Analisi del Paper
Data Poisoning Attacks on Stochastic Bandits
di Fang Liu e Ness Shroff

Jacopo Rodeschini 1046083
Department of Computer Science
Università degli studi Bergamo
Dalmine, 24044
j.rodeschini@studenti.unibg.it

Abstract

Il Paper proposto dagli autori dimostra come sia possibile manipolare il comportamento di un algoritmo di apprendimento. Nello studio proposto il caso preso in esame è il problema denominato *Multi-Armed Bandits problems* (MAB), questo tipo di problema potrebbe essere riassunto nel seguente modo: "scegliere tra un insieme di scelte indipendenti quella che possa generare il massimo guadagno". Sullo studio approfondito degli algoritmi che risolvono il problema MAB, gli autori del paper forniscono un metodo di analisi in grado di formulare il problema per dirottare il comportamento del decisore. Poiché esistono diversi algoritmi e diverse situazioni in cui il problema dei banditi viene applicato gli autori propongo in prima battuta una serie di algoritmi attaccanti specifici in base al tipo di problema MAB in analisi, per giungere poi ad un algoritmo adattivo all'istanza specifica del problema. Questo è un risultato di grande importanza perchè non richiede nessun tipo di conoscenza preliminare sul problema base da parte dell'attaccante che si "adatta" al problema. Infine mostrano come al momento non esiste un decisore (algoritmo) che sia in grado di difendersi dagli attacchi proposti con l'algoritmo adattativo. Il contributo portato da chi sta scrivendo questo testo è stato dimostrare la bontà degli algoritmi proposti dagli autori anche in problemi differenti rispetto a quelli studiati nel paper.

***Articolo Originale*:** http://proceedings.mlr.press/v97/liu19e

Delivery of Artificial Intelligence course project 7-nov 2020, Dalmine, Italy.

Contents

1 Contesto

In questo capitolo vengono introdotte le nozioni necessarie per comprendere appieno gli argomenti trattati nel seguito, con particolare attenzione alla descrizione del problema MAB e alle varie strategie che risolvono il problema, sia nella versione *offline* che nella versione *online*. La differenza tra le due versioni è che nella versione *offline* (o anche *batch*-learning) il sistema ri-stima i suoi parametri dopo aver ricevuto un numero ben definito di sample organizzati in *batch* (lotti), nella versione *online* il sistema ri-stima i propri parametri (ovvero la propria conoscenza) dopo aver campionato ogni sample.

Il lavoro è organizzato in questo modo: nel prossimo paragrafo verranno introdotte le nozioni di base al problema MAB con i principali algoritmi risolutivi, a seguire verrà descritto lo scenario in cui è presente un agente malintenzionato (attaccante). Nel secondo capitolo verranno elencati gli studi citati nel paper in analisi e nell'ultimo capitolo verrà approfondito ed analizzato lo studio condotto dagli autori Fang Liu e Ness Shroff.

1.1 Descrizione del problema

Il problema noto come *Multi-Armed Bandits problems* (MAB) è un problema in cui il *decisore* deve scegliere tra un insieme di scelte indipendenti quella che possa generare il massimo guadagno. La probabilità che una determinata scelta possa generare una ricompensa (*reward*) è ignota all'inizio del processo, e può essere parzialmente appresa solo "provando" tali scelte. In caso contrario il decisore ovviamente sceglierebbe sempre la scelta con più alta probabilità di successo. Inolte, la probabilità di ricevere un reward è indipendente dalla scelta effettuata al tempo precedente. Il decisore (noto anche come *giocatore* o *agente*) deve quindi scegliere quale azione scegliere in base alla conoscenza acquisita sino a quel momento, deve valutare se conviene operare ancora la stessa scelta (che ritiene sia la scelta migliore) o "provare" un scelta nuova. Questa ultima considerazione rientra in un dilemma noto come **exploration vs exploitation** trade-off ovvero, il decisore deve lavorare in una "via di mezzo" tra effettuare la scelta ottima in base alla conoscenza acquisita e provare nuove scelte. Infatti, decidere di non esploare nuove scelte potrebbe portare ad ottenere una ricompensa totale piccola, viceversa, continuare a esplorare nuove scelte porterebbe ad avere una scarsa conoscenza dell'ambiente (*environment*) perchè si avrebbero pochi sample su cui valutare le alternative e quindi operare scelte potenzialmente non ottime. La scelta della giusta "via di mezzo" è interpretata dall'algoritmo utilizzato, infatti molti algoritmi si differenziano proprio per come questa soglia viene selezionata. Spesso le scelte disponibili al decisore sono dette *braccia* o *leve* in paragone alle leve delle slot-machines, per cui "*tirare un braccio*" ha il significato di: "prendere una scelta".

Se supponiamo che l'algoritmo lavori in tempo discreto con $t = 1, 2, ..., T$, possiamo definire una *politica* π come la sequenza delle decisioni π_t prese in ogni istante $t \in T$, ovvero: $\pi = (\pi_t)_{t=1}^{T}$. Un indicatore di quanto una polita π del giocatore sia buona è la funzione di **regret** (rammarico): ogni volta che il decisore prende una decisione sub-ottima, ovvero con il valore atteso del reward che non sia quello massimo, il decisore a posteriori prova del rammarico tanto più alto quanto il valore ottenuto sia lontano dal profitto massimo ottenibile. Misura quindi la differenza tra il reward totale ottenibile da un oracolo che sceglierebbe sempre la scelta migliore e il reward cumulato durante tutti i tentativi collezionato dal decisore secondo la politica π .

Definizione del problema Indichiamo con $A = 1, 2, 3, ..., K$ l'insieme delle possibili K scelte. Ad ogni time-step *t* l'algoritmo prende la scelta $a_t \in A$ considerata ottima sino a quel momento ($\pi_t = a_t$) e in base alla scelta presa l'ambiente in cui l'agente opera genera una ricompensa (reward) $r_t \in \mathbb{R}$ con $r_t = \mu_{a_t} + n_t$ dove μ_a è il valore atteso del reward per il braccio a_t e n_t è un rumore additivo normale con media nulla e varianza σ^2, $\sim N(0, \sigma^2)$. Il valore atteso dei reward $\mathbb{E}[r_t] = \mu_{a_t}$, ovvero il valore atteso della ricompensa scegliendo il braccio a_t è μ_{a_t}. Come già anticipato, i valori attesi $\mu_{a \in A}$ non sono noti a priori al giocatore. Una prima possibile strategia per il giocatore con cui scegliere a_t potrebbe essere ad esempio $a_t = arg\max_{a \in A} \mathbb{E}_t[r_a]$, in questo modo sceglieremmo il braccio sfruttando la conoscenza acquisita delle prove precedenti. Se π è la politica del giocatore, il reward può essere espresso come:

$$R(T) = \sum_{t=1}^{T} [\max_{a \in A} \mu_a - r_t^{(\pi_t)}] \tag{1}$$

$$R(T) = \max_{a \in A}(\mu_a)T - \sum_{t=1}^{T} r_t^{\pi_t} \tag{2}$$

in base alla definizione data di $R(T)$ in 1, l'algoritmo che si avvicina alla soluzione ottima è quindi quello che minimizza la funzione di regret oppure quello che massimizza il reward cumulato totale $V(T)$ definito come $V(T) = \sum_{t=1}^{T} r_t^{\pi_t}$.

Esistono diverse strategie per minimizzare la funzione di regret e per ognuna ne esistono diverse varianti. Di seguito vengono riportate quelle più note. Per ognuna ne viene anche fornita una semplice implementazione software[1]. Le simulazioni numeriche sono state effettuate su 1000 trails di 1000 time-step ciascuno,con K = 5 le cui ricompense medie μ_a sono: [0.2 0.4 0.9 0.3 0] e il rumore è campionato da una normale $\sim N(0, \sigma^2)$ con σ = 0.1. Se facessimo sempre le scelta ottima otterremmo un valore totale finale di 900 (0.9*1000).

Prima di iniziare definiamo le seguente quantità che sarà utile per l'analisi di tutti gli algoritmi proposti: se definiamo $N_a(t)$ come il numero di volte che viene presa la scelta $a_t = a$ sino al tempo t,con $a \in A$, allora:

$$\hat{\mu}_a(t) := 1/N_a(t) \sum r_t I(a_t = a) \tag{3}$$

per cui $\hat{\mu}_a(t)$ è la media aritmetica dei reward ottenuti fino al tempo t quando è stata presa la scelta a.

1.1.1 Agente casuale

L'agente casuale ad ogni turno sceglie in modo casuale una possibile scelta tra le K ovvero $a_t = randi(1, K)$. Risulta essere il peggiore degli agenti in quanto non sfrutta la conoscenza acquisita ma decide sempre di operare scelte nuove. In questo caso il reward medio cumulato teorico dovrebbe essere 360. L'algoritmo computa un reward $V(T)$ medio di 360.27 .

1.1.2 Agente epsilon-greedy

Con questa strategia ad ogni passo, con una probabilità $p = 1 - \epsilon$ viene selezionato il braccio che con la conoscenza attuale ha la media stimata dei reward maggiore. ϵ è un parametro che regola il trade-off tra esplorare nuove scelte o operare la scelta considerata ottima, in letteratura è comunemente usato un valore ϵ di 0.1. Il comportamento di ϵ-greedy può essere formalizzato come:

$$\forall t \in T, a_t := \begin{cases} \text{estrazione da variabile uniforme} & p <= \epsilon \\ arg\max_{a \in A} \hat{\mu}_a(t-1) & \text{altrimenti} \end{cases} \tag{4}$$

Con questa strategia il reward cumulato totale medio simulato è: 843.41 ben superiore ha quanto ottenuto con un agente casuale.

Di questa strategia è presente anche una variante molto popolare per cui la probabilità ϵ non è fissa nel tempo ma decresce man mano che il tempo passa. Questo perchè l'agente acquisisce sempre più conoscenza fino ad un punto in cui non deve più esplorare nuove scelte ma deve operare solo la scelta che ritiene ottima. In letteratura è comunemente accettato un $\epsilon = 1/t$ che decresce molto velocemente. Con questa variante il reward medio ottenuto è: 692.94.

1.1.3 Agente UCB

Molto simile a quanto visto per ϵ-greedy anche UCB (Upper Confident Bound) si basa sulla stima della media empirica osservata dei reward e seleziona la scelta che presenta l'upper bound di ampiezza maggiore.

$$\forall t \in T, a_t = arg\max_{a \in A} \hat{u}_a(t) := \hat{\mu}_a(t-1) + 3\sigma\sqrt{\frac{log(t)}{N_a(t-1)}} \tag{5}$$

[1] Gli algoritmi sono stati implementati con il software MATLAB versione R2020a. Il software è disponibile al seguente link: GitHub.

