Minimização de Arrependimento

Vinícius Pimentel Couto
Prof. Rafael Schouery

IC/UNICAMP

Dezembro 2014

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t ($\in [0, 1]$)

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t $(\in [0, 1])$
  - ℓ^t = vetor de prejuízo das N ações

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t ($\in [0, 1]$)
  - ℓ^t = vetor de prejuízo das N ações
- $\ell_H^t = \sum_{i=1}^{N} p_i^t \ell_i^t$ = Prejuízo do algoritmo H no passo t

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t ($\in [0, 1]$)
  - ℓ^t = vetor de prejuízo das N ações
- $\ell_H^t = \sum_{i=1}^N p_i^t \ell_i^t$ = Prejuízo do algoritmo H no passo t
- $L_i^T = \sum_{t=1}^T \ell_i^t$ = Prejuízo da ação i nos primeiros T turnos

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t $(\in [0, 1])$
  - ℓ^t = vetor de prejuízo das N ações
- $\ell_H^t = \sum_{i=1}^N p_i^t \ell_i^t$ = Prejuízo do algoritmo H no passo t
- $L_i^T = \sum_{t=1}^T \ell_i^t$ = Prejuízo da ação i nos primeiros T turnos
- $L_H^T = \sum_{t=1}^T \ell_H^t = \sum_{t=1}^T \sum_{i=1}^N p_i^t \ell_i^t$ = Prejuízo do algoritmo H nos primeiros T turnos

Problema e modelo

Problema: decisões repetidas com resultados incertos.

Modelo:

- $X = \{1, ..., N\}$ = Conjunto das N ações disponíveis
- p_i^t = probabilidade da ação i no turno t
  - p^t = distribuição de probabilidade nas N ações no turno t
- ℓ_i^t = prejuízo da ação i no turno t $(\in [0, 1])$
  - ℓ^t = vetor de prejuízo das N ações
- $\ell_H^t = \sum_{i=1}^N p_i^t \ell_i^t$ = Prejuízo do algoritmo H no passo t
- $L_i^T = \sum_{t=1}^T \ell_i^t$ = Prejuízo da ação i nos primeiros T turnos
- $L_H^T = \sum_{t=1}^T \ell_H^t = \sum_{t=1}^T \sum_{i=1}^N p_i^t \ell_i^t$ = Prejuízo do algoritmo H nos primeiros T turnos
- $R = L_H^T - L_G^T$ = Arrependimento (*Regret*)

Problema e modelo

Problema e modelo

Theorem

Para qualquer algoritmo H existe uma sequência de T vetores de prejuízo tais que:

$$R \geq T(1 - 1/N)$$

Problema e modelo

Theorem

Para qualquer algoritmo H existe uma sequência de T vetores de prejuízo tais que:

$$R \geq T(1 - 1/N)$$

PROVA: a cada turno t a ação de menor probabilidade i_t recebe prejuízo 0 enquanto as outras recebem prejuízo 1. Como $min_i\{p_i^t\} \leq 1/N$ o prejuízo a cada turno é pelo menos $1 - 1/N$ e o prejuízo em T turnos é $T(1 - 1/N)$. Por outro lado, existe o algoritmo $g(t) = i_t$ com um prejuízo total de 0.

Problema e modelo

Theorem

Para qualquer algoritmo H existe uma sequência de T vetores de prejuízo tais que:

$$R \geq T(1 - 1/N)$$

PROVA: a cada turno t a ação de menor probabilidade i_t recebe prejuízo 0 enquanto as outras recebem prejuízo 1. Como $min_i\{p_i^t\} \leq 1/N$ o prejuízo a cada turno é pelo menos $1 - 1/N$ e o prejuízo em T turnos é $T(1 - 1/N)$. Por outro lado, existe o algoritmo $g(t) = i_t$ com um prejuízo total de 0.

