UPenn NETS 412: Algorithmic Game Theory January 30, 2018

Lecture 5

Lecturer: Bo Waggoner *Scribe: Aaron Roth, Bo Waggoner*

The Polynomial Weights Algorithm

Last time, we introduced the problem of **learning from expert advice** in a repeated setting. We saw the *iterated halving* algorithm, which makes at most $\log(N)\,(OPT+1)$ mistakes, where N is the number of experts and OPT is the number of mistakes of the best expert.

We should be able to do better, though. The iterated halving algorithm is wasteful in that every time we reset, we forget what we have learned! The weighted majority algorithm can be viewed as a "softer" version of the halving algorithm: rather than eliminating experts who make mistakes, we just down-weight them:

Algorithm 1 The Weighted Majority Algorithm

```
Set weights $w_i^1 \leftarrow 1$ for all experts $i$.
for $t = 1$ to $T$ do
    Let $W_U^t = \sum_{i:p_i^t=U} w_i$ be the weight of experts who predict up, and $W_D^t = \sum_{i:p_i^t=D} w_i$ be the weight of those who predict down.
    Predict with the weighted majority vote: If $W_U^t > W_D^t$, predict $p_A^t = U$, else predict $p_A^t = D$.
    Down-weight experts who made mistakes: For all $i$ such that $p_i^t \neq o^t$, set $w_i^{t+1} \leftarrow w_i^t/2$
end for
```

Theorem 1 *The weighted majority algorithm makes at most* $2.4\,(OPT+\log(N))$ *mistakes.*

Note that $\log(N)$ is a fixed constant, so the ratio of mistakes the algorithm makes compared to OPT is just 2.4 in the limit as OPT grows larger – not great, but not bad.

Proof Let M be the total number of mistakes that the algorithm makes, and let $W^t = \sum_i w_i^t$ be the total weight at step t. Note that on any round t in which the algorithm makes a mistake, at least half of the total weight (corresponding to experts who made mistakes) is cut in half, and so $W^{t+1} \leq (3/4)W^t$. Hence, we know that if the algorithm makes M mistakes, we have $W^T \leq N \cdot (3/4)^M$. Let i^* be the best expert. We also know that $w_{i^*}^T = (1/2)^{\text{OPT}}$, and so in particular, $W^T > (1/2)^{\text{OPT}}$. Combining these two observations we know:

$$\left(\frac{1}{2}\right)^{\text{OPT}} \leq W \leq N\left(\frac{3}{4}\right)^M$$

$$\left(\frac{4}{3}\right)^M \leq N \cdot 2^{\text{OPT}}$$

$$M \leq 2.4(\text{OPT} + \log(N))$$

as claimed. ■

We've been doing well; lets get greedy. What do we want in an algorithm? We might want:

1. It to make only 1 times as many mistakes as the best expert in the limit, rather than 2.4 times...
2. It to be able to handle N distinct actions (a separate action for each expert), not just two (up and down)...
3. It to be able to handle experts having arbitrary costs in $[0,1]$ at each round, not just binary costs (right vs. wrong).

Formally, we want an algorithm that works in the following **online learning** framework, which generalizes learning from expert advice:

1. There are N *actions.*
2. In each round $t = 1, \dots, T$, the algorithm chooses some *action* i^t.
3. Each action i experiences a loss $\ell_i^t \in [0,1]$. The algorithm experiences the loss of the action it chooses: $\ell_A^t = \ell_{i^t}^t$.
4. The total loss of action i is $L_i = \sum_{t=1}^T \ell_i^t$, and the total loss of the algorithm is $L_A = \sum_{t=1}^T \ell_A^t$. The goal of the algorithm is to obtain loss not much worse than that of always choosing the best action: $\mathrm{OPT} := \min_i L_i$.

In the previous setting of learning from expert advice, each action corresponded to following the advice of a particular expert. The loss was 1 if that expert made a mistake in that round, and 0 otherwise.

Example 1 *There are N different routes to class. Each day, you must select a route $i^t \in \{1, \dots, N\}$. Then, you find out how long each route i would have taken, which is the "loss" ℓ_i^t. This loss is between* 0 *and* 1 *hours. Your total loss L_A is the total number of hours you spent commuting over the course of the semester. For each route i, its total loss L_i is the total number of hours you would have spent if you had just taken the same route i every day.*

To measure the algorithm's performance, we will use *regret.*

Definition 2 *In the online learning setting, the **regret** of an algorithm is the difference between its loss and that of the best action in hindsight:*

$$\textit{regret} := L_A \ - \ OPT.$$

*The **average regret** is the regret "per time step":*

$$\textit{average regret} := \frac{\textit{regret}}{T}.$$

Corollary 3 *The regret of Weighted Majority is at most* $1.4OPT + 2.4\log(N)$.