Con questa strategia il reward medio ottenuto è: 897.32

1.1.4 Agente Thompson Sampling (TS)

L'agente di Thompson a differenza delle tecniche precedenti fa uso della stima bayesiana. $\forall t \in T, \forall a \in A$ si estrae un parametro $\theta_a(t)$ dalla distribuzione a posteriori $N(\frac{\hat{\mu}_a(t-1)}{\sigma^2}, \frac{\sigma^2}{N_a(t-1)})$ suggerita da [KKM13]. Dopodiche l'algoritmo opera la scelta $a_t = arg\max_{a \in A} \theta_a(t)$. Per generare θ_a è sufficiente campionare da una distribuzione $x \in N(0,1)$ e successivamente operare la seguente trasformazione:

$$\theta_a = x * \sigma_{\theta_a} + \mu_{\theta a} = x * \sqrt{\frac{\sigma^2}{N_a(t-1)}} + \frac{\hat{\mu}_a(t-1)}{\sigma^2} \tag{6}$$

Con questa strategia il reward medio ottenuto è: 897.28

Approfondimenti TS: Abbiamo detto che l'algoritmo Thompson Sampling fa uso della stima bayesiana, in pratica possiamo vedere ogni braccio come una distribuzione $(\rho_a, a \in A)$ da cui possiamo campionare i reward per massimizzare la ricompensa totale. Le distribuzioni sui bracci dipendono da un parametro $\theta \in \mathbb{R}$ per cui $\rho_a(\cdot, \theta_a)$. L'algoritmo deve quindi fornire un metodo per selezionare la distribuzione da cui campionare il reward osservando le precedenti scelte ed i relativi reward ottenuti. L'algoritmo TS fissa una *prior* π_0 su ogni θ_a, la probabilità a posteriori $\pi_{a,t}$ è continuamente aggiornata in accordo con i reward ottenuti. Per cui, ad ogni time-step t viene campionato θ_a dalla distribuzione a posteriori $\pi_{a,t}$ e successivamente viene selezionato il braccio ottimo: $a_t = arg\max_{a \in A} \theta_a$. La probabilità a posteriori è ricavata tramite il teorema di bayes, per cui, in forma generale abbiamo:

$$p(\theta_a|x) = \frac{p(x|\theta_a)p(\theta_a)}{\int_{\theta'} p(x|\theta')p(\theta')\mathrm{d}\theta'} \tag{7}$$

dove $p(\theta_a|x)$ è la probabilità a posteriori $\pi_{\mathrm{a,t}}$, ed è ignota al modello, $p(x|\theta_a)$ è la verosimiglianza, ed indica la probabilità di aver osservato un certo dato x (il reward) condizionato a un certo valore di θ_a, mentre $p(\theta_a)$ è la distribuzione a priori (*prior*) del parametro θ_a, π_0. In questo testo, nel caso in cui i reward siano distribuiti come una normale useremo come distribuzione a priori la *Jeffrey prior* definita, nella sua formulazione generale, nel modo seguente:

$$p_J(\overline{\theta}) \propto \sqrt{\det \mathcal{I}(\overline{\theta})}$$

dove $\overline{\theta}$ è il vettore dei parametri ignoti, per cui, la densità di probabilità è proporzionale alla radice quadrata del determinante dell'*informazione di Fisher*. Dalla equazione 7 se $p(\theta)$ è gaussiana e $p(\cdot, \theta)$ è prodotto di gaussiane (verosimiglianza), allora anche $p(\theta, \cdot)$ è gaussiana[2] in quanto risulta essere un prodotto di gaussiane.

Inolte sappiamo che per distribuzioni canoniche esponenziali, con $\theta \in \mathbb{R}$, nella forma :

$$p(x|\theta) = A(x)exp(T(x)\theta - F(\theta)) \tag{8}$$

l'informazione di Fisher può essere calcolata come la derivata seconda di $F(\theta)$ [3], per cui:

$$\mathcal{I}(\theta) = F''(\theta) \text{ da cui: } p_J(\theta) \propto \sqrt{|F''(\theta)|} \tag{9}$$

Allora, sotto l'ipotesi di usare la *Jeffrey Prior*, la distribuzione a posteriori di θ dopo aver osservato n sample è:

[2] Quando $p(\theta, \cdot)$ e $p(\theta)$ hanno la stessa distribuzione, allora $p(\cdot, \theta)$ e $p(\theta)$ si dicono *coniugate*.

[3] Facciamo l'assunzione che $F(\theta)$ sia differenziabile due vote con la derivata seconda continua.

$$p(\theta|y1,\cdots,y_n) \propto \sqrt{F''(\theta)}exp\left(\theta\sum_{i=1}^{n}T(y_i) - nF(\theta_i)\right) \tag{10}$$

per cui, se considero $p(x|\theta) \sim N(\mu,\sigma)$ e definisco $\theta = \frac{\mu}{\sigma^2}$ allora è possibile ricavare la distribuzione a posteriori:

$$p(\theta,\overline{y}) \sim N(\frac{\sum_1^n y_i}{n}, \frac{\sigma^2}{n}) \tag{11}$$

Per cui, se consideriamo y_i come i reward ottenuti e ipotizziamo siano affetti da rumore normale additivo n_t allora: $N(\frac{\sum_1^n y_i}{n}, \frac{\sigma^2}{n}) = N(\frac{\tilde{\mu}_a(t-1)}{\sigma^2}, \frac{\sigma^2}{n})$.

Nel caso si stia analizzando una distribuzione dei reward secondo una binomiale è stata utilizzata la distribuzione *Beta* [4].

1.1.5 Comparazione risultati

Si riportano in Figura 1 i risultati raggiunti della varie strategie, i metodi *UCB* e *Thompson Sampling* raggiungono le performance migliori vicine al massimo ottenibile, questo perchè la varianza dell'errore σ bassa fa si che l'intervallo $\mu_a \pm 3\sigma_a$ sia centrato sulla media μ_a e non vi siano sovrapposizioni tra le distribuzioni. In Figura 2 il reward cumulato medio totale con $\sigma = 1$. Il reward cumulato medio molto vicino al massimo ottenibile significa che è stata presa spesso la decisione ottima. Una variante ai metodi ϵ-greedy e Thompson-Sampling potrebbe essere la generazione dei reward r_t, non più distribuiti come la normale ma secondo un'altra distribuzione, per esempio la binomiale. In Figura 3, le prestazione raggiunte dagli algoritmi con una distribuzione dei reward binomiale con probabilità di successo = μ_a. Per l'algoritmo di Thompson Sampling è stata utilizzata la distribuzione a posteriori definita in [KKM13].

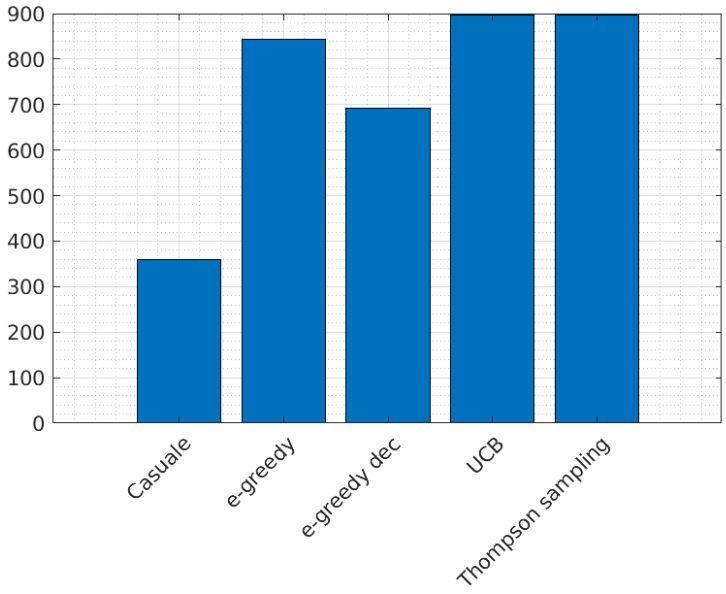

Figure 1: Risultati raggiunti delle varie strategie $\sigma = 0.1$

[4] I passaggi seguiti per trovare la distribuzione a priori nel caso in cui i reward siano distribuiti come una binomiale sono stati presi da: Conjungate Prior.

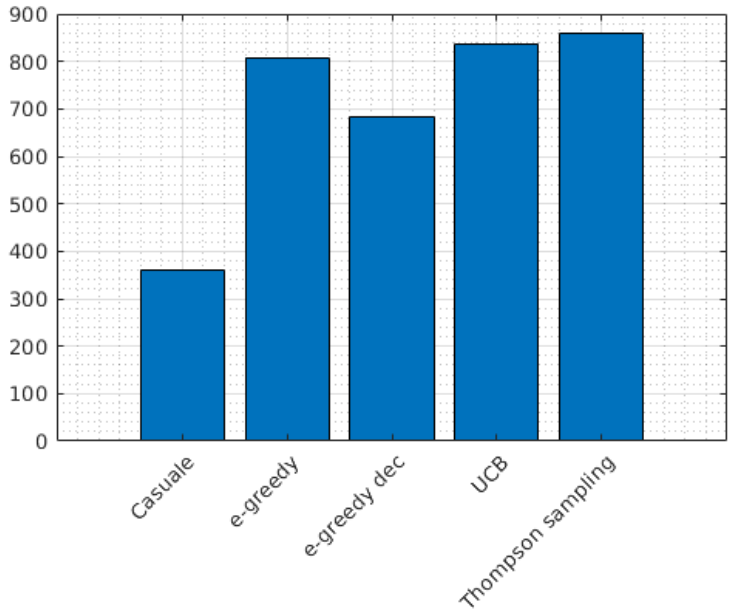

Figure 2: Risultati raggiunti delle varie strategie $\sigma = 1$

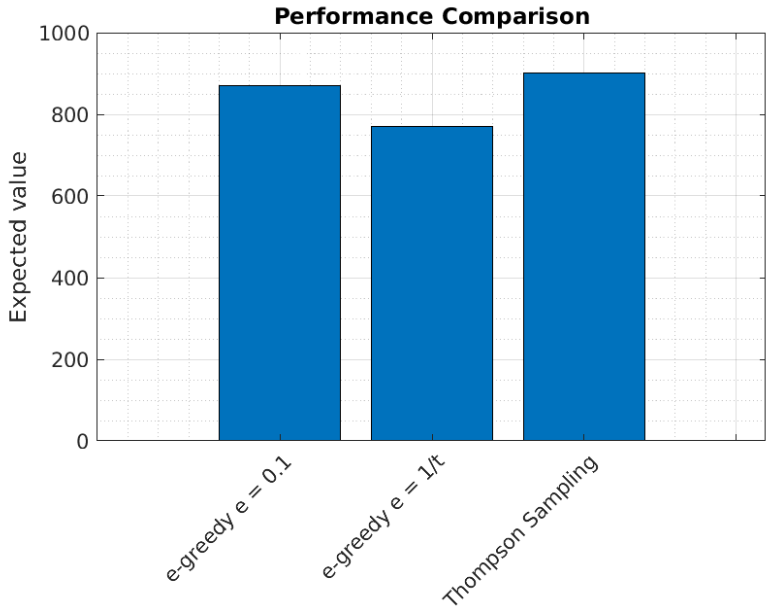

Figure 3: Risultati raggiunti delle varie strategie con distribuzione dei reward binomiale con probabilità di successo = μ_a

1.2 Attacchi

Un altro aspetto di valutazione degli algoritmi, è quanto siano sensibili ai disturbi o agli attacchi dall'esterno. Esistono diverse tipologie di attacco ad un sistema di apprendimento automatico, quella a cui qui si fa riferimento in questo testo e sarà sottintesa da ora in avanti è nota come *Adversarial Poisoning Attacks*: *Adversarial* indica che rientra nella categoria degli attacchi per contraddittorio ed è una tecnica che tenta di ingannare i modelli fornendo input "ingannevoli", è spesso utilizzata per causare malfunzionamenti ai sistemi di machine learning (ML); *Poisoning Attacks* è una tecnica

che consiste nella contaminazione dei dati di training, questa tecnica è stata molto studiata per quasi tutti i modelli di apprendimento: dai modelli autoregressivi in [AZB16], ai modelli SVM in [BNL12] con estensione a kernel non lineari, [HPGDA17] su reti neurali, in [JLMZ18] su MAB-stocastici e in [MJLZ18] sulla versione *Contestual*-MAB fino anche in [XBBFER15] su modelli di features selection come LASSO e Ridge. Questo tipo di attacco ha riscosso molto interesse nell'ultimo decennio, quando i modelli di apprendimento automatico (ML) hanno iniziato a diventare ampiamente usati in contesto industriale, ad esempio suggerire articoli o prodotti in siti di e-commece oppure per le campagne pubblicitarie e inserzioni sui siti web e sono spesso ri-stimati per adattarsi sempre meglio alle esperienze dei singoli utenti. In questo modo è anche possibile però inserire dati corrotti appositamente studiati per sfruttare vulnerabilità del modello di apprendimento. Sebbene siano molto studiati algoritmi di difesa per sistemi di *deep-learning* e di *superfised learning* ancora poco è stato fatto per i sistemi MAB sempre più usati in contesto industriale.