Solução: comparar somente com algoritmos que jogam sempre uma única ação. Em particular, comparar H com o melhor desses algoritmos. Assim, temos o conceito do *external regret*:

Problema e modelo

Theorem

Para qualquer algoritmo H existe uma sequência de T vetores de prejuízo tais que:

$$R \geq T(1 - 1/N)$$

PROVA: a cada turno t a ação de menor probabilidade i_t recebe prejuízo 0 enquanto as outras recebem prejuízo 1. Como $min_i\{p_i^t\} \leq 1/N$ o prejuízo a cada turno é pelo menos $1 - 1/N$ e o prejuízo em T turnos é $T(1 - 1/N)$. Por outro lado, existe o algoritmo $g(t) = i_t$ com um prejuízo total de 0.

Solução: comparar somente com algoritmos que jogam sempre uma única ação. Em particular, comparar H com o melhor desses algoritmos. Assim, temos o conceito do *external regret*:

- $R = L_H^T - L_{min}^T$, onde $L_{min}^T = min_i L_i^T$

Algoritmo Guloso (*Greedy*)

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

- Inicialização: $x^1 = 1$

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

- Inicialização: $x^1 = 1$
- A cada passo t:

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

- Inicialização: $x^1 = 1$
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

- Inicialização: $x^1 = 1$
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$
  - $S^{t-1} = \{i : L_i^{t-1} = L_{min}^{t-1}\}$

Algoritmo Guloso (*Greedy*)

- Ideia do algoritmo: a cada turno escolher uma ação com o menor prejuízo acumulado até o momento.
- Por simplicidade, assumimos que $\ell_i^t = \{0, 1\}$

Algoritmo:

- Inicialização: $x^1 = 1$
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$
  - $S^{t-1} = \{i : L_i^{t-1} = L_{min}^{t-1}\}$
  - $x^t = min\{S^{t-1}\}$

Algoritmo Guloso (*Greedy*)

Algoritmo Guloso (*Greedy*)

Theorem

Para qualquer sequência de prejuízos o algoritmo guloso tem:

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

Algoritmo Guloso (*Greedy*)

Theorem

Para qualquer sequência de prejuízos o algoritmo guloso tem:

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

PROVA: a cada turno t em que há prejuízo na ação escolhida pelo algoritmo, mas L^t_{min} não aumenta, então ao menos uma ação é removida de S^t. Isso ocorre no máximo N vezes antes que L^t_{min} aumente em 1. Portanto, o algoritmo causa um prejuízo de no máximo N entre cada vez que L^t_{min} aumenta.

Prejuízo de algoritmos determinísticos

Prejuízo de algoritmos determinísticos

Theorem

Para qualquer algoritmo determinístico D existe uma sequência de prejuízo para a qual $L_D^T = T$ e $L_{min}^T = \lfloor T/N \rfloor$.

(O que implica que $L_D^T \geq N \cdot L_{min}^T + (T mod N)$*, o que é próximo do limite do Greedy).*

Prejuízo de algoritmos determinísticos

Theorem

Para qualquer algoritmo determinístico D existe uma sequência de prejuízo para a qual $L_D^T = T$ e $L_{min}^T = \lfloor T/N \rfloor$.

(O que implica que $L_D^T \geq N \cdot L_{min}^T + (T mod N)$*, o que é próximo do limite do Greedy).*

PROVA: D escolhe a ação x^t no turno t. Construímos os vetores de prejuízo da seguinte maneira: no turno t o prejuízo de x^t é 1 e das outras ações é 0. Portanto, D tem um prejuízo de 1 a cada turno e, assim, $L_D^T = T$.

Como existem N ações diferentes, existe alguma ação que o algoritmo D escolheu no máximo $\lfloor T/N \rfloor$ vezes e, por construção, somente essas ações geraram prejuízo. Portanto, $L_{min}^T \leq \lfloor T/N \rfloor$.

Algoritmo guloso randomizado

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $p_i^1 = 1/N$ para todo $i \in X$.

