CIS 320: Algorithms October 27 and November 1, 2021

Lecture 15-16

Lecturer: Aaron Roth *Scribe: Aaron Roth*

Polynomial Weights I and II

In the next two lectures, we will develop algorithms for *prediction* problems that must be solved in an adversarial, sequential setting. These algorithms operate in an environment that we haven't studied before in this class: rather than having a problem instance fully described up front, that we must solve, we will have an environment in which our algorithm must interact dynamically. The algorithms we derive will be interesting in their own right, and are fundamental building blocks in machine learning, but as we will see, they are also useful and powerful tools for solving other algorithmic problems in a more standard setting.

As a simple example to keep in mind, consider the following toy model of predicting the stock market: every day the market goes *up* or *down*, and you must predict what it will do before it happens (so that you can either buy or short shares). You don't have any information about what the market will do, and it may behave arbitrarily, so you can't hope to do well in an absolute sense. However, every day, before you make your prediction, you get to hear the advice of a bunch of experts, who make their own predictions. These "experts" may or may not know what they are talking about, and you start off knowing nothing about them. Nevertheless, you want to come up with a rule to aggregate their advice so that you end up doing (almost) as well as the best expert (whomever he might turn out to be) in hindsight. Sounds tough.

Lets start with an even easier case:

- There are N experts who will make predictions in T rounds.
- At each round t, each expert i makes a prediction $p_i^t \in \{U, D\}$ (up or down).
- We (the algorithm) aggregate these predictions somehow, to make our own prediction $p_A^t \in \{U, D\}$. Then we learn the true outcome $o^t \in \{U, D\}$. If we predicted incorrectly (i.e. $p_A^t \neq o^t$), then we *made a mistake*.
- To make things easy, we will assume at first that there is one *perfect* expert who never makes a mistake (but we don't know who he is).

Can we find a strategy that is guaranteed to make at most $\log(N)$ mistakes?

We can, using the simple halving algorithm!

Algorithm 1 The Halving Algorithm

Let $S^1 \leftarrow \{1, \ldots, N\}$ be the set of all experts.
for $t = 1$ to T **do**
  Let $S_U^t = \{i \in S : p_i^t = U\}$ be the set of experts in S^t who predict up, and $S_D^t = S^t \setminus S_U^t$ be the set who predict down.
  Predict with the majority vote: If $|S_U^t| > |S_D^t|$, predict $p_A^t = U$, else predict $p_A^t = D$.
  Eliminate all experts that made a mistake: If $o^T = U$, then let $S^{t+1} = S_U^t$, else let $S^{t+1} = S_D^t$
end for

Its not hard to see that the halving algorithm makes at most $\log N$ mistakes under the assumption that one expert is perfect:

Theorem 1 *If there is at least one perfect expert, the halving algorithm makes at most* $\log N$ *mistakes.*

Proof Since the algorithm predicts with the majority vote, every time it makes a mistake at some round t, at least half of the remaining experts have made a mistake and are eliminated, and hence $|S^{t+1}| \leq |S^t|/2$. On the other hand, the perfect expert is never eliminated, and hence $|S^t| \geq 1$ for all t. Since $|S^1| = N$, this means there can be at most $\log N$ mistakes. ■

Not bad – $\log N$ is pretty small even if N is large (e.g. if $N = 1024$, $\log N = 10$, if $N = 1,048,576$, $\log N = 20$), and doesn't grow with T, so even with a huge number of experts, the average number of mistakes made by this algorithm is tiny.

What if no expert is perfect? Suppose the best expert makes OPT mistakes. Can we find a way to make not too many more than OPT mistakes?

The first approach you might try is the iterated halving algorithm:

Algorithm 2 The Iterated Halving Algorithm

Let $S^1 \leftarrow \{1, \ldots, N\}$ be the set of all experts.
for $t = 1$ to T **do**
  If $|S^t| = 0$ **Reset**: Set $S^t \leftarrow \{1, \ldots, N\}$.
  Let $S^t_U = \{i \in S : p^t_i = U\}$ be the set of experts in S^t who predict up, and $S^t_D = S^t \setminus S^t_U$ be the set who predict down.
  Predict with the majority vote: If $|S^t_U| > |S^t_D|$, predict $p^t_A = U$, else predict $p^t_A = D$.
  Eliminate all experts that made a mistake: If $o^T = U$, then let $S^{t+1} = S^t_U$, else let $S^{t+1} = S^t_D$
end for

Theorem 2 *The iterated halving algorithm makes at most* $\log(N)(OPT + 1)$ *mistakes.*

Proof As before, whenever the algorithm makes a mistake, we eliminate half of the experts, and so the algorithm can make at most $\log N$ mistakes between any two resets. But if we reset, it is because since the last reset, *every* expert has made a mistake: in particular, between any two resets, the *best* expert has made at least 1 mistake. This gives the claimed bound. ■

We should be able to do better though. The above 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 3 The Weighted Majority Algorithm

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

Theorem 3 *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 – not great, but not bad.