Nel paper proposto viene messo sotto analisi quanto gli algoritmi per MAB siano robusti rispetto a piccole perturbazioni nei dati e vengono forniti degli algoritmi attaccanti che riescono a manipolare il comportamento dell'algoritmo MAB portandolo a scegliere una scelta sub-ottima scelta a piacere dall'attaccante. Gli algoritmi proposti sono: sia per la versione offline sia per la versione online del problema MAB. Il contributo portato da chi sta scrivendo il testo è stato mostrare la bontà degli algoritmi forniti anche in situazioni differenti da quelle studiate, in particolare si analizza la situazione denominata *Multi-Batch*, verrà inoltre introdotta una variante all'algoritmo ϵ-greedy, verrà modificata la distribuzione dei reward, è verrà adoperato un attacco al problema MAB che sfrutta l'algoritmo *BaSE* (BAtch Succesive Eliminatios). Seppure in letteratura possiamo trovare studi che forniscono modelli di difesa agli attacchi provocati da una manipolazione nei dati di training (ad es: [GTK19] e [LMP18]) perdendo in prestazioni, Fang Liu e Ness Shroff dimostrano che non esistono algoritmi immuni da *Data Poisoning Attacks* rispetto alla strategia da loro proposta nella configurazione MAB-offline.

2 Studi Correlati

In questa sezione vengono riportati gli studi citati dal paper, di tutti è stato letto l'abstract, nel caso in cui siano stati necessari ulteriori chiarimenti sono state lette anche le sezioni di interesse.

[AZB16] Scott Alfeld, Xiaojin Zhu e Paul Barford: Data poisoning attacks against autoregressive models. n *AAAI*, pp. 1452–1458, 2016.

Gli autori si focalizzano sullo studio di possibili attacchi a modelli di previsione, in quanto, specialmente sui modelli economici un attore in un dato mercato può essere incentivato a guidare le previsioni in una certa direzione a proprio vantaggio. Nell'articolo si occupano dell'impostazione non iid delle previsioni di serie temporali considerando la classe dei modelli autoregressivi lineari. Descrivono un metodo per calcolare l'attacco ottimale, trattabile computazionalmente e ne dimostrano empiricamente la sua efficacia. Infine trattano possibili strategie difensive di fronte a possibili attacchi di questo tipo.

[BNL12] Battista Biggio, Blaine Nelson e Pavel Laskov: Poisoning attacks against support vector machines. *In Proceedings of the 29th International Conference on Machine Learning*, pp. 1467–1474, 2012.

Gli autori analizzano la classe degli attacchi *Poisoning Attacks* contro Support Vector Machines (SVM). Dimostrano che un avversario intelligente può, in una certa misura, prevedere il cambiamento della funzione decisionale dell'SVM a causa di input dannosi e utilizzare questa capacità per costruire dati dannosi. L'attacco proposto utilizza una strategia di *gradient ascent* in cui il gradiente viene calcolato in base alle proprietà della soluzione ottimale dell'SVM. Questo metodo può essere *kernelized* e consente di costruire l'attacco nello spazio di input anche per kernel non lineari.

[CMR15] Olivier Chapelle, Eren Manavoglu e Rómer E Rosales: Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2015.

In questo articolo, gli autori presentano un framework di apprendimento automatico basato sulla regressione logistica progettato specificamente per ottimizzare la visibilità degli annunci online, basandosi sulla frequenza di click e sulle conversazioni.

[GLK16] Aurélien Garivier, Emilie Kaufmann e Tor Lattimore: On explore-then-commit strategies. *In Advances in Neural Informa-tion Processing Systems*, pp. 784–792, 2016.

Gli autori affrontano il problema di minimizzare il più possibile il *regret* nei problemi *two-armed bandit problems* (2AB) con reward distribuito secondo una gaussiana. L'obiettivo è illustrare che le strategie basate su una fase di esplorazione (fino a un tempo di arresto) seguita dallo sfruttamento della conoscenza sono necessariamente sub-ottimali. Inoltre analizzano strategie completamente sequenziali con garanzie di *regret* a tempo finito che sono (1) asintoticamente ottimali al crescere dell'orizzonte temporale e (2) ordine-ottimale nel senso minimax. Infine ne discutono l'estensione al caso non gaussiano e al problema *multi-armed bandit problems* (MAB).

[GSS15] Ian J. Goodfellow, Jonathon Shlens e Christian Szegedy: Explaining and harnessing adversarial examples. *In International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572**.**

Gli autori studiano e sostengono che la causa principale della vulnerabilità delle reti neurali ad *Adversarial Attacks* è la loro natura lineare invece che, come comunemente spiegato, con la non linearità delle rete neurali e l'overfitting. Inoltre progettano un metodo semplice e veloce per generare esempi di contraddittorio, ed utilizzando questi esempi durante la fase di addestramento, riducono l'errore sul dataset di *test* di una rete maxout.

[GKT19] Anupam Gupta, Tomer Koren e Kunal Talwar: Better algorithms for stochastic bandits with adversarial corruptions. *arXiv preprint 1902. 08647*, 2019.

Gli autori studiano la versione stocastica del problema *multi-armed Bandits* (MAB) in presenza di corruzione dei dati, presentando un nuovo algoritmo in cui regret è quasi ottimale. L'algoritmo proposto è indipendente dal livello di contaminazione dei dati e può tollerare una quantità significativa di corruzione senza praticamente alcun degrado delle prestazioni.

[HPGDA17] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan e Pieter Abbeel: Adversarial attacks on neural network policies. *arXiv preprint 1702. 02284*, 2017.

In questo lavoro, gli autori dimostrano che gli attacchi avversari sono efficaci anche quando prendono di mira le politiche delle reti neurali nell'apprendimento per rinforzo. In particolare, dimostrano che le tecniche di creazione di esempi di "contraddittorio" esistenti possono essere utilizzate per ridurre in modo significativo le prestazioni sul dataset di *test*. Il modello di attacco utilizzato considera gli avversari in grado di introdurre piccole perturbazioni nell'input del modello.

[JLMZ18] Kwang-Sung Jun, Lihong Li, Yuzhe Ma e Xiaojin Zhu: Adversarial attacks on stochastic bandits. *arXiv preprint 1810. 12188*, 2018.

Gli autori studiano gli *Adversarial Attacks* che manipolano i reward per controllare le azioni scelte di un algoritmo *Multi-Armed Bandits* (MAB) stocastico. Propongono il primo attacco contro due popolari algoritmi di banditi: ϵ-greedy e UCB, *senza* la conoscenza delle ricompense medie.

[KKM13] Nathaniel Korda, Emilie Kaufmann e Remi Munos : Thompson sampling for 1-dimensional exponential family bandits. In Advances in Neural Information Processing Systems, pp. 1448–1456, 2013.

Thompson Sampling è stato dimostrato ottimale in molti modelli *Multi-Armed Bandits*, tuttavia le garanzie teoriche disponibili sono ancora limitate al caso Bernoulli. In questo lavoro gli autori estendono e dimostrando l'ottimalità asintotica dell'algoritmo. Inoltre l'analisi proposta, copre alcune distribuzioni per le quali non è stato ancora proposto alcun algoritmo ottimo, comprese le famiglie esponenziali a coda pesante. Questo lavoro, verrà ripreso più volte in questo lavoro, in quanto fornisce una distribuzione a posteriori da cui campionare il parametro θ per l'algoritmo di Thompson-Sampling. Nello specifico, per una distribuzione normale dei reward ed utilizzando la *Jeffreys Prior* come a distribuzione a priori, il parametro si distribuisce secondo una normale:

$\forall a \in A, \theta_a(t) \sim N(\frac{\tilde{\mu}_a(t-1)}{\sigma^2}, \frac{\sigma^2}{N_a(t)})$

[LWSV16] Bo Li, Yining Wang, Aarti Singh e Yevgeniy Vorobeychik: Data poisoning attacks on factorization-based collaborative filtering. *In Advances in neural information processing systems*, pp. 1885–1893, 2016.

Gli autori, studiano un attacco di avvelenamento dei dati sui sistemi di filtraggio collaborativo, dimostrando come un attaccante con piena conoscenza dell'algoritmo utilizzato può generare dati dannosi in modo da massimizzare i suoi obiettivi, imitando il normale comportamento dell'utente per evitare di essere rilevato. Sebbene il presupposto della conoscenza completa sembri un vincolo forte, consente una solida valutazione della vulnerabilità degli schemi di filtraggio collaborativo ad attacchi. Inoltre, presentano attaccanti efficienti per due popolari algoritmi di filtraggio collaborativo basati sulla fattorizzazione: la formulazione *alternative minimization* e il metodo *nuclear norm minimization*. Infine discutono potenziali strategie difensive.

[LCLS10] Lihong Li, Wei Chu, John Langford e Robert E. Schapire: A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670. ACM,2010.

In questo lavoro, gli autori modellano il sistema delle raccomandazioni personalizzate di articoli e notizie come un problema *Contestual multi-armed Bandits*. Un algoritmo di apprendimento seleziona sequenzialmente gli articoli per servire gli utenti sulla base di informazioni contestuali sia degli utenti che degli articoli, adattando contemporaneamente la sua strategia di selezione degli articoli basandosi sul feedback dei clic degli utenti per massimizzare i clic totali su tutti gli articoli. A seguito dello studio condotto, propongono un algoritmo che è computazionalmente efficiente.

[LHLSLS17] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu e Min Sun: Tactics of adversarial attack on deep reinforcement learning agents. arXiv preprint 1703.06748, 2017.

In questo testo, gli autori introducono due tattiche per attaccare gli agenti addestrati tramite *Deep Reinforcement Learning* utilizzando esempi di dati contraddittori, vale a dire l'attacco noto come: *strategically-timed* (tempo strategico) e l'attacco: *enchanting* (incanto). Nell'attacco a *tempo strategico*, l'avversario mira a ridurre al minimo la ricompensa dell'agente attaccando l'agente solo in un piccolo sottoinsieme di fasi temporali in un episodio per ridurre al minimo la possibilità di essere scoperti. Nell'attacco ad *incanto*, l'avversario mira ad attirare l'agente in uno stato bersaglio designato. Viene quindi creata una sequenza di esempi contraddittori per indurre l'agente a eseguire la sequenza di azioni desiderata.

[LWBSU18] Fang Liu, Sinong Wang, Swapna Buccapatnam e Ness Shroff: Ucboost, a boosting approach to tame complexity and optimality for stochastic bandits. *arXiv preprint 1804.05929*, 2018.

In questo lavoro gli autori, affrontano il problema di trovare algoritmi per problemi *Multi-Armed Bandits* (MAB) a bassa complessità computazionale e che siano ottimali. Propongono un approccio migliorativo agli algoritmi basati sull'Upper Confidence Bound (UCB) per (MAB)-stocastici, fornendo un algoritmo chiamato UCBoost. Ne forniscono due versioni: UCBoost(D) che gode sia di complessità O(1) per ogni braccio per round sia della garanzia di *regret* vicino a quello ottimo e UCBoost(ϵ) basato sull'approssimazione, che gode sia della una garanzia di *reget* ϵ-vicina a quella di un algoritmo ottimo sia di complessità O(log $(1/\epsilon)$) per ogni braccio per round. Quindi, gli algoritmi forniscono un modo pratico per scambiare l'ottimalità con la complessità computazionale.

[LMP18] Thodoris Lykouris, Vahab Mirrokni e Renato Paes Leme: Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 114–122. ACM, 2018.

Gli autori forniscono un nuovo modello di banditi stocastici con corruzioni contraddittorie che mira a catturare impostazioni in cui una frazione dell'input può essere manipolata per ingannare l'algoritmo. L'obiettivo di questo modello è incoraggiare la progettazione di algoritmi per MAB che funzionano bene in anche in presenza di avversari. L'algoritmo fornito conserva l'ottimalità (fino a un termine logaritmico) se l'input è stocastico e le cui prestazioni degradano linearmente con la quantità di corruzione totale aggiunta C. Dimostrando infine, che la perdita di prestazioni è necessaria se l'algoritmo raggiunge prestazioni ottimali nell'impostazione MAB stocastica.

[MJLZ18] Yuzhe Ma, Kwang-Sung Jun, Lihong Li e Xiaojin Zhu: Data poisoning attacks in contextual bandits. *arXiv preprint 1808.05760*, 2018.