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $p_i^1 = 1/N$ para todo $i \in X$.
- A cada passo t:

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $p_i^1 = 1/N$ para todo $i \in X$.
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $p_i^1 = 1/N$ para todo $i \in X$.
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$
  - $S^{t-1} = \{i : L_i^{t-1} = L_{min}^{t-1}\}$

Algoritmo guloso randomizado

- Ideia: semelhante ao algoritmo guloso, mas, em caso de empate, distribui a probabilidade entre as ações empatadas ao invés de escolher uma única ação.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $p_i^1 = 1/N$ para todo $i \in X$.
- A cada passo t:
  - $L_{min}^{t-1} = min_{i \in X} L_i^{t-1}$
  - $S^{t-1} = \{i : L_i^{t-1} = L_{min}^{t-1}\}$
  - $p_i^t = \frac{1}{|S^{t-1}|}$, se $i \in S^{t-1}$;
    ou 0, caso contrário.

Algoritmo guloso randomizado

Algoritmo guloso randomizado

Theorem

O algoritmo guloso randomizado (RG), para quaisquer sequências de prejuízo, tem:

$$L_{RG}^T \leq (1 + lnN)L_{min}^T + (lnN)$$

Algoritmo guloso randomizado

Theorem

O algoritmo guloso randomizado (RG), para quaisquer sequências de prejuízo, tem:

$$L_{RG}^{T} \leq (1 + lnN)L_{min}^{T} + (lnN)$$

PROVA: seja t_j o turno em que L_{min}^{t} atinje um prejuízo j. A qualquer turno t temos que $1 \leq |S^t| \leq N$. Além disso, se no turno $t \in (t_j, t_{j+1}]$ o tamanho de S^t diminui em k de n' para $n' - k$, então o prejuízo de RG é k/n', já que cada ação tem peso $1/n'$. Por resultados conhecidos, k/n' pode ser limitado por $\frac{1}{n'} + \frac{1}{n'-1} + ... + \frac{1}{(n'-k+1)}$. Portanto, ao longo do intervalo $(t_j, t_{j+1}]$, o prejuízo do RG é no máximo:

$$\frac{1}{N} + \frac{1}{N-1} + ... + \frac{1}{1} \leq 1 + lnN.$$

Randomized weighted majority

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1 - \eta)^{L_i}$.

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0, 1\}$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:
  - Se $\ell_i^{t-1} = 1$, $w_i^t = w_i^{t-1}(1-\eta)$; senão, $w_i^t = w_i^{t-1}$.

Randomized weighted majority

- Ideia: uma das fraquezas do RG é quando $|S^t|$ é pequeno. O RWM dá um peso a ações que são próximas do ótimo, ao invés de ignora-las completamente.Assim, cada ação i com um prejuízo total L_i tem um peso $w_i = (1-\eta)^{L_i}$.
- Ainda assumimos que $\ell_i^t = \{0,1\}$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:
  - Se $\ell_i^{t-1} = 1$, $w_i^t = w_i^{t-1}(1-\eta)$; senão, $w_i^t = w_i^{t-1}$.
  - $p_i^t = w_i^t / W^t$, onde $W^t = \sum_{i \in X} w_i^t$.

Randomized weighted majority

Randomized weighted majority

Theorem

Para $\eta \leq 1/2$ o prejuízo do Randomized Weighted Majority(RWM), para quaisquer sequências de prejuízos, satisfaz:

$$L_{RWM}^{T} \leq (1+\eta)L_{min}^{T} + \frac{lnN}{\eta}$$

Polynomial weights algorithm

Polynomial weights algorithm

- Ideia: generalização do RWM para prejuízos em $[0,1]$.

Algoritmo:

Polynomial weights algorithm

- Ideia: generalização do RWM para prejuízos em $[0, 1]$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$

Polynomial weights algorithm

- Ideia: generalização do RWM para prejuízos em $[0, 1]$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:

Polynomial weights algorithm

- Ideia: generalização do RWM para prejuízos em $[0,1]$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:
  - $w_i^t = w_i^{t-1}(1 - \eta \ell_i^{t-1})$.

Polynomial weights algorithm

- Ideia: generalização do RWM para prejuízos em $[0, 1]$.

