Algorithmic game theory

Martin Balko

6th lecture

November 11th 2025

Regret minimization

Regret minimization

Regret minimization

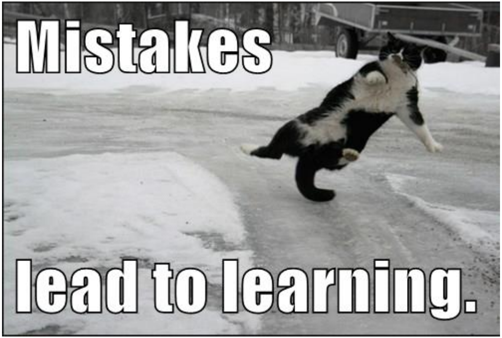
- We introduce a completely new model of interactions based on so-called regret minimization.

Regret minimization

- We introduce a completely new model of interactions based on so-called regret minimization. We apply online learning.

Regret minimization

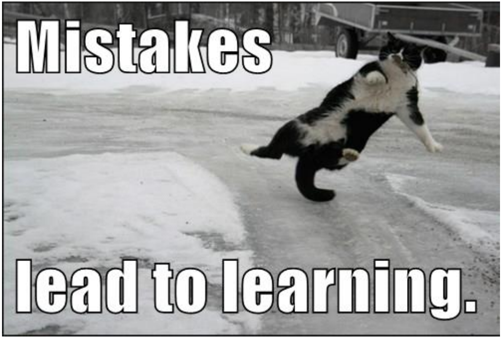
- We introduce a completely new model of interactions based on so-called regret minimization. We apply online learning.



Sources: https://blogger.googleusercontent.com/

Regret minimization

- We introduce a completely new model of interactions based on so-called regret minimization. We apply online learning.

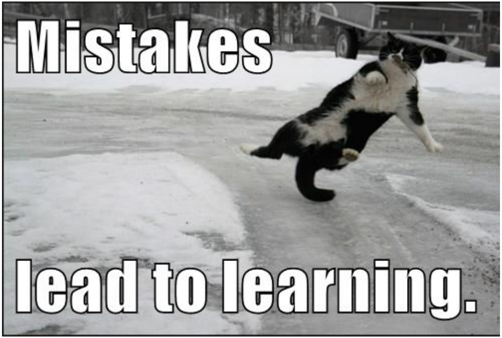


Sources: https://blogger.googleusercontent.com/

- Later, we apply these new methods to design new fast algorithms to approximate correlated equilibria.

Regret minimization

- We introduce a completely new model of interactions based on so-called regret minimization. We apply online learning.



Sources: https://blogger.googleusercontent.com/

- Later, we apply these new methods to design new fast algorithms to approximate correlated equilibria.
- Today, we introduce the model and some basic algorithms on how to minimize regret.

The setting

The setting

- Since we are introducing a new model, we will need some notation.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.
- At each step $t = 1, \ldots, T$:

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \dots, N\}$.

- At each step $t = 1, \dots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \dots, p_N^t)$ over X,

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
  - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$,

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
  - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [-1, 1]$ is the loss of action i in step t.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
  - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [-1, 1]$ is the loss of action i in step t.
  - The agent A then experiences loss $\ell_A^t = \sum_{i=1}^N p_i^t \ell_i^t$.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
    - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
    - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [-1, 1]$ is the loss of action i in step t.
    - The agent A then experiences loss $\ell_A^t = \sum_{i=1}^N p_i^t \ell_i^t$. This is the expected loss of A in step t.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
  - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [-1, 1]$ is the loss of action i in step t.
  - The agent A then experiences loss $\ell_A^t = \sum_{i=1}^N p_i^t \ell_i^t$. This is the expected loss of A in step t.

- After T steps, the cumulative loss of action i is $L_i^T = \sum_{t=1}^T \ell_i^t$.

The setting

- Since we are introducing a new model, we will need some notation.
- We have an agent A in an adversary environment.
- There are N available actions for A in the set $X = \{1, \ldots, N\}$.

- At each step $t = 1, \ldots, T$:
  - Our agent A selects a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over X, where p_i^t is the probability that A selects i in step t.
  - Then, the adversary chooses a loss vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [-1, 1]$ is the loss of action i in step t.
  - The agent A then experiences loss $\ell_A^t = \sum_{i=1}^N p_i^t \ell_i^t$. This is the expected loss of A in step t.

- After T steps, the cumulative loss of action i is $L_i^T = \sum_{t=1}^T \ell_i^t$.
- The cumulative loss of A is $L_A^T = \sum_{t=1}^T \ell_A^t$.

External regret

External regret

- We need to be able to tell how well is our agent doing.