Gli autori studiano gli attacchi per avvelenamento (*Data poisoning attacks*) nel problemi *Multi-Armed Bandits* offline. Forniscono un framework di attacco generale basato sull'ottimizzazione convessa e dimostriamo che manipolando leggermente le ricompense nei dati, un aggressore può forzare

l'algoritmo a scegliere un braccio bersaglio. Questo lavoro è alla base degli studi condotti nel paper in analisi, infatti la metodologia di ottimizzazione convessa verrà utilizzata anche nel paper in analisi. Inoltre, forniscono una misura per definire l'efficacia di un attacco: il *ratio*: $\|\overline{\epsilon}\|_2/\|\overline{t}\|_2$. Tale misura sarà utilizzata anche nei nostri risultati numerici per caratterizzare gli algoritmi di attacco.

[MZ15] Shike Mei e Xiaojin Zhu: Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pp. 2871–2877, 2015.

In questo studio è considerato lo scenario in cui un aggressore contamina il dataset di *train* in modo che l'algoritmo di apprendimento produca un modello redditizio per l'attaccante. Comprendere gli attacchi sul dataset di *train* è importante poiché la capacità di apprendimento può essere potenzialmente violata tramite i dati che ricevono dall'ambiente. Questo documento identifica l'attacco ottimale sul dataset di *train* su un'ampia categoria di *learners*. Per fare ciò, mostrano che l'attacco ottimale sul dataset di *train* può essere formulato come un problema di ottimizzazione. Infine, discutono le potenziali difese contro tali attacchi.

[Tho33] William R Thompson: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Articolo nel quale William R. Thompson introduce l'euristica denominata Thompson Sampling, di notevole importanza applicativa.

[WC18] Yizhen Wang e Kamalika Chaudhuri: Data poisoning attacks against online learning. *arXiv preprint 1808.08994*, 2018.

Gli autori studiano la classe degli attacchi: *Data poisoning attacks* (avvelenamento dei dati) per l'apprendimento online. Formalizzano il problema in due impostazioni e propongono una strategia di attacco generale, formulata come un problema di ottimizzazione, che si applica ad entrambi con alcune modifiche. Infine, discutono le implicazioni dei risultati per la costruzione di difese di successo.

[XBBFER15] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert e Fabio Roli: Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pp. 1689–1698, 2015.

In questo lavoro, gli autori forniscono una struttura per studiare la robustezza, rispetto a manipolazioni dei dati di *train*, dei metodi di selezione delle features più diffusi, tra cui LASSO, RIDGE regression ed elastic net. I risultati mostrano che i metodi di selezione delle features possono essere significativamente compromessi quando vittime di attacchi, per esempio possiamo ridurre LASSO a scelte quasi casuali delle features principali inserendo attentamente meno del 5% di campioni manipolati nel dataset di *train*.

3 Studio del Paper

Recentemente ([MJLZ18]), hanno dimostrano uno dei primi attacchi ad un problema MAB nella sua versione offline. In quel caso l'attaccante poteva manipolare i reward prima che venissero usati per l'update in modo da modificare il comportamento del decisore. Il risultato fondamentale raggiunto è stato dimostrare come una attacco ad un problema MAB possa essere formulato come un problema di ottimizzazione convessa. Come già introdotto nei capitoli precedenti, esistono numerose strategie e numerose varianti per risolvere il problema MAB, ognuna adattabile a situazioni particolari. Nei capitoli seguenti verranno studiati diversi algoritmi attaccanti, uno per ogni tipologia di strategia nella versione *offline* per seguire poi alla descrizione della strategia *online* e ad una sua formulazione generale per strategia di attacco. Infine, nella sezione Appendice 6 alcuni dettagli sull'implementazione software.

3.1 Definizione del problema

Definita la funzione $R(T)$ in (1), per ogni r_t ritornato dall'ambiente in cui l'agente opera, l'attaccante manipola il reward sommandoci un errore di manipolazione ϵ:

$$\forall t \in T : r'_t = r_t + \epsilon_t \tag{12}$$

Il costo totale pagato dall'attaccante $C(T)$ è calcolato come $C(T) = norma([\epsilon_1, \epsilon_2, ..., \epsilon_t]^T)$. Senza perdere di generalità possiamo considerare la norma-l^2 per il problema nella versione offline e la norma-l^1 per la versione online, i risultati ottenuti possono essere estesi alla norma-l^p. Se definiamo a^* la scelta sub-ottima target dell'attaccante tale che $u_{a^*} < \max_{a \in A} \mu_a$, allora l'obiettivo dell'attaccante è fare in modo che il decisore scelga a^* come scelta ottima con probabilità $1 - \delta$ con: $\delta > 0$. Ciò che si vuole fare però non è soltanto manipolare i dati affinché il comportamento dell'agente venga modificato, ma l'obiettivo è di farlo con un costo che sia il minimo possibile, ovvero l'attaccante vuole intervenire sui dati poche volte ed aggiungendo piccole dosi di errore di manipolazione al reward, in modo da non essere scoperto, o magari perchè inserire dati corrotti ha una costo pratico poi nella realtà in cui l'attaccante opera. Per cui è necessario, aggiungere un vincolo affinché $C(T)$ sia il più piccolo possibile per non essere scoperti.

3.2 Problema Offline.

Supponiamo che il sistema ri-stimi i sui parametri dopo aver acquisito batch di samples con lunghezza T per aggiornare la sua conoscenza. Allora $\forall t \in T$ il sistema seleziona una scelta a_t, a seguito della scelta presa l'ambiente genera un reward (ricompensa) ed invia la tupla $(a_t, r_t)_{t<T}$ al buffer che mantiene tutte le tuple in cui $t < T$. Prima che il buffer con tutte le T tuple sia inviato al decisore, l'attaccante accede ai dati e manipola gli r_t sommandoci un errore di manipolazione (12). Il buffer modificato $(a_t, r'_t)_{t<T}$ viene inviato al decisore che aggiorna la sua conscenza senza sapere che è vittima di un attacco, per cui l'attaccante forza il decisore a scegliere all'istante $t = T + 1$ la scelta a^* ovvero $a_{T+1} = a^*$ con $C(T)$ minimo possibile.

Definiamo $m_a := N_a(T)$ come il numero di volte che è stata presa la scelta a fino al tempo T. $\forall a \in A$ definiamo $\overline{y_a} \in R^{m_a}$ il vettore dei reward ottenuti quando è presa la scelta a.

$$\overline{y_a} := (r_t | a_t = a)^T \tag{13}$$

e sia $\overline{\epsilon_a} \in R^{m_a}$ la strategia dell'attaccante (ovvero, tutti gli errori introdotti dall'attaccante per forzare il comportamento dell'agente quando il braccio scelto $a_t = a$)

$$\forall a \in A : \overline{\epsilon_a} := (\epsilon_a | a_t = a)^T \tag{14}$$

Il reward ottenuto della scelta a dopo l'attacco è $\overline{y_a} + \overline{\epsilon_a}$ e per evitare di essere scoperti $\overline{\epsilon_a}$ deve essere il più piccolo possibile, se misuriamo il costo con la norma-l^2 allora:

$$C(T)^2 = \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2 \tag{15}$$

A questo punto possiamo definire il problema di ottimizzazione P associato al problema MAB offline.

$$P := \min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2, \text{ con vincoli: } \left\{ \mathbb{P}(a_{T+1} = a^*) >= 1 - \delta \qquad \forall a \neq a^* \right. \tag{16}$$

Ovvero, è un problema di minimizzazione sul costo totale dell'attacco con il vincolo che la probabilità di scegliere a^* al tempo $T + 1$ sia almeno $1 - \sigma$, dove $\delta \in (0, 1]$ indica una tolleranza accettata. Non è possibile avere $\mathbb{P}a_{T+1} = a^* = 1$ perchè in alcuni algoritmi è presente una componente stocastica di casualità che non può essere controllata dall'attaccante.

Proposizione 1. Data una qualsiasi quantità $\delta > 0$. Se $\overline{\epsilon^*_{a \in A}}$ è la soluzione ottima al problema P allora è anche la migliore strategia di attacco.

La Proposizione 1. è molto forte, infatti ci basta risolvere il problema di ottimizzazione P per trovare la strategia di attacco. A questo punto abbiamo che la $\mathbb{P}(a_t = a)$ dipende dall'algoritmo utilizzato per il problema MAB, in seguito vengono proposti degli algoritmi specifici di attacco in base al problema MAB. Per cui supponiamo che l'attaccante conosca l'algoritmo utilizzato dal problema MAB, in particolare gli autori del paper forniscono gli algoritmi di attacco derivati dal problema P per gli algoritmi ϵ-greedy, UCB e Thompson Sampling.

Prima di cominciare definiamo

$$\tilde{\mu}_a(t) := \frac{1}{N_a(t)} \sum_{1}^{t} r'_t I(a_t = a) \tag{17}$$

come la media empirica osservata dopo l'attacco al tempo t. La funzione $I(.)$ è una funzione indicatrice ovvero, vale 1 dove l'argomento è vero, 0 altrimenti. In altre parole $\tilde{\mu}_a(t)$ per la scelta a è definita come il rapporto tra tutti i reward osservati fino a t a seguito dell'attacco quando $a_t = a$ e il numero di volte che ho preso la scelta a.

Per cui:

$$\tilde{\mu}_a(t) := \frac{1}{N_a(t)} \sum_{1}^{t} (y_a(t) + \epsilon_a(t)) I(a_t = a) \tag{18}$$

dove $(\epsilon_a(t))_{a \in A}$ è una quantità ignota (è la soluzione del nostro problema di ottimizzazione). Passando ad una formulazione vettoriale:

$$\tilde{\mu}_a(t) : \frac{1}{N_a(t)} \overline{y_a} I(a_t = a) + \frac{1}{N_a(t)} \overline{\epsilon_a} I(a_t = a) \tag{19}$$

Possiamo ora osservare che la quantità $\overline{y_a}$ è nota all'attaccante per cui il primo termine è la media aritmetica dei reward ottenuti sino al tempo t, non distorta dal rumore. Per cui,

$$\tilde{\mu}_a(t) : avg_t(\overline{y_a}) + \frac{1}{N_a(t)} \overline{\epsilon_a} I(a_t = a) \tag{20}$$

Questa nuova formulazione varrà più comoda nelle dimostrazioni presentate nei prossimi capitoli. Per semplicità di notazione, nei prossimi capitoli useremo la seguente notazione: $avg_t(\overline{y_a}) \rightarrow avg(a)$ indicando il valore atteso stimato dei reward per il braccio a al tempo T. Di seguito vengono proposte le strategie di attacco specifiche per ogni algoritmo. Per le simulazioni numeriche sono stati utilizzati gli stessi parametri del paper, in particolare: rumore campionato da una normale $\sim N(0, \sigma^2)$, con: $\sigma = 0.1$, il numero di scelte $K = 5$, il braccio (scelta) target a^* dell'attaccante ha $\mu_a^* = 0$ e si trova nella posizione $a_5, k = 5$, mentre $\forall a \in A, a \neq a^*, \mu_a$ è estratta casualmente da una distribuzione uniforme $\sim U(0, 1)$. Con questa configurazione, senza eseguire l'attacco, il braccio targhet non dovrebbe mai essere scelto come scelta ottima. Per valutare le performance raggiunte dagli algoritmi specifici di attacco si utilizza il ratio, definito in [MJLZ18] come:

$$\frac{\|\overline{\epsilon}\|_2}{\|\overline{y}\|_2} = \sqrt{\frac{\sum_{a \in A} \|\overline{\epsilon_a}\|_2}{\sum_{a \in A} \|\overline{y_a}\|_2}} \tag{21}$$

e misura la frazione di costo rispetto al valore totale raggiunto dall'agente senza attacco.