Algoritmo:

- Inicialização: $w_i^1 = 1$ e $p_i^1 = 1/N$, para todo $i \in X$
- A cada passo t:
  - $w_i^t = w_i^{t-1}(1 - \eta \ell_i^{t-1})$.
  - $p_i^t = w_i^t / W^t$, onde $W^t = \sum_{i \in X} w_i^t$.

Polynomial weights algorithm

Theorem

Para $\eta \leq 1/2$*, qualquer* $\ell_i^t \in [0,1]$ *e qualquer* k *o prejuízo do Polynomial Weights Algorithm satisfaz:*

$$L_{PW}^T \leq L_k^T + \eta Q_k^T + \frac{lnN}{\eta}$$

onde $Q_k^T = \sum_{t=1}^T (\ell_k^t)^2$*.*

Constant sum game

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

- $M =$ Conjunto de m jogadores

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

- $M =$ Conjunto de m jogadores
- $X_i =$ Conjunto de N ações do jogador i

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

- M = Conjunto de m jogadores
- X_i = Conjunto de N ações do jogador i
- s_i = Função de prejuízo do jogador i; $s_i : X_i \times (\times_{j \neq i} X_j) \rightarrow [0, 1]$

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

- M = Conjunto de m jogadores
- X_i = Conjunto de N ações do jogador i
- s_i = Função de prejuízo do jogador i; $s_i : X_i \times (\times_{j \neq i} X_j) \rightarrow [0, 1]$
- P_i^t = distribuição de probabilidade do jogador i no turno i

Constant sum game

Um jogador i joga um jogo $G = \langle M, (X_i), (s_i) \rangle$ por T turnos usando o algoritmo ON.

- M = Conjunto de m jogadores
- X_i = Conjunto de N ações do jogador i
- s_i = Função de prejuízo do jogador i; $s_i : X_i \times (\times_{j \neq i} X_j) \rightarrow [0, 1]$
- P_i^t = distribuição de probabilidade do jogador i no turno i
- P_{-i}^t = distribuição de probabilidade dos outros jogadores

Constant sum game de dois jogadores

Constant sum game de dois jogadores

- $G = \langle \{1, 2\}, (X_i), (s_i) \rangle$

Constant sum game de dois jogadores

- $G = \langle \{1, 2\}, (X_i), (s_i) \rangle$
- $s_1(x_1, x_2) + s_2(x_1, x_2) = c$, para alguma constante c e quaisquer ações x_1 e x_2.

Constant sum game de dois jogadores

- $G = \langle \{1, 2\}, (X_i), (s_i) \rangle$
- $s_1(x_1, x_2) + s_2(x_1, x_2) = c$, para alguma constante c e quaisquer ações x_1 e x_2.
- Qualquer jogo de soma constante tem um valor v_i, tal que o jogador i tem uma estratégia mista que garante um prejuízo esperado de no máximo v_i, independente da estratégia do outro jogador.

Constant sum game de dois jogadores

Constant sum game de dois jogadores

Theorem

Seja G *um jogo de soma constante de valor* (v_1, v_2)*. Se o jogador* $i \in \{1, 2\}$ *jogar por* T *turnos usando um algoritmo* ON *de external regret* R*, então seu prejuízo médio será no máximo* $v_i + R/T$*.*

Constant sum game de dois jogadores

Theorem

Seja G um jogo de soma constante de valor (v_1, v_2). Se o jogador $i \in \{1, 2\}$ jogar por T turnos usando um algoritmo ON de external regret R, então seu prejuízo médio será no máximo $v_i + R/T$.

PROVA: Pela teoria de jogos de soma constante, para qualquer estratégia mista q do jogador 2, o jogador 1 tem alguma ação x_k que garante um prejuízo esperado de no máximo v_1. Ou seja, se o jogador 1 sempre jogar a ação x_k, seu prejuízo seria no máximo $v_1 T$ e, portanto, $L^T_{min} \leq L^T_k \leq v_1 T$. Como o jogador 1 está usando um algoritmo ON de *external regret* R, temos que $L^T_{ON} \leq L^T_{min} + R \leq v_1 T + R$.