Proof Let M be the total number of mistakes that the algorithm makes, and let $W^t = \sum_i w^t_i$ 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 framework:

1. In rounds $1, \ldots, T$, 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 $L_A^T = \sum_{t=1}^T \ell_A^t$. The goal of the algorithm is to obtain loss not much worse than that of the best expert: $\min_i L_i^T$.

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 experts. Notably, it is *randomized*: rather than making deterministic decisions, it randomly chooses an expert to follow with probability proportional to their weight.

Algorithm 4 The Polynomial Weights Algorithm (PW)

Set weights $w_i^1 \leftarrow 1$ for all experts i.
for $t = 1$ to T **do**
  Let $W^t = \sum_{i=1}^N w_i^t$.
  Choose expert 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, and any expert k:*

$$\frac{1}{T}\mathrm{E}[L_{PW}^T] \leq \frac{1}{T}L_k^T + \epsilon + \frac{\ln(N)}{\epsilon \cdot T}$$

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

$$\frac{1}{T}\mathrm{E}[L_{PW}^T] \leq \frac{1}{T}\min_k L_k^T + 2\sqrt{\frac{\ln(N)}{T}}$$

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.

Ok, on to the proof:

Proof Let F^t denote the expected loss of the polynomial weights algorithm at time t. By linearity of expectation, we have $\mathrm{E}[L_{PW}^T] = \sum_{t=1}^T F^t$. 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:

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

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

$$\begin{aligned} \ln(W^{t+1}) &= \ln(N) + \sum_{t=1}^T \ln(1 - \epsilon F^t) \\ &\leq \ln(N) - \epsilon \sum_{t=1}^T F^t \\ &= \ln(N) - \epsilon \mathrm{E}[L_{PW}^T] \end{aligned}$$

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

$$\begin{aligned} \ln(W^{T+1}) &\geq \ln(w_k^{T+1}) \\ &= \sum_{t=1}^T \ln(1 - \epsilon \ell_k^t) \\ &\geq -\sum_{t=1}^T \epsilon \ell_k^t - \sum_{t=1}^T (\epsilon \ell_k^t)^2 \\ &\geq -\epsilon L_k^T - \epsilon^2 T \end{aligned}$$

Combining these two bounds, we get:

$$\ln(N) - \epsilon L_{PW}^T \geq -\epsilon L_k^T - \epsilon^2 T$$

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

$$L_{PW}^T \leq \min_k L_k^T + \epsilon T + \frac{\ln(N)}{\epsilon}$$

■

One final observation: we have described the algorithm so far as if it is *randomly* selecting some action i at each round, and have been measuring its expected loss at each round: $\sum_{i=1}^{N} \frac{w_i^t}{W^t} \ell_i^t$. This makes sense if the algorithm must choose an expert to play at every round. But in some settings, it makes sense for the algorithm to play a *vector* in $\Delta[n] = \{p \in [0,1]^N : \sum_{i=1}^{n} p_i = 1\}$ at every round. For example, it might be interacting in the following setting, called online adversarial linear optimization:

1. In rounds $1, \ldots, T$ the algorithm chooses a vector $w^t \in \Delta[N]$.
2. The adversary chooses a loss vector $\ell^t \in [0,1]^N$.
3. The algorithm experiences loss $\ell_A^t = \langle w^t, \ell^t \rangle$.
4. The goal of the algorithm is to guarantee that:

$$\frac{1}{T}\sum_{t=1}^{T} \langle w^t, \ell^t \rangle \leq \min_{w^* \in \Delta[n]} \frac{1}{T}\sum_{t=1}^{T} \langle w^*, \ell^t \rangle + o(1).$$

In this case, we can view the exact same algorithm we have derived and analyzed as a deterministic algorithm for choosing such a vector — at round t is plays the vector $w^t = \{\frac{w_i^t}{W^t}\}_{i=1}^n$.