3.2.1 ϵ-greedy

In questa sezione viene descritta la strategia di attacco ad un problema MAB che sfrutta l'algoritmo ϵ-greedy. Il funzionamento di ϵ-greedy è descritto da 4. Il parametro ϵ con cui l'algoritmo estrae casualmente una possibile scelta non è noto a priori e non può essere controllato dall'attaccante. Per fare in modo che il decisore selezioni la scelta a^* è necessario che l'attaccante soddisfi questa condizione: $\forall a \in A, \tilde{\mu}_a^* \geq \tilde{\mu}_a$ ovvero il valore atteso dei reward di a^* deve essere maggiore rispetto ai valori attesi per le altre possibile scelte. Inoltre deve essere che $\mathbb{P}(a_{T+1} = a^*) = 1 - \sigma$. Le due condizioni appena descritte possono essere riformulate come segue: $\tilde{\mu}_a^* \geq \tilde{\mu}_a + \xi$ dove $\xi > 0$ è una variabile di scarto (o di surplus) comunemente usata nei problemi di ottimizzazione. Chiaramente al diminuire di ξ anche $C(T)$ decresce. A livello pratico è anche possibile utilizzare l'uguaglianza: $\tilde{\mu}_a^* = \tilde{\mu}_a + \xi$, in questo modo la differenza tra il valore atteso della scelta a^* e le altre possibili scelte sarà esattamente ξ.

Possiamo definire quindi il problema di ottimizzazione P_1 derivato dal problema P in 16 specializzandolo per l'istanza ϵ-greedy del problema MAB in questo modo:

$$P_1 := min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2, \text{ con vincoli: } \left\{ \tilde{\mu}_a^*(T) \geq \tilde{\mu}_a(T) + \xi \qquad \forall a \neq a^* \right. \tag{22}$$

Questo problema di ottimizzazione implica che: $\mathbb{P}(a_{T+1} = a^*) = 1 - \frac{K-1}{K}\epsilon_{T+1}$, per cui si evidenzia la dipendenza da K e da ϵ. In una prima approssimazione, se $K \sim 1$ allora $\mathbb{P}(a_{T+1} = a^*) = 1$, ovvero posso scegliere l'unico braccio disponibile, risulta quindi non essere un caso rilevante ai fini pratici; Nel caso in cui $K \sim \infty$ allora $\mathbb{P}(a_{T+1} = a^*) = 1 - \epsilon_{T+1}$ ovvero dipende solo da $\epsilon_{\text{T+1}}$. La condizione $K \sim \infty$ è facile da raggiungere, infatti è necessario solo che il rapporto $(K-1)/K \sim 1$ si pensi per esempio a problemi di marketing in cui si hanno a disposizione centinaia di scelte, inoltre se è utilizzata la versione ϵ-greedy decrescente, avremo che $\lim_{t \to \infty} \epsilon_t \sim 0$ per cui dopo un certo t avremo che $\mathbb{P}(a_{T+1} = a^*) \sim 1$

Teorema 1. $\forall \xi > 0, \forall \overline{y_a}_{\in A} \; \exists$ almeno una soluzione ottima al problema P1 che sia lineare nei vincoli.

Per dimostrare il Teorema 1 è sufficiente verificare che l'insieme delle soluzioni ammissibili non sia vuoto, ovvero trovare una soluzione $\overline{\epsilon_a}$ che soddisfi i vincoli. Partendo dalla definizione del vincolo: $\tilde{\mu}_{a^*} \geq \tilde{\mu}_a + \xi$ e scomponendo i termini nelle varie componenti abbiamo:

$$(\overline{y_{a^*}} + \overline{\epsilon_{a^*}})1/m_{a^*} = (\overline{y_a} + \overline{\epsilon_a})1/m_a) + \xi \tag{23}$$

possiamo osservare che $\forall \xi > 0, \forall \overline{\epsilon_{a^*}}, \forall \overline{y_a}_{\in A}$:

$$\forall a \in A : \overline{\epsilon_a} = [(\overline{y_{a^*}} + \overline{\epsilon_{a^*}})1/ma^* - \overline{y_a}1/ma - \xi]1 \tag{24}$$

dove 1 è un vettore di tutti uni e serve per permettere la divisione tra vettore e scalare. Per cui siamo riusciti a trovare una soluzione ammissibile al problema e dimostrare quindi che esiste almeno una soluzione. Dimostrando così il Teorema 1.

Successivamente aver dimostrato che P_1 è un problema di ottimizzazione quadratica con vincoli lineari, per risolverlo si è ricorsi alla programmazione quadratica definita come:

$$x = min_x \frac{1}{2} x^T H x + f^t x, \text{ con vincoli } \begin{cases} Ax \leq b \\ lb \leq x \leq ub \\ Aeqx = beq \end{cases} \tag{25}$$

dove $H, A, Aeq = 0$ sono matrici, $f = 0, b, beq = 0, lb = 0, ub = 0$ sono vettori, la costante $\frac{1}{2}$ non modifica la soluzione ottima del problema. Definendo la matrice H in questo modo:

$$H_{T,T} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{26}$$

si minimizza $\sum_{i=1}^{T} x_i^2$ ovvero la (norma-l^2)^2. La dimensione della matrice H è $[T \times T]$ perchè il vettore x è di dimensioni $[T \times 1]$ ovvero un vettore di lunghezza T, questo perchè abbiamo la possibilità di inserire un termine di manipolazione del reward ϵ_t fino al tempo T. I vincoli lineari sono espressi tramite la matrice A e il vettore dei termini noti b. Osserviamo che il vincolo è nella forma $\forall a \neq a^* : \tilde{\mu}_a^* \geq \tilde{\mu}_a + \xi$

per cui la possiamo scrivere come:

$$\forall a \neq a^* : avg(a^*) + \frac{1}{m_{a^*}} \overline{\epsilon_a} I(a_t = a*) \geq avg(a) + \frac{1}{m_a} \overline{\epsilon_a} I(a_t = a) + \xi \tag{27}$$

separiamo i termini noti al secondo membro e le incognite $\overline{\epsilon_a}$ al primo membro.

$$\forall a \neq a^* : \frac{1}{m_{a^*}} \overline{\epsilon_a} I(a_t = a*) - \frac{1}{m_a} \overline{\epsilon_a} I(a_t = a) \geq avg(a) - avg(a^*) + \xi \tag{28}$$

moltiplicando per -1 e scrivendolo nella forma $Ax \leq b$ abbiamo:

$$A = \begin{array}{c} \\ a_1 \\ \vdots \\ a_{K-1} \end{array} \begin{array}{c} \begin{array}{cccc} \epsilon_{(1:m_{a_1})} & \cdots & \cdots & \epsilon_{(m_{a_k}:T)} \end{array} \\ \begin{pmatrix} \frac{1}{m_{a_1}} & 0 & \cdots & -\frac{1}{m_{a_k}} \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & \frac{1}{m_{a_{K-1}}} & -\frac{1}{m_{a_K}} \end{pmatrix} \end{array} \tag{29}$$

$$b = \begin{array}{c} a_1 \\ \vdots \\ a_{K-1} \end{array} \begin{pmatrix} avg(a^*) - avg(a_1) - \xi \\ \vdots \\ avg(a^*) - avg(a_{K-1}) - \xi \end{pmatrix} \tag{30}$$

La matrice $A : [(K-1) \times T]$ così formata modella i vincoli del problema P_1, ogni riga corrisponde ad un vincolo sul braccio $a \neq a^*$, dove sono state inserite nelle prime m_{a_1} posizioni tutte le clausole sul primo vincolo in modo da interpretare $1/m_{a_1}$ per le volte in cui il decisore ha scelto il braccio a_1 in modo da modellizzare la funzione indicatrice $I(a_t = a_1)$. Stesso discorso per gli altri vincoli. Nelle ultime m_{a^*} colonne viene sottratta la quantità $-1/m_{a^*}$ per le volte che è stata vera la condizione $a_t = a^*$. Notare che non importa l'ordine con cui vengono inserite nelle righe i termini $1/m_a$ noi li consideriamo sequenziali per m_a volte, non è necessario che il decisore prenda m_a volte continuamente la stessa scelta a. Il vettore dei termini noti $b : [(K-1) \times 1]$ è costruito semplicemente sottraendo le medie stimate dei reward tra il braccio target e gli altri bracci rispettivamente $avg(a^*)$ e $avg(a) : \forall a \in A, a \neq a^*$.

Per calcolare il vettore delle medie al $t = T + 1$ è sufficiente osservare l'equazione 20, è quindi possibile creare una matrice $C(K \times T)$:

$$C = \begin{array}{c} \\ a_1 \\ \vdots \\ a_K \end{array} \begin{array}{c} \begin{array}{ccc} \epsilon_{(1:m_{a_1})} & \cdots & \epsilon_{(m_{a_k}:T)} \end{array} \\ \begin{pmatrix} \frac{1}{m_{a_1}} & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \frac{1}{m_{a_K}} \end{pmatrix} \end{array} \tag{31}$$

In questo modo $\tilde{\mu} = avg(\overline{y_a}) + C\epsilon^T$ e il braccio scelto all'istante $t = T + 1$ è: $a_{T+1} = arg \max_{a \in A}(\tilde{\mu})$.

Di seguito vengono riportati i risultati numerici ottenuti.

Commento risultati: Dai risultati numerici ottenuti possiamo riportare le seguenti considerazioni: la prima è che i risultati ottenuti non sono molto simili a quelli ottenuti nel paper studiato infatti il costo di attacco è differente. Come viene evidenziato in Figura 4 e in Figura 6, per tutti i tentativi compiuti l'attacco ha sempre successo, infatti a $t = T + 1$ il decisore prende la scelta K che corrisponde a a^*. Se non fosse stato eseguito l'attacco, in Figura 5 la scelta presa al $t = T + 1$ sarebbe stata la scelta 4, ovvero quella con la probabilità più alta. Infine, il costo di attacco mostrato nell'istogramma empirico dei ratio è del 35% per l'agente ϵ-greedy decrescente (valore diverso da quanto riportato nel paper, in Figura 3.a, pp 8) e inferiore del 18% per l'agente ϵ-greedy. Viene riportato in Figura 9 il diagramma delle frequenza di selezione per le K scelte, in questo caso si è preso un batch di dimensione 100 sample, per cui in totale 10 batch. Come si vede dal grafico durante il primo batch è la prima scelta ad essere selezionata per il numero maggiore di tentativi. A questo punto i reward vengono manipolati ed inviati all'agente che, inconsapevole di essere vittima di un attacco, aggiorna la sua conoscenza. Poiché i reward sono stati manipolati, risulta che il braccio ottimo è il braccio a^* obiettivo dell'attaccante. Dal secondo batch in poi sarà sempre a^* il braccio che verrà selezionato il maggior numero di volte. In figura 10 il costo di attacco in funzione della grandezza dei batch.

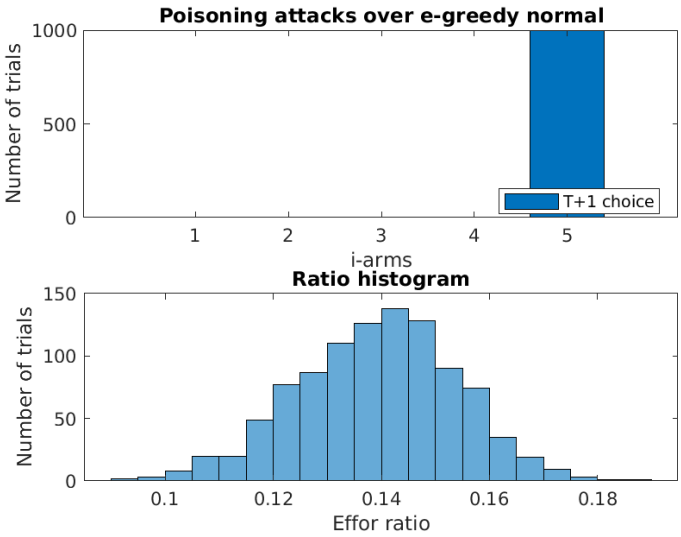

Figure 4: Performance attacco sul problema MAB-(-greedy) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T + 1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° $\mu_{a_5} = 0$). Il secondo grafico mostra l'istogramma della distribuzione empirica dei ratio.

