_i^T$.

Multiplicative Weights (MW) Algorithm

initialize weights $w_i^1 = 1$ for every expert $i = 1 \dots, N$

for each time step $t = 1, 2, \dots, T$ **do**

 let $W^t = \sum_{i=1}^N w_k^t$ be the sum of the weights

 choose expert k with probability $p_k^t = w_k^t / W^t$

for each expert k , update weights

$$w_k^{t+1} = w_k^t \cdot (1 - \varepsilon \ell_k^t)$$

Theorem 1. For any sequence of losses, over the randomness of our algorithmic choices \mathbf{p} ,

$$\mathbb{E}_{\mathbf{p}}[\text{REGRET}_{\text{MWU}}] \leq 2\sqrt{\ln(N)T} + \varepsilon T.$$

That is, for any expert k

$$\frac{1}{T} \mathbb{E}_{\mathbf{p}} \left[\sum_{t=1}^T \ell_{i^t}^t \right] \leq \frac{1}{T} \left[\sum_{t=1}^T \ell_k^t \right] + \varepsilon + \frac{\ln(N)}{\varepsilon \cdot T}$$

In particular, by setting $\varepsilon = \sqrt{\frac{\ln(N)}{T}}$ we get:

$$\frac{1}{T} \mathbb{E}_{\mathbf{p}} \left[\sum_{t=1}^T \ell_{i^t}^t \right] - \frac{1}{T} \left[\min_{i=1}^N \sum_{t=1}^T \ell_i^t \right] \leq \varepsilon + 2\sqrt{\frac{\ln(N)}{T}}.$$

In other words, the average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$. Note that this works against an arbitrary sequence of losses, which might be chosen adaptively by an adversary. This is pretty incredible. And it will be the source of the power of this framework in applications: we (the algorithm designer) can play the role of the adversary to get the results that we want.

Corollary 2. There is an online decision-making algorithm that, for every adversary and $\varepsilon > 0$, has expected time-averaged regret² at most ε after at most $(4 \ln n)/\varepsilon^2$ time steps.

Recap of notation:

- N : the number of experts (actions)
- i, k : index of a specific expert (action)
- w : weights assigned to experts, a vector for each expert, indexed for each time step t and expert i
- W^t : the sum over all experts of weights at time t — $W^t = \sum_{i=1}^N w_i^t$.
- p : a probability distribution over experts, indexed for each time step t and expert i , equal to weights w normalized by the sum W — $p_i^t = w_i^t/W^t$.
- ε : update parameter
- ℓ : adversary's loss assignments for each time step and expert, $\ell_i^t \in [0, 1]$.
- F : expected loss. $F^t = \sum_{i=1}^N p_i^t \ell_i^t$.

²Time-averaged regret just means the regret, divided by T .

Proof. Let F^t denote the expected loss of the MWU algorithm at time t . By linearity of expectation, we have $\mathbb{E}[L_{MWU}^T] = \sum_{t=1}^T F^t$. We also know that:

$$F^t = \tag{2}$$

Thus we want to lower bound the sum of the F^t 's.

How does W^t change between rounds? We know that $W^1 = N$, and looking at the algorithm, we derive W^{t+1} as a function of W^t and the expected loss (2)

$$W^{t+1} =$$

So by induction, we can write:

$$W^{T+1} =$$

Taking the log, and using the fact that $\ln(1 - x) \leq -x$, we can write:

$$\ln(W^{T+1}) =$$

Similarly, we can unroll the update rule for our weights

$$w_k^{t+1} = \tag{3}$$

(using the fact that $\ln(1 - x) \geq -x - x^2$ for $0 < x < \frac{1}{2}$), we know that for every expert k :

$$\ln(W^{T+1})$$

Combining these two bounds, we get that for all k :

for all k . Dividing by ε and rearranging, we get:

□