Algorithm 5 The Polynomial Weights Algorithm for Online Linear Optimization

Set weights $w_i^1 \leftarrow 1$ for all experts i.
for $t = 1$ to T **do**
  Let $W^t = \sum_{i=1}^{N} w_i^t$.
  Play vector $w^t = \{\frac{w_i^t}{W^t}\}_{i=1}^n$
  For each i, set $w_i^{t+1} \leftarrow w_i^t \cdot (1 - \epsilon \ell_i^t)$.
end for

Our existing analysis proves the following theorem:

Theorem 5 *Setting $\epsilon = \sqrt{\frac{\ln(N)}{T}}$, for any sequence of losses $\ell^t \in [0,1]^N$:*

$$\frac{1}{T}\sum_{t=1}^{T} \langle w^t, \ell^t \rangle \leq \min_{w^* \in \Delta[n]} \frac{1}{T}\sum_{t=1}^{T} \langle w^*, \ell^t \rangle + 2\sqrt{\frac{\ln(N)}{T}}$$

Proof The left hand side is exactly the expected loss of the polynomial weights algorithm we analyzed in the experts setting. Continuing the translation, the "loss of expert i" corresponds to $\frac{1}{T}\sum_{t=1}^{T} \langle e_i, \ell^t \rangle$, where e_i is the i'th standard basis vector (with a 1 in the i'th coordinate and a 0 in every other coordinate). Finally observe that we always have that for every sequence of losses:

$$\min_{w^* \in \Delta[n]} \sum_{t=1}^{T} \langle w^*, \ell^t \rangle = \sum_{t=1}^{T} \langle e_{i^*}, \ell^t \rangle$$

where $i^* = \arg\min_{i \in [N]} \sum_{t=1}^{T} \ell_i^t$. Hence regret to the best basis vector e_{i^*} (i.e. the best expert) implies regret to the best vector $w^* \in \Delta[N]$. ■

Finally, we observe that there if we are using polynomial weights for online linear optimization, there is no reason to restrict attention to vectors w^* whose coordinates sum to 1, or losses that lie in the range $[0,1]$. We simply have to pay for the scale of the vectors we are optimizing over. Lets see how we could use the polynomial weights algorithm to solve the online linear optimization problem over the set of

non-negative vectors w that sum to at most R_1: $B_N(R_1) = \{w \in \mathbb{R}^N_{\geq 0} : \sum_{i=1}^n \leq R_1\}$, for loss functions that take values in the range $\ell_i^t \in [-R_2/2, R_2/2]$.

First lets deal with the issue of having coordinates of w that sum to *at most* some value R_1 rather than exactly R_1. We can simply add an extra $N+1$'st coordinate that always has loss $\ell^t_{N+1} = 0$. Running our algorithm in this augmented $N+1$ dimensional space means that if the $N+1$ dimensional vector w^t has coordinates summing to exactly R_1 at every round, the first N coordinates of w (the "real ones") sum to at most R_1 — and the algorithm experiences the same loss as if it played in only the real N dimensional space.

Next lets deal with the issue of negative losses. This is also easy: simply shift them by adding $R_2/2$ to every coordinate. Now we have $\ell_i^t \in [0, R_2]$, and note that the regret to any target w^* remains unchanged under this shift, because:

$$\langle w^t, \ell^t + R_2/2\rangle - \langle w^*, \ell^t + R_2/2\rangle = \big(\langle w^t, \ell^t\rangle - \langle w^*, \ell^t\rangle\big) + \big(\langle R_2/2, \ell^t\rangle - \langle R_2/2, \ell^t\rangle\big) = \langle w^t, \ell^t\rangle - \langle w^*, \ell^t\rangle$$

So regret bounds for the shifted space hold also for the original losses.

We're almost done. We simply have to scale down everything, apply our bounds, and then remember to scale back up. Suppose we divide the coordinates of w^t by R_1 and the coordinates of ℓ^t by R_2. We are now in the setting for which we have proven the regret bound for the polynomial weights algorithm, and so we have that the polynomial weights algorithm can obtain the regret bound:

$$\frac{1}{T}\sum_{t=1}^T \langle \frac{w^t}{R_1}, \frac{\ell^t}{R_2}\rangle \leq \min_{w^* \in \Delta[n]} \frac{1}{T}\sum_{t=1}^T \langle \frac{w^*}{R_1}, \frac{\ell^t}{R_2}\rangle + 2\sqrt{\frac{\ln(N)}{T}}$$

Multiplying this bound through by $R_1 \cdot R_2$ we obtain:

Theorem 6 *For any sequence of losses $\ell^t \in [-R_2/2, R_2/2]^N$, the polynomial weights algorithm can be used to play vectors $w^t \in B_N(R_1)$ and obtain:*

$$\frac{1}{T}\sum_{t=1}^T \langle w^t, \ell^t\rangle \leq \min_{w^* \in B_N(R_1)} \frac{1}{T}\sum_{t=1}^T \langle w^*, \ell^t\rangle + 2R_1R_2\sqrt{\frac{\ln(N)}{T}}$$