External regret

- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.

External regret

- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.
- We will mostly consider the class $\mathcal{A}_X = \{A_i \colon i \in X\}$, where an agent A_i always chooses action i.

External regret

- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.
- We will mostly consider the class $\mathcal{A}_X = \{A_i \colon i \in X\}$, where an agent A_i always chooses action i.
- Let $R_A^T = L_A^T - \min\{L_B^T \colon B \in \mathcal{A}_X\}$ be the external regret of A (with respect to $\mathcal{A}_X$).

External regret

- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.
- We will mostly consider the class $\mathcal{A}_X = \{A_i \colon i \in X\}$, where an agent A_i always chooses action i.
- Let $R_A^T = L_A^T - \min\{L_B^T \colon B \in \mathcal{A}_X\}$ be the external regret of A (with respect to $\mathcal{A}_X$). That is, $R_A^T = L_A^T - \min\{L_i^T : i \in X\}$

External regret

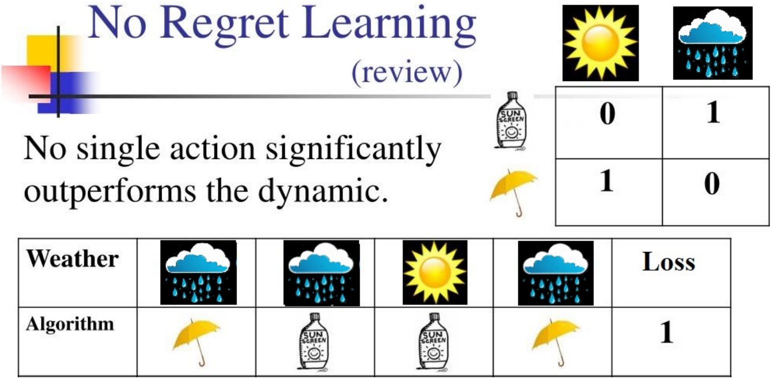
- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.
- We will mostly consider the class $\mathcal{A}_X = \{A_i : i \in X\}$, where an agent A_i always chooses action i.
- Let $R_A^T = L_A^T - \min\{L_B^T : B \in \mathcal{A}_X\}$ be the external regret of A (with respect to $\mathcal{A}_X$). That is, $R_A^T = L_A^T - \min\{L_i^T : i \in X\}$
- Until specified otherwise, we consider only loss vectors from $\{0, 1\}^N$.

External regret

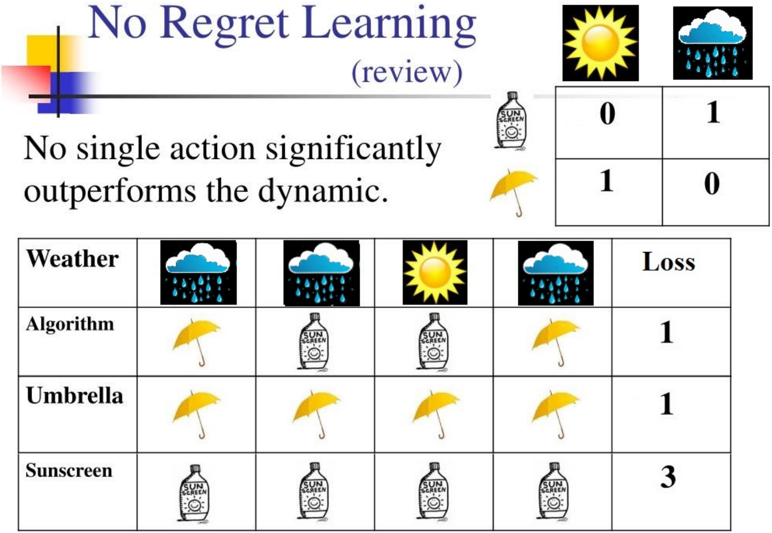
- We need to be able to tell how well is our agent doing. We choose an "external approach" and compare his loss to the loss of the best agent from some comparison class $\mathcal{A}$.
- We will mostly consider the class $\mathcal{A}_X = \{A_i : i \in X\}$, where an agent A_i always chooses action i.
- Let $R_A^T = L_A^T - \min\{L_B^T : B \in \mathcal{A}_X\}$ be the external regret of A (with respect to $\mathcal{A}_X$). That is, $R_A^T = L_A^T - \min\{L_i^T : i \in X\}$
- Until specified otherwise, we consider only loss vectors from $\{0,1\}^N$. This is only to simplify the notation, all presented results can be extended to the general case.