Swap regret

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- F = Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- $F =$ Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.
- $f_i^t = \sum_{j:F^t(j)=i} p_j^t =$ Nova probabilidade de escolher a ação i.

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- F = Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.
- $f_i^t = \sum_{j:F^t(j)=i} p_j^t$ = Nova probabilidade de escolher a ação i.
  - $f^t = F^t(p^t)$ = Nova distribuição de probabilidade

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- $F =$ Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.
- $f_i^t = \sum_{j:F^t(j)=i} p_j^t =$ Nova probabilidade de escolher a ação i.
  - $f^t = F^t(p^t) =$ Nova distribuição de probabilidade

Por exemplo, podemos ter a seguinte regra de modificação (que troca x_1 por b_2) e cálculo de arrependimento:

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- $F =$ Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.
- $f_i^t = \sum_{j:F^t(j)=i} p_j^t =$ Nova probabilidade de escolher a ação i.
  - $f^t = F^t(p^t) =$ Nova distribuição de probabilidade

Por exemplo, podemos ter a seguinte regra de modificação (que troca x_1 por b_2) e cálculo de arrependimento:

- $switch_i(x_1, b_1, b_2) = b_2$, se $x_1 = b_1$;
  ou x_1, caso contrário.

Swap regret

Consiste em comparar o seu algoritmo com um algoritmo parecido, que modifica algumas das suas ações por outras segundo uma *modification rule*.

- F = Função que recebe o histórico de ações e a ação atual e devolve uma nova ação.
- $f_i^t = \sum_{j:F^t(j)=i} p_j^t$ = Nova probabilidade de escolher a ação i.
  - $f^t = F^t(p^t)$ = Nova distribuição de probabilidade

Por exemplo, podemos ter a seguinte regra de modificação (que troca x_1 por b_2) e cálculo de arrependimento:

- $switch_i(x_1, b_1, b_2) = b_2$, se $x_1 = b_1$;
  ou x_1, caso contrário.
- $regret_i(x, f) = s_i - s_i(f(x_i), x_{-i})$

Correlated Equilibrium

Correlated Equilibrium

Definition

Uma probabilidade conjunta P em X é um *equilíbrio correlato* se, para todo jogador i e quaisquer ações $b_1, b_2 \in X$, temos que:

$$E_{x \sim P}[regret_i(x, switch_i(\cdot, b_1, b_2))] \leq 0$$

Correlated Equilibrium

Definition

Uma probabilidade conjunta P em X é um *equilíbrio correlato* se, para todo jogador i e quaisquer ações $b_1, b_2 \in X$, temos que:

$$E_{x \sim P}[regret_i(x, switch_i(\cdot, b_1, b_2))] \leq 0$$

Definition

Uma probabilidade conjunta P em X é um *equilíbrio ϵ-correlato* se, para todo jogador i e quaisquer ações $b_1, b_2 \in X$, temos que:

$$E_{x \sim P}[regret_i(x, switch_i(\cdot, b_1, b_2))] \leq \epsilon$$

Swap Regret e Correlated Equilibrium

Swap Regret e Correlated Equilibrium

Theorem

Dado o jogo $G = \langle\{1,2\},(X_i),(s_i)\rangle$ e assumindo que, por T turnos todo jogador segue uma estratégia de swap regret no máximo R. Então a distribuição empírica Q das ações conjuntas jogadas é um (R/T)-equilíbrio.

Swap Regret e Correlated Equilibrium

Theorem

Dado o jogo $G = \langle \{1,2\}, (X_i), (s_i) \rangle$ e assumindo que, por T turnos todo jogador segue uma estratégia de swap regret no máximo R. Então a distribuição empírica Q das ações conjuntas jogadas é um (R/T)-equilíbrio.

Ou seja, se todos os jogadores jogarem uma estratégia com arrependimento R, o jogo convergirá para um equilíbrio correlato.

Redução de External para Swap Regret

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

- Considere N cópias do algoritmo A de *external regret* R, $\{A_1, ..., A_N\}$.

