The Polynomial Weights Algorithm

Aaron Roth

University of Pennsylvania

February 9 2021

Overview

- We've started studying sequential learning...

Overview

- We've started studying sequential learning...
- As predicting from expert advice.

Overview

- We've started studying sequential learning...
- As predicting from expert advice.
- We made progress under a big assumption: A Perfect Expert.

Overview

- We've started studying sequential learning...
- As predicting from expert advice.
- We made progress under a big assumption: A Perfect Expert.
- What do we do without that assumption?

Review

The Setting:

- There are N experts who will make predictions in T rounds.

Review

The Setting:

- 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).

Review

The Setting:

- 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*.

Review

The Setting:

- 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*.
- Easy Case: there is one *perfect* expert who never makes a mistake (but we don't know who he is).

The 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

The Halving Algorithm

Theorem

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

The Halving Algorithm

Theorem

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

Proof.

1. The algorithm predicts with the majority vote, so every time it makes a mistake at some round t, at least half of the remaining experts have made a mistake and are eliminated.

□

The Halving Algorithm

Theorem

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

Proof.

1. The algorithm predicts with the majority vote, so every time it makes a mistake at some round t, at least half of the remaining experts have made a mistake and are eliminated.
2. Hence $|S^{t+1}| \leq |S^t|/2$.

□

The Halving Algorithm

Theorem

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

Proof.

1. The algorithm predicts with the majority vote, so every time it makes a mistake at some round t, at least half of the remaining experts have made a mistake and are eliminated.
2. Hence $|S^{t+1}| \leq |S^t|/2$.
3. On the other hand, the perfect expert is never eliminated.

□

The Halving Algorithm

Theorem

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

Proof.

1. The algorithm predicts with the majority vote, so every time it makes a mistake at some round t, at least half of the remaining experts have made a mistake and are eliminated.
2. Hence $|S^{t+1}| \leq |S^t|/2$.
3. On the other hand, the perfect expert is never eliminated.
4. Hence $|S^t| \geq 1$ for all t.

□

The Halving Algorithm

Theorem

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

Proof.

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

$\square$

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

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT}+1)$ *mistakes.*

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT}+1)$ *mistakes.*

Proof.

1. Whenever the algorithm makes a mistake, we eliminate half of the experts.

□

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT}+1)$ *mistakes.*

Proof.

1. Whenever the algorithm makes a mistake, we eliminate half of the experts.
2. So the algorithm can make at most $\log N$ mistakes between any two resets.

□

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT}+1)$ *mistakes.*

Proof.

1. Whenever the algorithm makes a mistake, we eliminate half of the experts.
2. So the algorithm can make at most $\log N$ mistakes between any two resets.
3. But if we reset, it is because since the last reset, *every* expert has made a mistake.

□

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT}+1)$ *mistakes.*

Proof.

1. Whenever the algorithm makes a mistake, we eliminate half of the experts.
2. So the algorithm can make at most $\log N$ mistakes between any two resets.
3. But if we reset, it is because since the last reset, *every* expert has made a mistake.
4. in particular, between any two resets, the *best* expert has made at least 1 mistake.

□

The Iterated Halving Algorithm

Theorem

The iterated halving algorithm makes at most $\log(N)(\mathrm{OPT} + 1)$ *mistakes.*

Proof.

1. Whenever the algorithm makes a mistake, we eliminate half of the experts.
2. So the algorithm can make at most $\log N$ mistakes between any two resets.
3. But if we reset, it is because since the last reset, *every* expert has made a mistake.
4. in particular, between any two resets, the *best* expert has made at least 1 mistake.
5. This gives the claimed bound.

□

Review

1. We should be able to do better though.

Review

1. We should be able to do better though.
2. The above algorithm is wasteful in that every time we reset, we forget what we have learned!

Review

1. We should be able to do better though.
2. The above algorithm is wasteful in that every time we reset, we forget what we have learned!
3. What should we do instead?

Review

1. We should be able to do better though.
2. The above algorithm is wasteful in that every time we reset, we forget what we have learned!
3. What should we do instead?
4. How about just *downweight* experts who make mistakes?

The Weighted Majority Algorithm

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

The Weighted Majority Algorithm

Theorem

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

The Weighted Majority Algorithm

Theorem

The weighted majority algorithm makes at most $2.4\,(\mathrm{OPT} + \log(N))$ *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

1. Let M be the total number of mistakes that the algorithm makes.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.
3. Observe: 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.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.
3. Observe: 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.
4. So: $W^{t+1} \leq (3/4)W^t$.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.
3. Observe: 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.
4. So: $W^{t+1} \leq (3/4)W^t$.
5. If the algorithm makes M mistakes, $W^T \leq N \cdot (3/4)^M$.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.
3. Observe: 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.
4. So: $W^{t+1} \leq (3/4)W^t$.
5. If the algorithm makes M mistakes, $W^T \leq N \cdot (3/4)^M$.
6. Let i^* be the best expert. $W^T > w_i^T = (1/2)^{\mathrm{OPT}}$.

