25.1 Introduction

Today we're going to talk about machine learning, but from an algorithms point of view. Machine learning is a pretty broad and interdisciplinary field, including ideas from AI, statistics, and theoretical computer science. All of these are important ingredients, but we're going to focus on machine learning from a TCS point of view. This means in particular that we're going to focus on provable guarantees and we'll try to make our analyses as worst-case as possible, as opposed to more AI or stats-based points of view where the focus is on making probabilistic assumptions and then designing algorithms that seem to work well (either provably or in practice).

25.2 Online Learning

Suppose that we want to predict whether the stock market will go up or down tomorrow. There are many experts who will tell us what they think will happen, but obviously they might disagree with each other. We want to use their advice to take some action (e.g., buy or sell a stock). Then the next day we will find out what happens, and we receive some loss based on our action. Then each expert will make another prediction for the day after, we take some action, and the process repeats (either forever or for some number of rounds).

What is a principled way to approach the problem of deciding which experts to follow, and what kinds of guarantees can we obtain? The basic question that we'll consider is whether we can do nearly as well as the best expert in hindsight. In other words, we will look at how well each expert does over all time (in our simplified setting, the number of mistakes that they make), and we will try to design an algorithm which does nearly as well as the best of them. Note that this seems extremely difficult to do – we don't know anything about these n experts, and the adversary can control what happens at every time point? So one expert might be good for a while and then become terrible, and another expert might start off terrible but get better, etc. Nevertheless, we'll want to guarantee that we do almost as well as the *best* single expert.

25.2.1 Perfect expert

Let's start with a simpler setting: suppose that we know one of the experts is perfect (never makes a mistake), we just don't know which one. Is there a strategy which will let us identify this always-correct expert without making too many mistakes?

Let's do the following: each day, we take a majority vote of the experts and use that as our prediction. Then we eliminate any experts that were wrong. It is easy to see that this algorithm will only make at most $\log N$ mistakes. This is because every time it makes a mistake, that means that at least half of the remaining experts made the incorrect prediction, so we will eliminate at least half of the remaining experts. Since we started with N experts, we can only eliminate at least

Algorithm 1 The Halving Algorithm

Let $\mathcal{E}^1 = \{1, \ldots, N\}$ be the set of experts
for $t = 1, \ldots, T$ **do**
  Let $\mathcal{E}_U^t \subseteq \mathcal{E}^t$ be the subset of experts that predict "up" at time t
  Let $\mathcal{E}_D^t \subseteq \mathcal{E}^t$ be the subset of experts that predict "down" at time t
  if $|\mathcal{E}_U^t| > |\mathcal{E}_D^t|$ **then**
    Predict "buy"
  else
    Predict "sell"
  end if
  See outcome
  Remove all experts that made a mistake. That is, if the stock goes up, $\mathcal{E}^{t+1} \leftarrow \mathcal{E}^t \setminus \mathcal{E}_D^t$ and if the stock goes down, $\mathcal{E}^{t+1} \leftarrow \mathcal{E}^t \setminus \mathcal{E}_U^t$
end for

half of the remaining experts $\log N$ times before we are left with one expert. And since we are assuming that there is a perfect expert, this one will never be eliminated.

This gives a "mistake bound" of $\log N$.

25.2.2 No perfect expert

If there is no perfect expert then making a mistake doesn't completely disqualify an expert, so we don't want to completely eliminate experts who make a mistake. Instead, we'll give each expert a "weight", and decrease the weight of experts who make mistakes.

This idea gives the *Weighted Majority* algorithm:

Algorithm 2 Weighted Majority

For $i \in [N]$, let $w_i^1 = 1$ be the weight of expert i at time 1.
for $t = 1, \ldots, T$ **do**
  Let $W_U^t = \sum_{\substack{i \in [N]: \\ \text{expert } i \text{ predicted "up"}}} w_i^t$ be the weight of all experts that predict "up" at time t
  Let $W_D^t = \sum_{\substack{i \in [N]: \\ \text{expert } i \text{ predicted "down"}}} w_i^t$ be the weight of all experts that predict "down" at time t
  if $W_U^t > W_D^t$ **then**
    Predict "buy"
  else
    Predict "sell"
  end if
  See outcome
  If the stock goes up, $w_i^{t+1} \leftarrow w_i^t/2$ for all experts i that predicted "down" at time t.
  If the stock goes down, $w_i^{t+1} \leftarrow w_i^t/2$ for all experts i that predicted "up" at time t.
end for