Example

Example


No Regret Learning
(review)
SUN SCREEN
No single action significantly outperforms the dynamic.
0
1
1
0
Weather
Loss
Algorithm
1

Example


No Regret Learning
(review)
No single action significantly outperforms the dynamic.
0
1
1
0
Weather
Loss
Algorithm
1
Umbrella
1
Sunscreen
3

Is the setting too restrictive?

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof:

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
    - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$ and $\ell_i^t = 1$ for every $i \neq i_t$.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$ and $\ell_i^t = 1$ for every $i \neq i_t$.
- Since $p_{i_t}^t \leq 1/N$, we have $\ell_A^t \geq 1 - 1/N$

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$ and $\ell_i^t = 1$ for every $i \neq i_t$.
- Since $p_{i_t}^t \leq 1/N$, we have $\ell_A^t \geq 1 - 1/N$ and thus the cumulative loss L_A^T of A after T steps is at least $T(1 - 1/N)$.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
  - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$ and $\ell_i^t = 1$ for every $i \neq i_t$.
- Since $p_{i_t}^t \leq 1/N$, we have $\ell_A^t \geq 1 - 1/N$ and thus the cumulative loss L_A^T of A after T steps is at least $T(1 - 1/N)$.
- The algorithm $B \in \mathcal{A}_{all}$ that selects the action i_t in step t with probability 1 has the cumulative loss $L_B^T = 0$.

Is the setting too restrictive?

- It might seem that the class $\mathcal{A}_X$ contains too simple agents. However, we show that large comparison classes lead to a very large regret.
- Let $\mathcal{A}_{all}$ be the set of agents that assign probability 1 to an arbitrary action from X in every step.
    - In $\mathcal{A}_X$ each agent has to select the same action in all steps.

Observation 2.45

For any agent A and every $T \in \mathbb{N}$, there is a sequence of T loss vectors and an agent $B \in \mathcal{A}_{all}$ such that $L_A^T - L_B^T \geq T(1 - 1/N)$.

- That is almost as bad as it can get.
- Proof: For every t, let i_t be the action with the lowest probability p_i^t. We set $\ell_{i_t}^t = 0$ and $\ell_i^t = 1$ for every $i \neq i_t$.
- Since $p_{i_t}^t \leq 1/N$, we have $\ell_A^t \geq 1 - 1/N$ and thus the cumulative loss L_A^T of A after T steps is at least $T(1 - 1/N)$.
- The algorithm $B \in \mathcal{A}_{all}$ that selects the action i_t in step t with probability 1 has the cumulative loss $L_B^T = 0$. □

Greedy algorithm

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$.

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach:

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.6: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.7: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1, 0, \ldots, 0)$,

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.8: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1, 0, \ldots, 0)$,
for $t = 2, \ldots, T$
do $\{$

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.9: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \dots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \dots, T\}$.
$p^1 \leftarrow (1, 0, \dots, 0)$,
for $t = 2, \dots, T$
do $\left\{ L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \right.$

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.10: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \dots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \dots, T\}$.

$p^1 \leftarrow (1, 0, \dots, 0)$,

for $t = 2, \dots, T$

do $\begin{cases} L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \\ S^{t-1} \leftarrow \{i \in X \colon L_i^{t-1} = L_{min}^{t-1}\}, \\ k \leftarrow \min S^{t-1}, \end{cases}$

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.11: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1, 0, \ldots, 0)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \\ S^{t-1} \leftarrow \{i \in X : L_i^{t-1} = L_{min}^{t-1}\}, \\ k \leftarrow \min S^{t-1}, \\ p_k^t \leftarrow 1, p_i^t \leftarrow 0 \text{ for } i \neq k, \end{cases}$$

Greedy algorithm

- So we are good with the comparison class $\mathcal{A}_X$. How to design an agent A that performs well against agents from $\mathcal{A}_X$?
- We first try a natural greedy approach: select an action $i \in X$ for which the cumulative loss L_i^{t-1} at step $t-1$ is the smallest.

Algorithm 0.12: GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1, 0, \ldots, 0)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \\ S^{t-1} \leftarrow \{i \in X \colon L_i^{t-1} = L_{min}^{t-1}\}, \\ k \leftarrow \min S^{t-1}, \\ p_k^t \leftarrow 1, p_i^t \leftarrow 0 \text{ for } i \neq k, \end{cases}$$

Output $\{p^t \colon t \in \{1, \ldots, T\}\}$.

Analysis of the Greedy algorithm

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0, 1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{Greedy}} \le N \cdot L^T_{min} + (N - 1).$$

- Proof:

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step.

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1.