Proof

1. Let M be the total number of mistakes that the algorithm makes.
2. Let $W^t = \sum_i w_i^t$ be the total weight at step t.
3. Observe: 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.
4. So: $W^{t+1} \leq (3/4)W^t$.
5. If the algorithm makes M mistakes, $W^T \leq N \cdot (3/4)^M$.
6. Let i^* be the best expert. $W^T > w_i^T = (1/2)^{\mathrm{OPT}}$.
7. Together we have:

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

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

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

Getting Greedy

We've been doing well! What do we want in an algorithm?

1. It to make only 1 times as many mistakes as the best expert in the limit, rather than 2.4 times...

Getting Greedy

We've been doing well! What do we want in an algorithm?

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)...

Getting Greedy

We've been doing well! What do we want in an algorithm?

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)

Getting Greedy

We want an algorithm that works in the following framework:

1. In rounds $1, \ldots, T$, the algorithm chooses some expert i^t.

Getting Greedy

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: $\ell_A^t = \ell_{i^t}^t$.

Getting Greedy

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 $L_A^T = \sum_{t=1}^{T} \ell_A^t$.

Getting Greedy

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

Getting Greedy

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 $L_A^T = \sum_{t=1}^T \ell_A^t$.
4. 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 "smoothed" version of the weighted majority algorithm

Getting Greedy

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 $L_A^T = \sum_{t=1}^T \ell_A^t$.
4. 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 "smoothed" version of the weighted majority algorithm

1. Has a parameter ϵ which controls how quickly it down-weights experts.

Getting Greedy

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 $L_A^T = \sum_{t=1}^T \ell_A^t$.
4. 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 "smoothed" version of the weighted majority algorithm

1. Has a parameter ϵ which controls how quickly it down-weights experts.
2. Is *randomized* — chooses which expert to follow with probability proportional to its weight.

The Polynomial Weights Algorithm

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

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

1. The average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

1. The average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.
2. This works against an *arbitrary* sequence of losses, which might be chosen adaptively by an adversary.

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

1. The average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.
2. This works against an *arbitrary* sequence of losses, which might be chosen adaptively by an adversary.
3. So could ues it to play a game!

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

1. The average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.
2. This works against an *arbitrary* sequence of losses, which might be chosen adaptively by an adversary.
3. So could ues it to play a game!
4. Experts $\leftrightarrow$ Actions. Losses $\leftrightarrow$ costs.

The Polynomial Weights Algorithm

Theorem

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* $\epsilon = \sqrt{\frac{\ln(N)}{T}}$*:*

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

1. The average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.
2. This works against an *arbitrary* sequence of losses, which might be chosen adaptively by an adversary.
3. So could ues it to play a game!
4. Experts $\leftrightarrow$ Actions. Losses $\leftrightarrow$ costs.
5. Don't need to know much about the game. Just costs for each action given what the opponents did.

Proof

1. Let F^t denote the expected loss of the algorithm at time t.

Proof

1. Let F^t denote the expected loss of the algorithm at time t.
2. $\mathrm{E}[L^T_{PW}] = \sum_{t=1}^{T} F^t$.

Proof

1. Let F^t denote the expected loss of the algorithm at time t.
2. $\mathrm{E}[L^T_{PW}] = \sum_{t=1}^{T} F^t$.
3. We also know:

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

Proof

1. Let F^t denote the expected loss of the algorithm at time t.
2. $\mathrm{E}[L_{PW}^T] = \sum_{t=1}^T F^t$.
3. We also know:

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

4. $W^1 = N$, and:

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

Proof

1. Let F^t denote the expected loss of the algorithm at time t.
2. $\mathrm{E}[L^T_{PW}] = \sum_{t=1}^{T} F^t$.
3. We also know:
$$F^t = \frac{\sum_{i=1}^{N} w_i^t \ell_i^t}{W^t}$$
4. $W^1 = N$, and:
$$W^{t+1} = W^t - \sum_{i=1}^{N} \epsilon w_i^t \ell_i^t = W^t(1 - \epsilon F^t)$$
5. So by induction:
$$W^{T+1} = N \prod_{t=1}^{T} (1 - \epsilon F^t)$$

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

Proof

1. Taking the log, and using $\ln(1-x) \le -x$:

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

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

2. Similarly, using $\ln(1-x) \geq -x - x^2$ for $0 < x < \frac{1}{2}$:

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

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

2. Similarly, using $\ln(1-x) \geq -x - x^2$ for $0 < x < \frac{1}{2}$:

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

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

2. Similarly, using $\ln(1-x) \geq -x - x^2$ for $0 < x < \frac{1}{2}$:

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

Proof

1. Taking the log, and using $\ln(1-x) \leq -x$:

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

2. Similarly, using $\ln(1-x) \geq -x - x^2$ for $0 < x < \frac{1}{2}$:

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

Proof

1. Combining these two bounds, we get:

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

   for all k.

Proof

1. Combining these two bounds, we get:

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

   for all k.

2. Dividing by ϵ and rearranging:

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

Proof

1. Combining these two bounds, we get:

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

   for all k.

2. Dividing by ϵ and rearranging:

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

3. Fin.

Thanks!

See you next class — stay healthy, and wear a mask!