This is pretty good, but we might want to do better. The problem is, as time grows ($T \to \infty$), we would normally expect that OPT is also growing, and the regret – the gap between OPT and the algorithm – is growing even faster.

To improve, we will study the polynomial weights algorithm. The polynomial weights algorithm can be viewed as a further smoothed version of the weighted majority algorithm, and has a parameter ϵ which controls how quickly it down-weights actions. Notably, it is *randomized*: rather than making deterministic decisions, it randomly chooses an action with probability proportional to its weight.

Algorithm 2 The Polynomial Weights Algorithm (PW)

```
Set weights $w_i^1 \leftarrow 1$ for all actions $i$.
for $t = 1$ to $T$ do
  Let $W^t = \sum_{i=1}^N w_i^t$.
  Choose action $i$ with probability $w_i^t / W^t$.
  For each $i$, set $w_i^{t+1} \leftarrow w_i^t \cdot (1 - \epsilon \ell_i^t)$.
end for
```

Theorem 4 *For any sequence of losses, for any choice of $0 < \epsilon < \frac{1}{2}$, the PW Algorithm satisfies*

$$\mathbb{E}[\textit{average regret}] \leq \epsilon + \frac{\ln(N)}{\epsilon \cdot T}.$$

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

$$\mathbb{E}[\textit{average regret}] \leq 2\sqrt{\frac{\ln(N)}{T}}.$$

(Here the expectation is over the random choices of the algorithm.)

In other words, the average loss of the algorithm quickly approaches the average loss of the best action 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. In particular, we will see later that it means we can use the polynomial weights algorithm to successfully play in a repeated game.

Ok, on to the proof. First, let us state a useful fact: For $0 \leq c \leq \frac{1}{2}$, we have

$$e^{-c-c^2} \leq 1 - c \leq e^{-c}.$$

(You may wish to plot the three functions to visualize this.)

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

$$F^t = \frac{\sum_{i=1}^N w_i^t \ell_i^t}{W^t}$$

How does W^t change between rounds? We know that $W^1 = N$, and looking at the algorithm we see:

$$W^{t+1} = W^t - \sum_{i=1}^N \epsilon w_i^t \ell_i^t = W^t(1 - \epsilon F^t)$$

So by induction, we can write:

$$W^{T+1} = N \prod_{t=1}^T (1 - \epsilon F^t)$$

Now using our useful fact, namely that $(1 - \epsilon F^t) \leq e^{-\epsilon F^t}$, we get:

$$\begin{aligned} W^{T+1} &\leq N \prod_{t=1}^T e^{-\epsilon F^t} \\ &= N e^{-\epsilon \sum_{t=1}^T F^t} \\ &= N e^{-\epsilon \mathbb{E}[L_A]}. \end{aligned}$$

On the other hand, the total weight is at least the weight of the best expert (call her k):

$$\begin{aligned} W^{T+1} &\geq w_k^{T+1} \\ &= \prod_{t=1}^{T}(1-\epsilon \ell_k^t) \\ &\geq \prod_{t=1}^{T} e^{-\epsilon \ell_k^t - \epsilon^2 (\ell_k^t)^2} && \text{(by our useful fact)} \\ &\geq \prod_{t=1}^{T} e^{-\epsilon \ell_k^t - \epsilon^2} && \text{(because } (\ell_k^t)^2 \leq 1\text{)} \\ &\geq e^{-\epsilon \sum_{t=1}^{T} \ell_k^t \ - \ T\epsilon^2} \\ &= e^{-\epsilon L_k^T - T\epsilon^2}. \end{aligned}$$

Combining these inequalities of W^{T+1} and taking the log of both sides, we get

$$-\epsilon L_k - T\epsilon^2 \leq \ln(N) - \epsilon\,\mathbb{E}[L_A].$$

Rearranging and dividing by $T\epsilon$:

$$\frac{\mathbb{E}[L_A] - L_k}{T} \leq \epsilon + \frac{\ln(N)}{\epsilon \cdot T}.$$

■