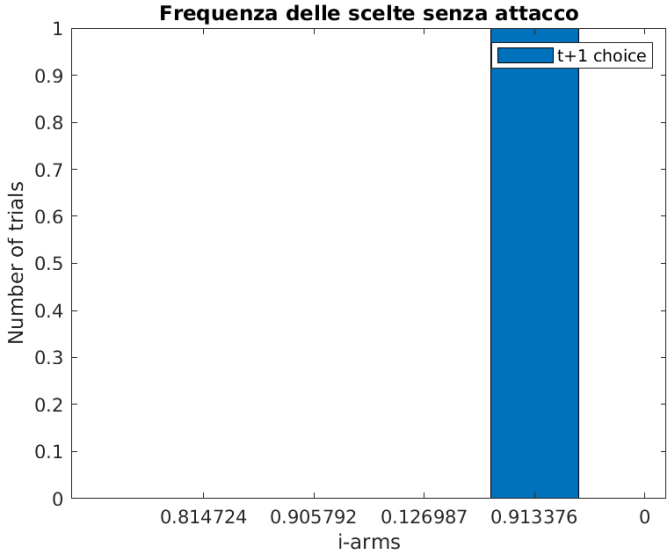

Figure 5: Scelta presa dall'algoritmo ϵ-greedy al tempo $t = T + 10$ se non fosse stato eseguito nessun attacco.

3.2.2 Upper Confident Bound (UCB)

In questa sezione viene descritta la strategia di attacco ad un problema MAB che sfrutta l'algoritmo UCB. Il funzionamento di UCB è simile a ϵ-greedy ed è descritto dall'equazione 5 . Un vantaggio di UCB rispetto ad ϵ-greedy è che non presenta nessun tipo di casualità per cui la strategia di attacco è valida per ogni tolleranza δ (anche 0). Anche l'impostazione del problema di minimizzazione è molto simile, infatti si tratta solo di sistemare gli assi temporali nei vincoli, il nuovo problema di minimizzazione P_2 derivato dal problema P in 16 è:

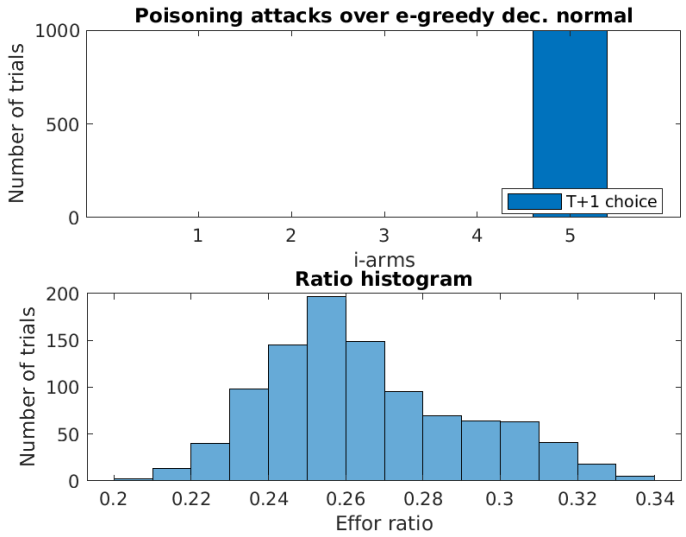

Figure 6: Performance attacco sul problema MAB-(-greedy decrescente $\epsilon_t = 1/t$) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T + 1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° $\mu_{a_5} = 0$). Il secondo grafico mostra l'istogramma della distribuzione empirica dei ratio.

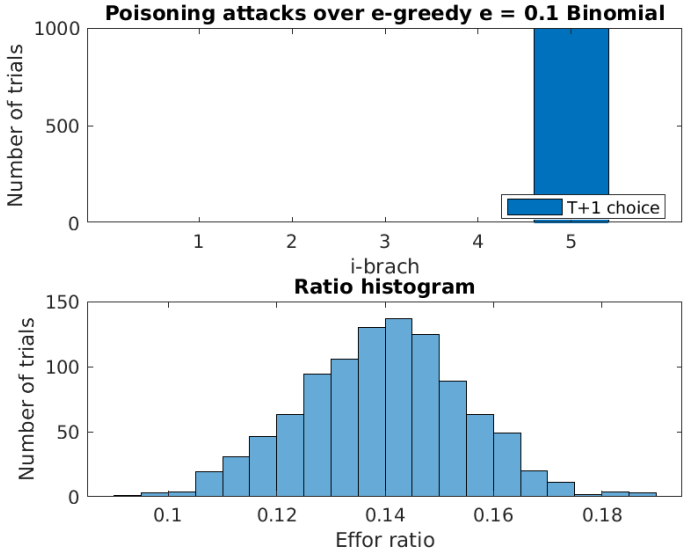

Figure 7: Performance attacco sul problema MAB-(-greedy) con distribuzione binomiale dei reward con probabilità di successo μ_a. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T+1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° $\mu_{a_5} = 0$). L'istogramma della distribuzione empirica dei ratio.

$$P_2 := min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2, \text{ con vincoli: } \left\{ \tilde{\mu}_a^*(T+1) \geq \tilde{\mu}_a(T+1) + \xi \qquad \forall a \neq a^* \right. \tag{32}$$

Anche in questo caso lo studio sui vincoli segue quanto visto per il caso di ϵ-greedy, l'unica differenza che incorre risiede nel vettore b dei termini noti, infatti:

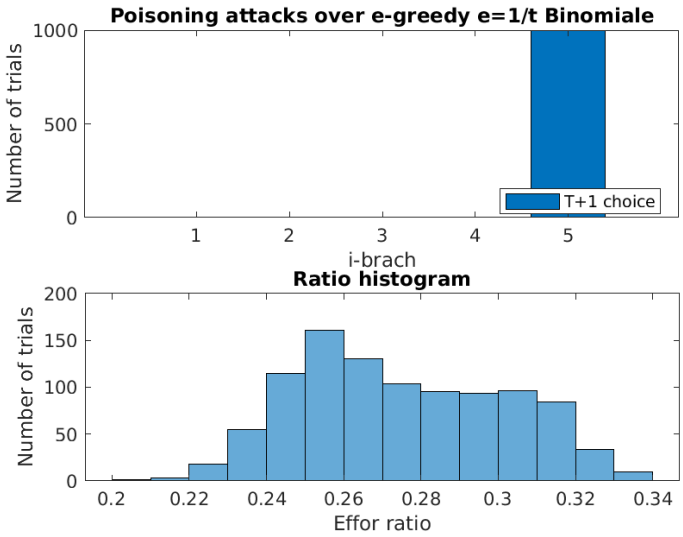

Figure 8: Performance attacco sul problema MAB-(-greedy decrescente, $\epsilon = 1/t$) con distribuzione binomiale dei reward con probabilità di successo μ_{a}. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T + 1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° μ_{a_5} = 0). L'istogramma della distribuzione empirica dei ratio.

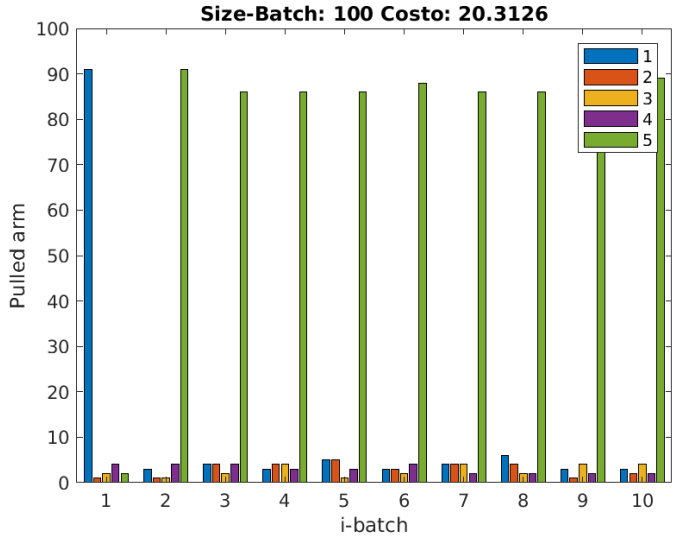

Figure 9: Performance attacco sul problema MAB-(-greedy) con distribuzione degli errori $N(0, \sigma^2)$. Il grafico mostra la frequenza di selezione per le K scelte all'aumentare dei batch processati.

$$\tilde{\mu}_a(T+1) = avg(a) + \frac{1}{m_a}\epsilon I(a_t = a) + 3\sigma\sqrt{\frac{log(t)}{m_a}} \tag{33}$$

per cui il vincolo può essere scritto come:

$$\forall a \neq a^* : avg(a^*) + \frac{1}{m_{a^*}}\epsilon I(a_t = a^*) + 3\sigma\sqrt{\frac{log(t)}{m_{a^*}}} \geq avg(a) + \frac{1}{m_a}\epsilon I(a_t = a) + 3\sigma\sqrt{\frac{log(t)}{m_a}} + \xi \tag{34}$$

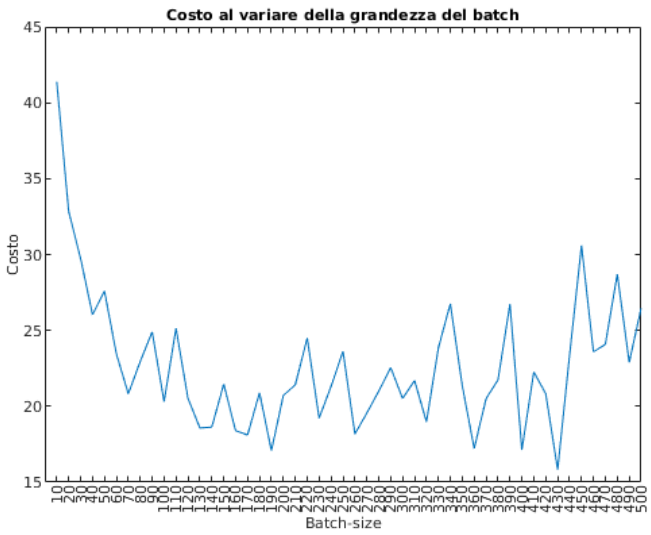

Figure 10: Performance attacco sul problema MAB-(-greedy) con distribuzione degli errori $N(0, \sigma^2)$. Il grafico mostra il costo dell'attacco (calcolato come norma-l^{22}) al variare della grandezza dei batch.

$$\forall a \neq a^* : \frac{1}{m_a}\epsilon I(a_t = a) - \frac{1}{m_{a^*}}\epsilon I(a_t = a^*) \leq avg(a^*) - avg(a) + 3\sigma\sqrt{\frac{log(t)}{m_{a^*}}} - 3\sigma\sqrt{\frac{log(t)}{m_a}} - \xi \tag{35}$$

per cui la matrice A rimane uguale a 29, mentre il vettore dei termini noti b diventa:

$$b = \begin{matrix} a_1 \\ \vdots \\ a_{K-1} \end{matrix} \begin{pmatrix} avg(a^*) - avg(a) + 3\sigma\sqrt{\frac{log(t)}{m_{a^*}}} - 3\sigma\sqrt{\frac{log(t)}{m_a}} - \xi \\ \vdots \\ avg(a^*) - avg(a) + 3\sigma\sqrt{\frac{log(t)}{m_{a^*}}} - 3\sigma\sqrt{\frac{log(t)}{m_a}} - \xi \end{pmatrix} \tag{36}$$

Anche in questo caso possiamo formulare il seguente teorema, analogo a quanto visto per ϵ-greedy: **Teorema 2.** Per ogni $\xi > 0$, per ogni istanza $\{\overline{y}_a\}_{a \in A}$ esiste almeno una soluzione ottima per P_2 che sia lineare nei vincoli. Per cui, dalla proposizione 1, esiste una strategia di attacco ottima per UCB.

In Figura 11 il grafico delle performance, come mostra la frequenza delle risposte, per tutti i 1000 trails la scelta al $t = T+1$ è sempre quella targhet $a^* = (K = 5)$. Inoltre il ratio rimane relativamente basso fermandosi al 2.5%. I risultati sono comparabili a quanto ottenuto dal paper in analisi.

