NETS 412: Algorithmic Game Theory	January 26 and 31, 2016

Lecture 4 and 5

Lecturer: Aaron Roth *Scribe: Aaron Roth*

The Polynomial Weights Algorithm

In the last several lectures, we have seen several games in which best response dynamics inevitably converges to a Nash equilibrium; however, there are also plenty of games (we saw an example in class) for which best response dynamics does not converge, *even if they have pure strategy Nash equilibria.* In such games, what should players do? What should we expect to happen?

In this lecture, we will give a natural learning algorithm that players can use to play a game. To introduce it, we abstract away the game, and the other players, and start by asking how a player should make predictions in a sequential 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

$|S^{t+1}| \le |S^t|/2$. On the other hand, the perfect expert is never eliminated, and hence $|S^t| \ge 1$ 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 $S^1 \leftarrow \{1, \dots, N\}$ be the set of all experts.
for $t = 1$ to T **do**
  If $|S^t| = 0$ **Reset**: Set $S^t \leftarrow \{1, \dots, N\}$.
  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

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 $w_i^1 \leftarrow 1$ 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 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 $W^t = \sum_i w_i^t$ 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 $W^{t+1} \le (3/4)W^t$. Hence, we know that if the algorithm makes M mistakes, we have $W^T \le N \cdot (3/4)^M$. 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:*

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

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

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

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. In particular, it means we can use the polynomial weights algorithm to play a game! We simply let each of the "experts" correspond to an action in the game, and let the losses of the experts correspond to our costs in the game, given what the other players did. The guarantee is that no matter what they do (even if they are trying explicitly to cause us high loss), we are guaranteed to obtain payoff nearly as high as that of the best action in hindsight! In fact, to obtain this guarantee, we don't even need to know the payoff structure of the game we are playing – to run the algorithm, all we need are the realized costs of each action, given what our opponents ended up doing.

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 $\mathrm{E}[L^T_{PW}] = \sum_{t=1}^T F^t$. 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:

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

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:

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

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

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

■