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

$$L^T_{\text{Greedy}}$$

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

$$L^T_{\text{Greedy}} \leq N \cdot L^T_{min} + N - |S^T|$$

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + N - |S^T| \leq N \cdot L^T_{min} + (N-1).$$

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0, 1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + (N - 1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + N - |S^T| \leq N \cdot L^T_{min} + (N - 1).$$

□

Analysis of the Greedy algorithm

Proposition 2.46

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{Greedy} of the Greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + (N-1).$$

- Proof: At step t, if the Greedy algorithm incurs a loss of 1 and L^t_{min} does not increase, then at least one action disappears from S^t in the next step. This occurs at most N times and then L^t_{min} increases by 1.
- Thus, the Greedy algorithm incurs a loss of at most N between successive increments of L^t_{min} by 1. It follows that

$$L^T_{\mathrm{Greedy}} \leq N \cdot L^T_{min} + N - |S^T| \leq N \cdot L^T_{min} + (N-1).$$

□

- This is rather weak since A can perform roughly N times worse than the best action.

Randomized Greedy algorithm

Randomized Greedy algorithm

- There is a good reason for the poor behavior.

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness.

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.19: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.20: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \dots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \dots, T\}$.
$p^1 \leftarrow (1/N, \dots, 1/N)$,

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.21: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$
do $\{$

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.22: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$
do $\left\{ L^{t-1}_{min} \leftarrow \min_{j \in X}\{L^{t-1}_j\}, \right.$

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.23: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \\ S^{t-1} \leftarrow \{i \in X : L_i^{t-1} = L_{min}^{t-1}\}, \\ \end{cases}$$

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.24: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} L^{t-1}_{min} \leftarrow \min_{j \in X}\{L^{t-1}_j\}, \\ S^{t-1} \leftarrow \{i \in X \colon L^{t-1}_i = L^{t-1}_{min}\}, \\ p^t_i \leftarrow 1/|S^{t-1}| \text{ for every } i \in S^{t-1} \text{ and } p^t_i \leftarrow 0 \text{ otherwise.} \end{cases}$$

Randomized Greedy algorithm

- There is a good reason for the poor behavior. No deterministic algorithm can perform significantly better (Exercise).
- So it makes sense to introduce some randomness. We break ties at random, splitting weights between the currently best actions.

Algorithm 0.25: RANDOMIZED GREEDY ALGORITHM(X, T)

Input : A set of actions $X = \{1, \ldots, N\}$ and number of steps $T \in \mathbb{N}$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$
do $\begin{cases} L_{min}^{t-1} \leftarrow \min_{j \in X}\{L_j^{t-1}\}, \\ S^{t-1} \leftarrow \{i \in X \colon L_i^{t-1} = L_{min}^{t-1}\}, \\ p_i^t \leftarrow 1/|S^{t-1}| \text{ for every } i \in S^{t-1} \text{ and } p_i^t \leftarrow 0 \text{ otherwise.} \end{cases}$
Output $\{p^t \colon t \in \{1, \ldots, T\}\}$.

Analysis of the Randomized greedy algorithm

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch):

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \le (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L_{RG}^{T} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L_{min}^{t} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L_{RG}^{T} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L_{min}^{t} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n',

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most $1/N + 1/(N-1) + \cdots + 1/1$

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most $1/N + 1/(N-1) + \cdots + 1/1 \leq 1 + \ln N$.

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L^T_{RG} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L^T_{\mathrm{RG}} \leq (1 + \ln N) \cdot L^T_{min} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L^t_{min} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most $1/N + 1/(N-1) + \cdots + 1/1 \leq 1 + \ln N$. It follows that

$$L^T_{\mathrm{RG}}$$

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L_{RG}^{T} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L_{min}^{t} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most $1/N + 1/(N-1) + \cdots + 1/1 \leq 1 + \ln N$. It follows that

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + (1/N + 1/(N-1) + \cdots + 1/(|S^T| + 1)).$$

Analysis of the Randomized greedy algorithm

Proposition 2.48

For any sequence of $\{0,1\}$-valued loss vectors, the cumulative loss L_{RG}^{T} of the Randomized greedy algorithm at time $T \in \mathbb{N}$ satisfies

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + \ln N.$$

- Proof (sketch): We proceed as in the previous proof. For $j \in \mathbb{N}$, let t_j be the time step t at which the loss L_{min}^{t} first reaches value j. We estimate the loss of the algorithm between steps t_j and t_{j+1}.
- Note that $1 \leq |S^t| \leq N$. If the size of S^t shrinks by k from n' to $n' - k$ at some time $t \in (t_j, t_{j+1}]$, then the loss of the algorithm at step t is k/n', since the weight of each such action is $1/n'$.
- Clearly, $k/n' \leq 1/n' + 1/(n'-1) + \cdots + 1/(n'-k+1)$, so we obtain that the loss for the entire time interval $(t_j, t_{j+1}]$ is at most $1/N + 1/(N-1) + \cdots + 1/1 \leq 1 + \ln N$. It follows that

$$L_{\mathrm{RG}}^{T} \leq (1 + \ln N) \cdot L_{min}^{T} + (1/N + 1/(N-1) + \cdots + 1/(|S^T| + 1)).$$

$\square$

Polynomial weights algorithm

Polynomial weights algorithm

- This is better, but still not optimal.