3.2.3 Batch Succesive Eliminatios (BaSE)

Questo algoritmo è una variate recente all'algoritmo *Successive Eliminatios* proposta in [GHRZ19]. L'idea è quella di esplorare i primi M-1 batch per poi selezionare durante l'ultimo batch la scelta ottima. Durante l'esplorazione degli M-1 batch vengono rimosse le scelte per i quali si prevede una scarsa probabilità di ottenere un reward. L'analisi degli autori si focalizza propio sulla scelta della soglia ottima oltre il quale un braccio può essere considerato come non più promettente e quindi rimuoverlo dalle future esplorazioni. Quello che si dimostra da un punto di vista algebrico è che la soglia oltre il quale rimuovere un scelta può essere espresso come differenza tra $max(\tilde{\mu}_a) - \tilde{\mu}_a, \forall a \in A$ è:

$$\max_{a \in A}(\tilde{\mu}_a) - \tilde{\mu}_a \geq \sqrt{\frac{\gamma log(KT)}{N_a(t)}} \rightarrow A := A - \{a\} \tag{37}$$

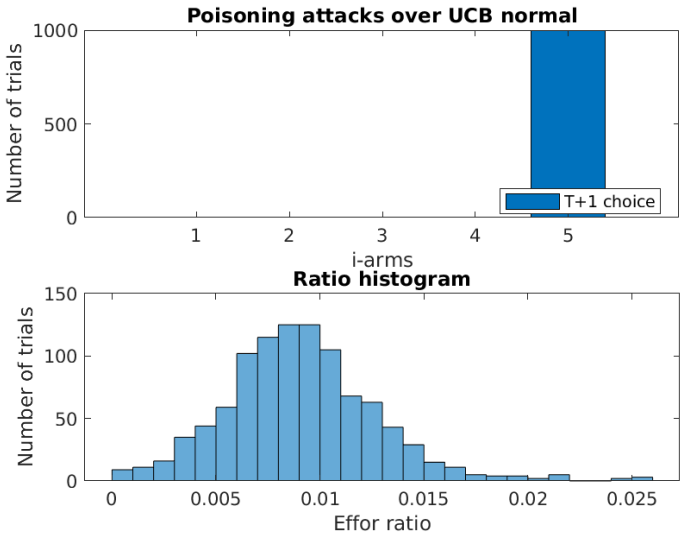

Figure 11: Performance attacco sul problema MAB-(UCB) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T + 1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° μ_{a_5} = 0). Il secondo grafico mostra l'istogramma della distribuzione empirica dei ratio.

le medie empiriche $\tilde{\mu}_a$ vengono stimate dopo aver esplorato ogni batch. Durante l'ultimo batch verrà selezionata solo la scelta $a \in A$ tale che $a = arg \max_{a \in A} \tilde{\mu_a}$

Abbiamo già introdotto un esempio di ϵ-greedy con dimensione dei batch $M < T$, il flusso delle informazioni anche in BaSE segue quanto già visto: l'agente esegue il primo batch di prove, ovvero tira per lo stesso numero di volte le K scelte possibili, a questo punto l'ambiente genera un vettore di M prove. Prima che i reward siano inviati all'agente, vengono manipolati dall'attaccante e poi inviati all'agente.

Poiché anche in questo caso si tratta di un algoritmo basato sulla stima empirica del valore atteso dei reward possiamo studiare un algoritmo di attacco con la metodologia vista per ϵ-greedy ed UCB. Per ulteriori dettagli sull'algoritmo BaSE e la scelta della soglia, si rimanda al paper originale.

Il valore atteso dei reward è calcolato per ogni batch, per cui l'agente calcola la media dei valori attesi per selezionare la scelta ottima. Definiamo $\tilde{\mu}_a$ come:

$$\tilde{\mu}_a = \tilde{\mu}_a(m-1)\rho + \tilde{\mu}_a(m)(1-\rho) \text{ con: } m \in M, \rho = \frac{N_a(m-1)}{N_a(m-1) + N_a(m)} \tag{38}$$

ovvero, il valore atteso è una media ponderata tra il valore atteso calcolato durante il batch precedente e il valore atteso calcolato durante questo batch. Se adoperiamo l'attacco, ai reward si aggiunge un termine di manipolazioni per cui, dopo aver valutato M-1 batch possiamo scrivere:

$$\tilde{\mu}_a = avg(a) + \frac{1}{N_a(1)}\epsilon_a(1)I(a(1)_t = a)\rho^{M-1} + \sum_{m=1}^{M-1} \frac{1}{N_a(m)}\epsilon_a(m)I(a(m)_t = a)(1-\rho)\rho^{M-m} \tag{39}$$

dove $N_a(b), \epsilon(b), a(b)_t$ indicano rispettivamente il numero di volte che è stata presa la scelta N_a durante il batch b, l'errore introdotto dopo aver valutato il batch b e la scelta al tempo t durante il batch b. Poiché l'attacco viene fatto subito dopo aver valutato un batch, è necessario solo mantenere l'informazione delle medie dei reward solo al tempo precedente.

In prima analisi potremmo semplicemente imporre un vincolo tale per cui

$$\forall a \in A, a \neq a^* : \tilde{\mu}_{a^*} > \tilde{\mu}_a + \sqrt{\frac{\gamma log(KT)}{N_a(t)}} \tag{40}$$

In questo modo quando si valuta la soglia, tutte le possibile scelte verranno scartate e rimarrà solo la scelta a^* come scelta possibile.

Possiamo procedere anche in un altro modo, infatti, non è richiesto che la scelta a^* sia la scelta che abbia il valore massimo dopo ogni valutazione di un batch, l'importante è che sia massimo dopo la valutazione dell' *(M-1)*-esimo batch. In questo modo il costo di attacco potrebbe essere minore. Scriviamo il nuovo problema di minimizzazione:

$$P_{BA} : min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2, \text{ con vincoli: } \begin{cases} \tilde{\mu}_{a_{max}}(t+1) \geq \tilde{\mu}_a(t+1) + \xi, \forall a \neq a^* \\ \tilde{\mu}_{a_{max}}(t+1) - \tilde{\mu}_{a^*}(t+1) < \sqrt{\frac{\gamma log(KT)}{N_{a^*}(t)}} \\ \sum_a a_{max} = 1 \end{cases} \tag{41}$$

Ovvero, il primo vincolo serve per cercare il braccio massimo, una volta selezionato il braccio massimo si deve soddisfare il vincolo imposto dall'algoritmo *BaSE* in modo che il braccio targhet a^* non sia eliminato dal set delle scelte disponibili. L'ultimo vincolo definisce la presenza di solo un braccio massimo a_{max}. Nella pratica risulta difficile da implementare, perchè richiede la possibilità di avere due variabili decisionali, sia gli $\overline{\epsilon_m}$ e sia di poter selezionare il massimo. Le funzioni software di MATLAB per le programmazione quadratica non prevedono (o comunque non è stato trovato il modo) per implementare questo problema. Ciò che è stato fatto è applicare una serie di ipotesi / rilassamenti. In prima istanza è stato implementato il vincolo 40. In pratica è come se a^* sia sempre il massimo rispetto agli altri bracci con una distanza di $\gamma log(KT)/N_a(t)$. Cosi facendo alla fine del primo batch nell'insieme delle scelte possibili rimane solo a^*.

Il secondo ragionamento fatto è stato implementare il seguente vincolo:

$$\forall a \in A, a \neq a^* : \tilde{\mu}_a - \tilde{\mu}_a^* < \sqrt{\frac{\gamma log(KT)}{N_{a^*}(t)}} \tag{42}$$

In questo modo non è necessario che a^* sia massimo. L'implementazione, è stato fatta in modo simile a quanto visto nei capitoli precedenti. Con la differenza che la media dei reward è calcolata come media ponderata.

La matrice A:

$$A = \begin{array}{c} \\ a_1 \\ \vdots \\ a_{K-1} \end{array} \begin{array}{c} \begin{array}{cccc} \epsilon_{(1:m_{a_1})} & \cdots & \cdots & \epsilon_{(m_{a_k}:T)} \end{array} \\ \begin{pmatrix} \frac{(1-\rho_{a_1})}{N_{a1}(t)} & 0 & \cdots & -\frac{(1-\rho_{a^*})}{N_{a^*}(t)} \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & \frac{(1-\rho_{K-1})}{N_{a_{K-1}(t)}} & -\frac{(1-\rho_{a^*})}{N_{a^*}(t)} \end{pmatrix} \end{array} \tag{43}$$

$$b = \begin{array}{c} \\ a_1 \\ \vdots \\ a_{K-1} \end{array} \begin{pmatrix} \sqrt{\frac{\gamma log(KT)}{N_{a^*}(t)}} - avg_{t-1}(a_1)\rho_{a_1} - avg_t(a_1)(1-\rho_{a_1}) + avg_{t-1}(a^*)\rho_{a^*} - avg_t(a^*)(1-\rho_{a^*}) \\ \vdots \\ \sqrt{\frac{\gamma log(KT)}{N_{a^*}(t)}} - avg_{t-1}(a_{K-1})\rho_{a_1} - avg(a_{K-1})(t)(1-\rho_{a_{K-1}}) + avg_{t-1}(a^*)\rho_{a^*} - avg_t(a^*)(1-\rho_{a^*}) \end{pmatrix} \tag{44}$$

mentre la matrice C è rimasta uguale a 31. Il risultato è stato che il braccio targhet a^* non veniva mai rimosso dal gruppo delle possibili scelte ma non rimaneva l'unica scelta possibile, quindi possiamo dire che la strategia di attacco prodotta dal vincolo 42 è fallimentare. Il problema risiede nel fatto che a^* non riesce mai a diventare il braccio con μ_a più alta. Per cui è possibile trovare una strategia di attacco con costo minore implementando 41.

3.2.4 Thompson-Sampling (TS)

Gli algoritmi specifici visti precedentemente hanno una struttura molto simile e possono essere generalizzati a molte altre varianti perchè si basano sull'osservazione empirica della media dei reward. E' logico chiedersi se algoritmi basati sulla stima Bayesiana siano robusti a perturbazioni sui reward. Un'importate contributo degli autori del paper in analisi è stato mostrare come anche l'algoritmo Bayesiano Thompson-Sampling con distribuzione Gaussiana come distribuzione di supporto sia vulnerabile ad attacchi. L'algoritmo TS è spiegato in 6, ricordiamo che θ_a ha distribuzione $\sim N(\hat{u_a}(T)/\sigma^2, \sigma^2/m_a)$

Procediamo quindi alla definizione del problema, abbiamo che:

$$\mathbb{P}(a_{T+1} \neq a^*) = \mathbb{P}(\cup_{a \neq a^*} \theta_{a^*} \leq \theta_a) \tag{45}$$

ovvero, la probabilità che la scelta all'istante $t = T + 1$ sia a^* è uguale all'unione delle probabilità per cui $\theta_{a^*} \leq \theta_a$. Se scompattiamo l'unione delle probabilità tramite il teorema sulle probabilità totale, la potremmo vedere come:

$$\mathbb{P}(a_{T+1} \neq a^*) = \mathbb{P}(\theta_{a^*} \leq \theta_{a_1}) \cup \mathbb{P}(\theta_{a^*} \leq \theta_{a_2}) ... \cup \mathbb{P}(\theta_{a^*} \leq \theta_{aK-1}) - \mathbb{P}(\theta_{a^*} \leq \theta_a \cap \theta_{a^*} \leq \theta_{a_1}) ... \tag{46}$$

in altre parole è sufficiente che θ_{a^*} sia più piccolo di un solo θ_a perchè a^* non sia scelto all'istante $t = T + 1$. In verità l'equazione dell'unione di n eventi si complica di molto, introducendo diversi termini di sommatoria. Per rendere l'equazione più semplice dal punto di vista dell'analisi, maggioriamo il secondo membro, per cui:

$$\mathbb{P}(\cup_{a \neq a^*} \theta_{a^*} \leq \theta_a) \leq \sum_{a \neq a^*} \mathbb{P}(\theta_{a^*} \leq \theta_a) \tag{47}$$

in pratica, al termine a destra dell'uguale, stiamo lasciando solo la sommatoria degli eventi come se fossero indipendenti, infatti non stiamo sottraendo l'intersezione delle probabilità nel caso in cui siano veri entrambi gli argomenti. In questo modo stiamo maggiorando il secondo termine, andando a creare un upper bound. Analizziamo ora i singoli termini della sommatoria $\mathbb{P}(\theta_{a^*} \leq \theta_a)$:

$$\mathbb{P}(\theta_{a^*} \leq \theta_a) = \mathbb{P}(\theta_{a^*} - \theta_a \leq 0) \text{ con: } (\theta_{a^*} - \theta_a) \sim N(\mu_{a^*} - \mu_a, \sigma_a^{2*} + \sigma_a^2) \tag{48}$$