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

- Considere N cópias do algoritmo A de *external regret* R, $\{A_1, ..., A_N\}$.
- A cada turno t:

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

- Considere N cópias do algoritmo A de *external regret* R, $\{A_1, ..., A_N\}$.
- A cada turno t:
  - Cada A_i devolve uma distribuição q_i^t, onde $q_{i,j}^t$ é a porcentagem que A_i atribuir à ação j;

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

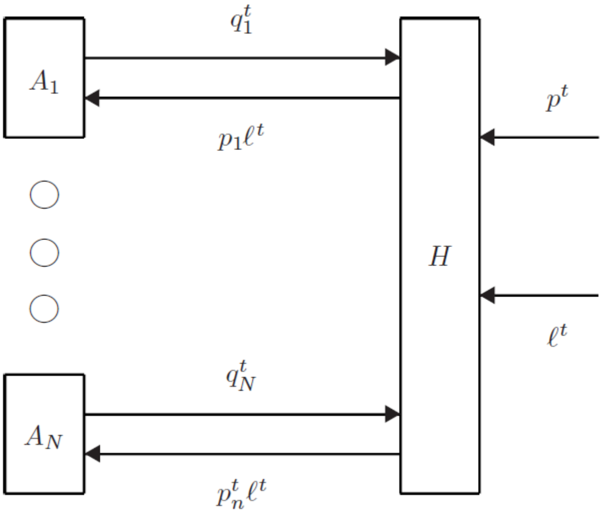
- Considere N cópias do algoritmo A de *external regret* R, $\{A_1, ..., A_N\}$.
- A cada turno t:
  - Cada A_i devolve uma distribuição q_i^t, onde $q_{i,j}^t$ é a porcentagem que A_i atribuir à ação j;
  - Calculamos p^t de modo que $p_j^t = \sum_i p_i^t q_{i,j}^t$.

Redução de External para Swap Regret

É possível, a partir de qualquer A algoritmo com um bom *external regret*, obter um algoritmo H com um bom *swap regret*.

- Considere N cópias do algoritmo A de *external regret* R, $\{A_1, ..., A_N\}$.
- A cada turno t:
  - Cada A_i devolve uma distribuição q_i^t, onde $q_{i,j}^t$ é a porcentagem que A_i atribuir à ação j;
  - Calculamos p^t de modo que $p_j^t = \sum_i p_i^t q_{i,j}^t$.
  - Quando recebermos o vetor de prejuízos ℓ^t, repassamos para cada A_i um "prejuízo ponderado" $p_i \ell^t$. Portanto, A_i tem um prejuízo $p_i^t(q_i^t \cdot \ell^t)$.

Redução de External para Swap Regret



Redução de External para Swap Regret

Redução de External para Swap Regret

Como A_i é um algoritmo de arrependimento R, temos que:

$$\sum_{t=1}^{T} p_i^t (q_i^t \cdot \ell^t) \leq \sum_{t=1}^{T} p_i^t \ell_i^t + R$$

Redução de External para Swap Regret

Como A_i é um algoritmo de arrependimento R, temos que:

$$\sum_{t=1}^{T} p_i^t (q_i^t \cdot \ell^t) \leq \sum_{t=1}^{T} p_i^t \ell_i^t + R$$

Somando os N algoritmos e considerando uma regra de modificação $F : \{1, ..., N\} \to \{1, ..., N\}$, temos que:

$$L_H^T \leq \sum_{i=1}^{N} \sum_{t=1}^{T} p_i^t \ell_{F(i)}^t + NR = L_{H,F}^T + NR$$

Redução de External para Swap Regret

Assim, temos os seguintes teorema:

Redução de External para Swap Regret

Assim, temos os seguintes teorema:

Theorem

Dado um algoritmo com external regret R *e qualquer função* $F : \{1, ..., N\} \rightarrow \{1, ..., N\}$, H *tem prejuízo:*

$$L_H \leq L_{H,F} + NR$$

Ou seja, o swap regret de H *é no máximo* NR.