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.30: POLYNOMIAL WEIGHTS ALGORITHM(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.31: Polynomial weights algorithm(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.32: Polynomial weights algorithm(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.33: POLYNOMIAL WEIGHTS ALGORITHM(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$
do $\left\{ \right.$

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.34: POLYNOMIAL WEIGHTS ALGORITHM(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$
do $\left\{ w_i^t \leftarrow w_i^{t-1}(1 - \eta \ell_i^{t-1}), \right.$

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.35: POLYNOMIAL WEIGHTS ALGORITHM(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} w_i^t \leftarrow w_i^{t-1}(1 - \eta \ell_i^{t-1}), \\ W^t \leftarrow \sum_{i \in X} w_i^t, \\ \end{cases}$$

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.36: POLYNOMIAL WEIGHTS ALGORITHM(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} w_i^t \leftarrow w_i^{t-1}(1 - \eta \ell_i^{t-1}), \\ W^t \leftarrow \sum_{i \in X} w_i^t, \\ p_i^t \leftarrow w_i^t / W^t \text{ for every } i \in X. \end{cases}$$

Polynomial weights algorithm

- This is better, but still not optimal. The losses are greatest when the sets S^t are small since the loss can be viewed as proportional to $1/|S^t|$.
- We overcome this issue by assigning larger weights to actions that are close to the best one.

Algorithm 0.37: Polynomial weights algorithm(X, T, η)

Input : A set of actions $X = \{1, \ldots, N\}$, $T \in \mathbb{N}$, and $\eta \in (0, 1/2]$.
Output : A probability distribution p^t for every $t \in \{1, \ldots, T\}$.
$w_i^1 \leftarrow 1$ for every $i \in X$,
$p^1 \leftarrow (1/N, \ldots, 1/N)$,
for $t = 2, \ldots, T$

$$\textbf{do} \begin{cases} w_i^t \leftarrow w_i^{t-1}(1 - \eta \ell_i^{t-1}), \\ W^t \leftarrow \sum_{i \in X} w_i^t, \\ p_i^t \leftarrow w_i^t / W^t \text{ for every } i \in X. \end{cases}$$

Output $\{p^t : t \in \{1, \ldots, T\}\}$.

Analysis of the Polynomial weights algorithm I

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$.

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \le L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \ge 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \le T$, we obtain

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \le L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \ge 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \le T$, we obtain

$$L^T_{\mathrm{PW}} \le L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch):

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \le L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \ge 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \le T$, we obtain

$$L^T_{\mathrm{PW}} \le L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially.

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially. For step t, we have $\ell^t_{PW} = \sum_{i=1}^{N} w^t_i \ell^t_i / W^t$,

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^T (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially. For step t, we have $\ell^t_{PW} = \sum_{i=1}^N w^t_i \ell^t_i / W^t$, that is, ℓ^t_{PW} is the expected loss at step t.

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially. For step t, we have $\ell^t_{PW} = \sum_{i=1}^{N} w^t_i \ell^t_i / W^t$, that is, ℓ^t_{PW} is the expected loss at step t.
- The weight w^t_i of every action i is multiplied by $(1 - \eta \ell^{t-1}_i)$ at step t.

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^{T} (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially. For step t, we have $\ell^t_{PW} = \sum_{i=1}^{N} w^t_i \ell^t_i / W^t$, that is, ℓ^t_{PW} is the expected loss at step t.
- The weight w^t_i of every action i is multiplied by $(1 - \eta \ell^{t-1}_i)$ at step t. Thus, $W^{t+1} = W^t - \sum_{i=1}^{N} \eta w^t_i \ell^t_i$

Analysis of the Polynomial weights algorithm I

Theorem 2.49

For $\eta \in (0, 1/2]$, every sequence of $[-1, 1]$-valued loss vectors, and every $k \in X$, the cumulative loss L^T_{PW} of the Polynomial weights algorithm satisfies

$$L^T_{\mathrm{PW}} \leq L^T_k + \eta Q^T_k + \ln N/\eta,$$

where $Q^T_k = \sum_{t=1}^T (\ell^t_k)^2$. In particular, if $T \geq 4 \ln N$, then by setting $\eta = \sqrt{\ln N/T}$ and noting that $Q^T_k \leq T$, we obtain

$$L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}.$$

- Proof (sketch): We show that if there is a significant loss, then the total weight W^t must drop substantially. For step t, we have $\ell^t_{PW} = \sum_{i=1}^N w^t_i \ell^t_i / W^t$, that is, ℓ^t_{PW} is the expected loss at step t.
- The weight w^t_i of every action i is multiplied by $(1 - \eta \ell^{t-1}_i)$ at step t. Thus, $W^{t+1} = W^t - \sum_{i=1}^N \eta w^t_i \ell^t_i = W^t(1 - \eta \ell^t_{PW})$.