To analyze this, let M be the number of mistakes that we have made so far, let m be the number of mistakes that the best expert has made so far, and let W be the total weight (which starts at n and decreases throughout the algorithm). When we make a mistake, that means that at least half of the current weight gets decreased by half, so W drops by at least 25%. So after we've made M mistakes, W is at most $N(3/4)^m$. On the other hand, the weight of the best expert is exactly $(1/2)^m$. Thus

$$\begin{aligned}(1/2)^m &\leq N(3/4)^M \\ (4/3)^M &\leq N2^m \\ M &\leq \log_{4/3}(N2^m) = (m + \log N)/\log(4/3) \approx 2.4(m + \log N)\end{aligned}$$

This is a pretty good result: the number of mistakes that we make is linear (with a reasonably small constant) in the number of mistakes made by the best expert. But what if the best expert is still wrong 20% of the time? Then our algorithm is only doing slightly better than random guessing! (Even if we ignore the $\log N$ part, a mistake bound of $2.4m$ would mean that we make mistakes 48% of the time).

It turns out that we can do even better, and also handle more general settings. Let's define some notation and a framework for online learning that generalize the stock market example.

- T: the number of time steps (e.g., days)
- N: the number of actions the algorithm can take (e.g., the number of experts in the stock market example)
- At every time step $t \in [T]$, the algorithm A chooses an action $i \in [N]$
- Each action $i \in [N]$ then receives a loss $\ell_i^t \in [0, 1]$, and the algorithm receives loss $\ell_A^t = \ell_i^t$ corresponding to the action i that it chose at time t

What is the objective of a learning algorithm in this more general setting? There's no longer a natural notion of "mistake". There are just actions the algorithm can pick from and losses associated with those actions at every timestep. So, rather than comparing the number of mistakes made by an algorithm to that of the best expert in hindsight, we will compare the loss suffered by our algorithm to that of the best fixed action it could have taken in hindsight.

Definition 25.2.1 (Regret) *For all $t \in [T]$, let ℓ_A^t be the loss suffered by algorithm A at time t. Then the* regret *of algorithm A is*

$$\textit{Regret}(A) = \frac{1}{T}\sum_{t=1}^{T} \ell_A^t - \frac{1}{T}\min_{i\in[N]} \sum_{t=1}^{T} \ell_i^t$$

We will now introduce the *Multiplicative Weights Algorithm* (MWA), which you can think of as a generalization of the weighted majority algorithm. As in the weighted majority algorithm we will downweight actions that perform poorly, but it's not clear how to take a weighted vote of actions in our more general setting! Instead, we will treat the weights over actions as a distribution, and our algorithm will choose an action at random according to that distribution.

Algorithm 3 Multiplicative Weights (MW)

For $i \in [N]$, let $w_i^1 = 1$ be the weight of action i at time 1.
for $t = 1, \ldots, T$ **do**
  Let $W^t = \sum_{i \in [N]} w_i^t$ be the total weight at time t.
  Choose action $i \in [N]$ at random according to the distribution $D(i) = \frac{w_i^t}{W^t}$
  See loss ℓ_i^t for action i at time t
  Update weights: $w_j^{t+1} \leftarrow w_j^t \cdot e^{-\varepsilon \ell_j^t}$ for all $j \in [N]$
end for

Theorem 25.2.2 *MW has an expected regret of* $O(\varepsilon + \frac{\log(N)}{\varepsilon T})$. *That is,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\ell_A^t] - \frac{1}{T}\min_{i \in [N]} \sum_{t=1}^{T} \ell_i^t \in O\left(\varepsilon + \frac{\log(N)}{\varepsilon T}\right)$$

We will analyze MW much like the weighted majority algorithm – we will use the sum of weights at time t to relate the loss of the best fixed action to the loss of the sequence of actions taken by the algorithm. At an intuitive level, in any round where the algorithm does very poorly, we expect the weight to drop significantly. At the same time, if any action does very well over all rounds, the total weight can't have dropped too much, since the total weight is always bounded below by the weight on the best action.

Proof: Using the fact that $e^{-x} \leq 1 - x + x^2$ for $x > 0$, and $1 + x \leq e^x$, we can bound the decrease in the total weights in terms of the expected loss of the algorithm as follows:

$$\begin{aligned}
W_{t+1} &= \sum_{i \in [N]} w_i^{t+1} && \text{by definition of } W_{t+1} \\
&= \sum_{i \in [N]} w_i^t \cdot e^{-\varepsilon \ell_i^t} && \text{by definition of } w_i^{t+1} \\
&\leq \sum_{i \in [N]} w_i^t \cdot (1 - \varepsilon \ell_i^t + \varepsilon^2 \ell_i^{t\,2}) && e^{-x} \leq 1 - x + x^2 \text{ for } x > 0 \\
&\leq \sum_{i \in [N]} w_i^t \cdot (1 - \varepsilon \ell_i^t + \varepsilon^2) && \ell_i^t \in [0, 1] \\
&= (1 + \varepsilon^2) \sum_{i \in [N]} w_i^t - \varepsilon \sum_{i \in [N]} w_i^t \ell_i^t \\
&= (1 + \varepsilon^2) W_t - \varepsilon W_t \sum_{i \in [N]} \tfrac{w_i^t}{W_t} \ell_i^t && \text{by definition of } W_t \\
&= W_t (1 + \varepsilon^2 - \varepsilon \sum_{i \in [N]} \tfrac{w_i^t}{W_t} \ell_i^t) \\
&\leq W_t \exp(\varepsilon^2 - \varepsilon \sum_{i \in [N]} \tfrac{w_i^t}{W_t} \ell_i^t) && 1 + x \leq e^x
\end{aligned}$$

$$= W_t \exp(\varepsilon^2 - \varepsilon \mathbb{E}[\ell_A^t]) \qquad \text{by definition of } \mathbb{E}[\ell_A^t]$$

where the last equality follows from the fact that the algorithm takes action i, and therefore suffers loss ℓ_i^t, with probability $\frac{w_i^t}{W_t}$.

This gives us a lower bound on how much the total weight decreases from one round of the algorithm to the next as a function of the expected loss, but recall that the regret of an algorithm relates the loss over *all rounds* to the loss of the best action in hindsight. So, we'll want to unroll this inequality over all T rounds of the algorithm.

$$\begin{aligned} W_{T+1} &\leq W_1 \prod_{t \in T} \exp(\varepsilon^2 - \varepsilon \mathbb{E}[\ell_A^t]) \\ &= W_1 \exp(\varepsilon^2 T - \varepsilon \sum_{t \in [T]} \mathbb{E}[\ell_A^t]) \\ &= N \exp(\varepsilon^2 T - \varepsilon \sum_{t \in [T]} \mathbb{E}[\ell_A^t]) \end{aligned}$$

where the last line follows from the initialization of $w_i^1 = 1$ for all $i \in [N]$.

Great! Now we have an upper bound on W_{T+1} in terms of the expected loss of the algorithm. Next, we want a lower bound on W_{T+1} in terms of the loss of the best fixed action in hindsight. This will allow us to bound regret by showing that the expected loss of the algorithm can't be too much larger than that of the best fixed action. This lower bound follows straightforwardly from the fact that the total weight is always at least the weight of the best fixed action. So we get

$$W_t \geq \sum_{i \in [N]} w_i^t = \exp(-\varepsilon \sum_{t \in [T]} \ell_i^t),$$

for any $i \in [N]$.

Putting our lower and upper bounds together, we can conclude that for any action $i \in [N]$ (including the best fixed action), the total weight is upper and lower bounded as follows:

$$\exp(-\varepsilon \sum_{t \in [T]} \ell_i^t) = w_i^t \leq W_T \leq N \exp(\varepsilon^2 T - \varepsilon \sum_{t \in [T]} \mathbb{E}[\ell_A^t])$$

We ultimately want to relate the sums of losses to eachother to bound regret, so let's take logs of both sides of the inequality above to get

$$-\varepsilon \sum_{t \in [T]} \ell_i^t \leq \ln N + \varepsilon^2 T - \varepsilon \sum_{t \in [T]} \mathbb{E}[\ell_A^t].$$

We then rearrange terms and divide through by ε to get

$$\sum_{t \in [T]} \mathbb{E}[\ell_A^t] - \sum_{t \in [T]} \ell_i^T \leq \frac{\ln N}{\varepsilon} + \varepsilon T$$

Now the left-hand side of the inequality is starting to look like our definition of regret! But we defined regret to be the difference of the average losses, so to finish the proof, we divide through by the total number of rounds T to get

$$\frac{1}{T}\sum_{t\in[T]}\mathbb{E}[\ell_A^t] - \frac{1}{T}\min_{i\in[N]}\sum_{t\in[T]}\ell_i^t \leq \frac{\ln N}{\varepsilon T} + \varepsilon$$

■

Then if we take $\varepsilon = \sqrt{\frac{\ln N}{T}}$, we get that the expected regret of MW is

$$\frac{1}{T}\sum_{t\in[T]}\mathbb{E}[\ell_A^t] - \frac{1}{T}\min_{i\in[N]}\sum_{t\in[T]}\ell_i^t \leq 2\sqrt{\frac{\ln N}{T}}.$$

This demonstrates that Multiplicative Weights is what is known as a "no-regret" algorithm: its expected regret goes to 0 as $T \to \infty$. Next time, we will see that Multiplicative Weights (and no-regret algorithms more generally) actually have a wide variety of applications beyond online learning, and in fact can be used to solve linear programming problems we discussed in previous lectures!