Analysis of the Polynomial weights algorithm II

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1}$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t)$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t}$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1}$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1}$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

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

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1} \geq \ln w_k^{T+1} = \sum_{t=1}^{T} \ln(1 - \eta \ell_k^t)$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1} \geq \ln w_k^{T+1} = \sum_{t=1}^{T} \ln(1 - \eta \ell_k^t) \geq -\eta L_k^T - \eta^2 Q_k^T.$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1} \geq \ln w_k^{T+1} = \sum_{t=1}^{T} \ln(1 - \eta \ell_k^t) \geq -\eta L_k^T - \eta^2 Q_k^T.$$

- Combining the lower and the upper bound, we have

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1 - z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1} \geq \ln w_k^{T+1} = \sum_{t=1}^{T} \ln(1 - \eta \ell_k^t) \geq -\eta L_k^T - \eta^2 Q_k^T.$$

- Combining the lower and the upper bound, we have

$$-\eta L_k^T - \eta^2 Q_k^t \leq \ln N - \eta L_{\mathrm{PW}}^T.$$

Analysis of the Polynomial weights algorithm II

- Using $W^1 = N$ and $1 - z \leq e^{-z}$ for every $z \in \mathbb{R}$, we obtain

$$W^{T+1} = W^1 \prod_{t=1}^{T} (1 - \eta \ell_{PW}^t) \leq N \prod_{t=1}^{T} e^{-\eta \ell_{PW}^t} = N e^{-\eta \sum_{t=1}^{T} \ell_{PW}^t}.$$

- Taking the logarithms, we obtain

$$\ln W^{T+1} \leq \ln N - \eta \sum_{t=1}^{T} \ell_{PW}^t = \ln N - \eta L_{\mathrm{PW}}^T.$$

- For the lower bound, we have $W^{T+1} \geq w_k^{T+1}$ and thus, by taking logarithms, using the recursive definition of weights and $\ln(1-z) \geq -z - z^2$ for $z \leq 1/2$, we obtain

$$\ln W^{T+1} \geq \ln w_k^{T+1} = \sum_{t=1}^{T} \ln(1 - \eta \ell_k^t) \geq -\eta L_k^T - \eta^2 Q_k^T.$$

- Combining the lower and the upper bound, we have

$$-\eta L_k^T - \eta^2 Q_k^t \leq \ln N - \eta L_{\mathrm{PW}}^T.$$

□

Polynomial weights algorithm: remarks

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.
- The bound $L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}$ is essentially optimal.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.
- The bound $L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}$ is essentially optimal.

Proposition 2.50

For integers N and T with $T < \lfloor \log_2 N \rfloor$, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A] \geq T/2$ and yet $L^T_{min} = 0$.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.
- The bound $L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}$ is essentially optimal.

Proposition 2.50

For integers N and T with $T < \lfloor \log_2 N \rfloor$, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A] \geq T/2$ and yet $L^T_{min} = 0$.

Proposition 2.51

In the case of $N = 2$ actions, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A - L^T_{min}] \geq \Omega(\sqrt{T})$.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.
- The bound $L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}$ is essentially optimal.

Proposition 2.50

For integers N and T with $T < \lfloor \log_2 N \rfloor$, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A] \geq T/2$ and yet $L^T_{min} = 0$.

Proposition 2.51

In the case of $N = 2$ actions, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A - L^T_{min}] \geq \Omega(\sqrt{T})$.

- See lecture notes for the proofs.

Polynomial weights algorithm: remarks

- This algorithm produces very good external regret. Time-averaged external regret goes to zero.
- The bound $L^T_{\mathrm{PW}} \leq L^T_{min} + 2\sqrt{T \ln N}$ is essentially optimal.

Proposition 2.50

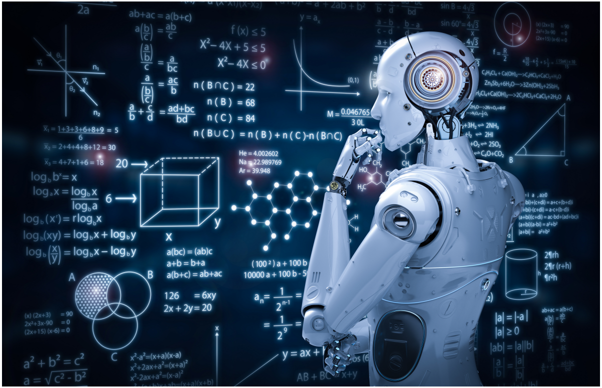
For integers N and T with $T < \lfloor \log_2 N \rfloor$, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A] \geq T/2$ and yet $L^T_{min} = 0$.

Proposition 2.51

In the case of $N = 2$ actions, there exists a stochastic generation of losses such that, for every online algorithm A, we have $\mathbb{E}[L^T_A - L^T_{min}] \geq \Omega(\sqrt{T})$.

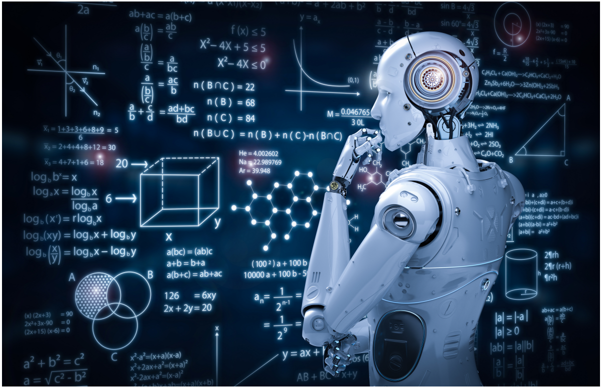
- See lecture notes for the proofs.
- We do not need to know T in advance (Exercise).

- Besides game theory, the “multiplicative weight update method” has many applications in various fields of science, for example in optimization, theoretical computer science, and machine learning.



Sources: https://clubitc.ro

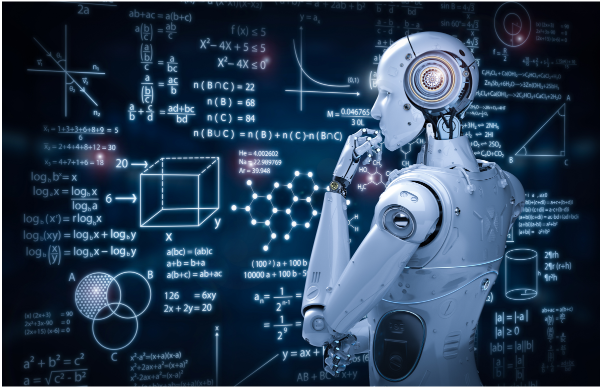
- Besides game theory, the “multiplicative weight update method” has many applications in various fields of science, for example in optimization, theoretical computer science, and machine learning.



Sources: https://clubitc.ro

- See `https://en.wikipedia.org/wiki/Multiplicative_weight_update_method#Applications`

- Besides game theory, the “multiplicative weight update method” has many applications in various fields of science, for example in optimization, theoretical computer science, and machine learning.



Sources: https://clubitc.ro

- See `https://en.wikipedia.org/wiki/Multiplicative_weight_update_method#Applications`
- There are other algorithms producing small external regret, for example, the Regret matching algorithm.

Regret matching

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

- The algorithm works as follows.

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

- The algorithm works as follows. We choose $R_1 = \mathbf{0} \in \mathbb{R}^N$

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

- The algorithm works as follows. We choose $R_1 = \mathbf{0} \in \mathbb{R}^N$ and then let for $t \geq 1$

$$p^t = \frac{[R^t]^+}{\|[R^t]^+\|_1}$$

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

- The algorithm works as follows. We choose $R_1 = \mathbf{0} \in \mathbb{R}^N$ and then let for $t \geq 1$

$$p^t = \frac{[R^t]^+}{\|[R^t]^+\|_1} \quad \text{and} \quad R^{t+1} = R^t + \langle p^t, \ell^t \rangle \cdot \mathbf{1} - \ell^t,$$

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} : i \in [N]).$$

- The algorithm works as follows. We choose $R_1 = \mathbf{0} \in \mathbb{R}^N$ and then let for $t \geq 1$

$$p^t = \frac{[R^t]^+}{\|[R^t]^+\|_1} \quad \text{and} \quad R^{t+1} = R^t + \langle p^t, \ell^t \rangle \cdot \mathbf{1} - \ell^t,$$

where $\mathbf{0}/0$ is defined as the uniform distribution.

Regret matching

- Algorithm with external regret at most $O(\sqrt{T})$ for fixed N.
- Introduced by Hart and Mas-Colell in 2000.
- For a vector $v \in \mathbb{R}^N$, we use $[v]^+$ to denote the vector

$$(\max\{0, v_i\} \colon i \in [N]).$$

- The algorithm works as follows. We choose $R_1 = \mathbf{0} \in \mathbb{R}^N$ and then let for $t \geq 1$

$$p^t = \frac{[R^t]^+}{\|[R^t]^+\|_1} \quad \text{and} \quad R^{t+1} = R^t + \langle p^t, \ell^t \rangle \cdot \mathbf{1} - \ell^t,$$

where $\mathbf{0}/0$ is defined as the uniform distribution.
- Strong empirical performance and admits many variants.

The No-regret dynamics

The No-regret dynamics

- "Players in a normal-form game play against each other by selecting actions according to the Polynomial-weights algorithm."

The No-regret dynamics

- "Players in a normal-form game play against each other by selecting actions according to the Polynomial-weights algorithm."

Algorithm 0.40: NO-REGRET DYNAMICS(G, T, ε)

Input : A normal-form game $G = (P, A, C)$ of n players, $T \in \mathbb{N}$, and $\varepsilon > 0$.
Output : A prob. distribution p_i^t on A_i for each $i \in P$ and $t \in \{1, \ldots, T\}$.
for every step $t = 1, \ldots, T$

do $\begin{cases} \text{Each player } i \in P \text{ independently chooses a mixed strategy } p_i^t \\ \text{using an algorithm with average regret at most } \varepsilon\text{, with actions} \\ \text{corresponding to pure strategies.} \\ \text{Each player } i \in P \text{ receives a loss vector } \ell_i^t = (\ell_i^t(a_i))_{a_i \in A_i}\text{, where} \\ \ell_i^t(a_i) \leftarrow \mathbb{E}_{a_{-i}^t \sim p_{-i}^t}[C_i(a_i; a_{-i}^t)] \text{ for the product distribution} \\ p_{-i}^t = \prod_{j \neq i} p_j^t. \end{cases}$

Output $\{p^t : t \in \{1, \ldots, T\}\}$.

"ENROLL IN AGT" THEY SAID

Algorithm 2.6.4: NO-REGRET DYNAMICS(G, T, ε)

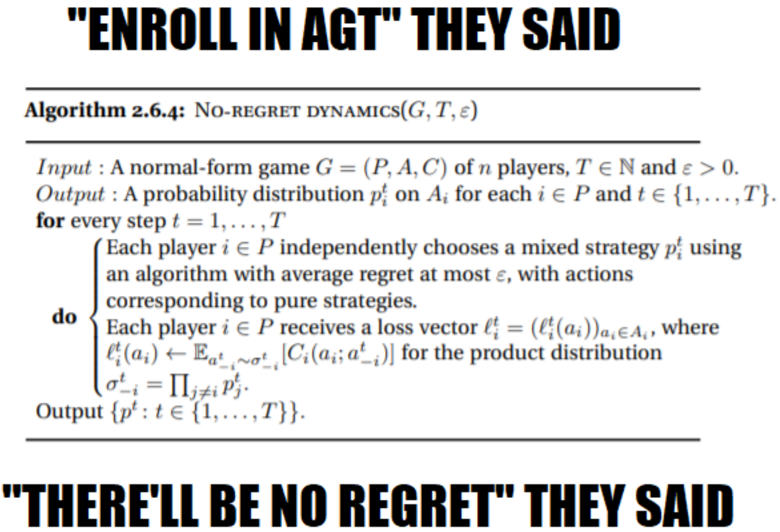
Input : A normal-form game $G = (P, A, C)$ of n players, $T \in \mathbb{N}$ and $\varepsilon > 0$.
Output : A probability distribution p_i^t on A_i for each $i \in P$ and $t \in \{1, \dots, T\}$.
for every step $t = 1, \dots, T$

do {
Each player $i \in P$ independently chooses a mixed strategy p_i^t using an algorithm with average regret at most ε, with actions corresponding to pure strategies.
Each player $i \in P$ receives a loss vector $\ell_i^t = (\ell_i^t(a_i))_{a_i \in A_i}$, where $\ell_i^t(a_i) \leftarrow \mathbb{E}_{a_{-i}^t \sim \sigma_{-i}^t}[C_i(a_i; a_{-i}^t)]$ for the product distribution $\sigma_{-i}^t = \prod_{j \neq i} p_j^t$.

Output $\{p^t : t \in \{1, \dots, T\}\}$.

"THERE'LL BE NO REGRET" THEY SAID

Sources: Students of MFF UK



Sources: Students of MFF UK

Thank you for your attention.