Sia Φ(x) la funzione di ripartizione di una normale $N(0, 1)$, per cui (ricordiamo che $\Phi(z < t) = \Phi(\frac{t-\mu_z}{\sigma_z})$):

$$\mathbb{P}(\theta_{a^*} - \theta_a \leq 0) = \Phi\left(\frac{0 - (\mu_{a^*} - \mu_a)}{\sqrt{\sigma_a^{2*} + \sigma_a^2}}\right) \tag{49}$$

Sapendo che $\theta_a \sim N(\frac{\tilde{\mu}_a(T)}{\sigma^2}, \frac{\sigma^2}{m_a})$:

$$\mathbb{P}(\theta_{a^*} - \theta_a \leq 0) = \Phi\left(\frac{\tilde{\mu}_a(T) - \tilde{\mu}_a^*(T)}{\sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}}}\right) \tag{50}$$

Per cui possiamo impostare il problema di minimizzazione P_3 derivato dal problema P in 16 come:

$$P_3 := min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2 \text{ vincoli: } \begin{cases} \sum_{a \neq a^*} \mathbb{P}(\theta_{a^*} \leq \theta_a) = \sum_{a \neq a^*} \Phi\left(\frac{\tilde{\mu}_a(T) - \tilde{\mu}_a^*(T)}{\sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}}}\right) \leq \delta & \forall a \neq a^* \\ \tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) \leq 0 \end{cases} \tag{51}$$

L'idea è quindi quella di fissare un δ basso, in modo che la probabilità di non scegliere a^* sia bassa. L'ultima condizione risulta spesso ridondante in quanto se $\delta \leq 0.5$, dalle proprietà della funzione di

ripartizione sappiamo che l'argomento di $\Phi(\cdot)$ è negativo, per cui $\tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) \leq 0$ e il secondo vincolo dell'equazione 51 risulta già essere soddisfatto.

E' necessario verificare che i vincoli siano convessi, per far ciò sfruttiamo il lemma 2 qui riportato:
Lemma 2. Data una qualsiasi costante $C_i > 0$ per ogni $i < K$, la funzione $f(\overline{x}) = \sum_{i=1}^{K-1} \Phi(C_i x_i - C_i x_k)$ è convessa nel dominio $D = (\overline{x} \in R^K | x_i - x_k \leq 0, \forall i < K)$.

Per dimostrare il Lemma 2., si calcola la matrice Hessiana della funzione $f(x)$, dopodiché si verifica che la matrice sia semidefinita positiva. Possiamo quindi formulare il seguente teorema:

Teorema 3. Per ogni $\delta > 0$, per ogni istanza $\{\overline{y}_a\}_{a \in A}$ esiste almeno una soluzione ottima per P_3 che sia lineare nei vincoli. Per cui, dalla proposizione 1, esiste una strategia di attacco ottima per TS. Anche in questo caso la dimostrazione è analoga a quanto visto per ϵ-greedy e UCB.

Possiamo rilassare ulteriormente il problema P_3, infatti:

$$\sum_{K-1} \Phi(x) \leq \delta \rightarrow \Phi(x) \leq \frac{\delta}{K-1} \tag{52}$$

da cui $x \leq \Phi^{-1}(\delta/(K-1)) = cost$. Se fisso $\delta/(K-1) \leq 0.5$ allora posso omettere il secondo vincolo del problema 51, per cui $\delta \leq (K-1)/2$.

Adoperando il rilassamento descritto al vincolo del problema P_3 posso scrivere:

$$\left(\frac{\tilde{\mu}_a(T) - \tilde{\mu}_a^*(T)}{\sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}}} \right) \leq \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \tag{53}$$

$$\tilde{\mu}_a(T) - \tilde{\mu}_a^*(T) \leq \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \tag{54}$$

Il termine a destra dell'equazione 54 risulta essere una costante perchè operazioni di termini noti all'attaccante. Possiamo quindi formulare il nuovo problema rilassato P_4 come:

$$P_4 := min \sum_{a \in A} \|\overline{\epsilon_a}\|_2^2 \text{ vincoli: } \left\{ \forall a \neq a^* : \tilde{\mu}_a(T) - \tilde{\mu}_a^*(T) \leq \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \right. \quad \forall a \neq a^* \tag{55}$$

con: $\sigma \leq \frac{K-1}{2}$

Il problema P_4 non porta alla soluzione ottima formalizzata dal problema P, questo perchè è stato minimizzato un upper bound e non una quantità certa. Dai risultati numerici elaborati si osserva che la soluzione al problema P_4 porta il decisore a scegliere il braccio targhet ma lo fa con un costo comparabile a quello P_3 implementato dagli autori del paper. Operando gli stessi passaggi fatti anche per i casi ϵ-greedy ed UCB possiamo riscrivere il vincolo come

$$\forall a \in A : a \neq a^* : \frac{1}{m_a} I(a_t = a) - \frac{1}{m_{a^*}} I(a_t = a*) \leq avg(a^*) - avg(a) + \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \tag{56}$$

per cui la matrice A rimane invariata (29) a quanto visto nei precedenti capitoli, mentre il vettore dei termini noti b diventa:

$$b = \begin{matrix} a_1 \\ \vdots \\ a_{K-1} \end{matrix} \begin{pmatrix} avg(a^*) - avg(a) + \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \\ \vdots \\ avg(a^*) - avg(a) + \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1} \left(\frac{\sigma}{K-1} \right) \end{pmatrix} \tag{57}$$

In figura 12 vengono riportati i risultati numeri ottenuti, come per i risultati ottenuti dal paper in analisi il ratio massimo è di 4.5% e il braccio targhet è sempre selezionato.

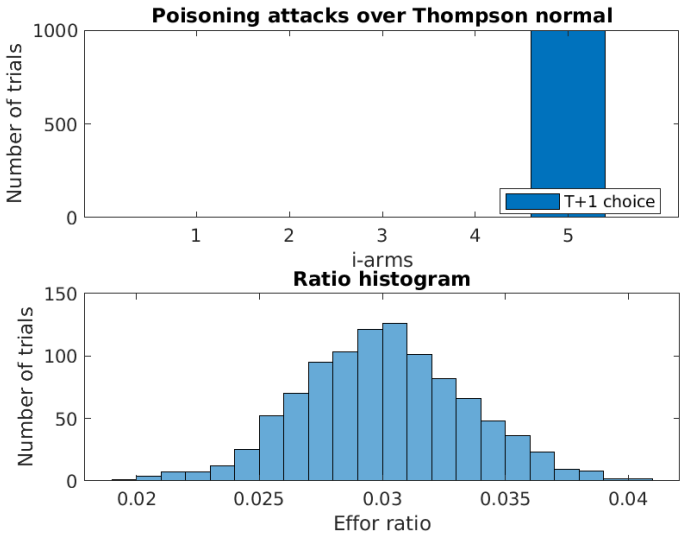

Figure 12: Performance attacco sul problema MAB-(TS) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra la frequenza dei bracci scelti all'istante $t = T + 1$, in questo caso viene scelto sempre il braccio target dell'attacco (il 5° μ_{a_5} = 0). Il secondo grafico mostra l'istogramma della distribuzione empirica dei ratio.

3.3 Online

Gli algoritmi online, come specificato nei precedenti paragrafi aggiornano la propria conoscenza ad ogni turno. In questa configurazione l'attaccante monitora costantemente la decisione presa a_t e manipola il reward aggiungendoci un errore additivo di manipolazione $\epsilon_t \in \mathbb{R}$. Il reward ottenuto dall'agente è $r'_t = r_t + \epsilon_t$. Anche in questo caso l'attaccante vuole manipolare i dati il meno possibile per non essere scoperto. Misuriamo il costo di attacco con la norma-l^1 ovvero:

$$C(t) = \sum_t |\epsilon_t| \tag{58}$$

L'obiettivo dell'attaccante è forzare l'agente a scegliere la scelta sub-ottima a^* con il $C(t)$ più piccolo possibile.

3.3.1 Oracolo

Supponiamo che l'algoritmo conosca il reward atteso, in tal caso può operare la seguente strategia:

$$\epsilon_t = -I(a_t \neq a^*) \max(0, \mu_{a_t} - \mu_{a^*} + \xi) \tag{59}$$

in questo modo avremo che:

$$\forall a \neq a^* \in A : \mu'_{a_t} = \mu_{a_t} + \epsilon_t = \begin{cases} \mu_{a_t} - \mu_{a_t} + \mu_{a^*} - \xi = \mu_{a^*} - \xi \\ \mu_{a_t} - 0 \end{cases} \tag{60}$$

Nel primo caso avremo che $\forall \mu_{a_t\, a \neq a^*} : \mu_{a_t} < \mu_{a_t^*}, \forall \xi > 0$, risulta quindi che $arg\max_{a \in A} \mu_a = a^*$ per cui a* è il braccio scelto. Nel secondo caso il reward non viene modificato, ed è corretto perchè risulta che $\mu_{a^*} \geq \mu_a + \xi$ per cui non è necessario modificare i reward. Da quanto mostrato in equazione 60 risulta che non è necessario compiere un attacco nel caso in cui $a_t = a^*$ (sembra anche una scelta ragionevole).

Nella realtà l'attaccante non conosce $\{\ \mu_a\ \}_{a \in A}$ per cui può indovinare un limite superiore $\{\ C_a\ \}_{a \neq a^*}$ sulla quantità $\max(0, \mu_a - \mu_{a^*})$. In questo caso la strategia di attacco tramite la costante C_a potrebbe essere costruita ottimizzando la seguente quantità:

$$\epsilon_t = -I(a_t \neq a^*)C_{a_t} \tag{61}$$

ovviamente deve essere : $(\forall a \in A, a \neq a^*$ se $\mu_{a^*} < \mu_a$ allora: $C_{a_t} \geq 0)$. Possiamo formulare quanto detto nella seguente proposizione:

Proposizione 4. L'attacco tramite la costate $\{\mathrm{C_a}\}_{\mathrm{a} \neq \mathrm{a}^*}$ ha successo se e solo se $C_a > max(0, \mu_a - \mu_{a^*}), \forall a \neq a^*$.

Supponiamo infatti che esista un braccio $i \neq a^*$ tale per cui $C_i \leq max(0, \mu_i - \mu_{a^*})$, in questo caso il braccio i risulta essere ottimo rispetto al braccio a^* infatti avremo che: $\mu_i - C_i \geq \mu_i - (\mu_i - \mu_{a^*}) \rightarrow \mu_i - C_i > \mu_{a^*}$. Nel caso in cui invece $C_i = max(0, \mu_i - \mu_{a^*})$ avremo che il braccio i e i braccio a^* avranno lo stesso valore atteso $\mu_i = \mu_{a^*}$ per cui entrambi saranno ottimi. In entrambi i casi, l'attacco non ha successo.

Dalla proposizione 4 si evince come l'attaccante debba conoscere le quantità ignote μ_a per costruire una strategia di attacco. Inolte l'attacco tramite costante risulta non adattativo all'istanza del problema per cui la quantità C_a potrebbe risultare molto più grande di $\mu_a - \mu_{a^*}$ per cui l'attaccante paga un costo molto più grande del necessario. Gli autori propongono quindi una strategia di attacco che sia adattativa all'istanza del problema MAB.

3.3.2 Attacco con Costante Stimata (ACE)

L'idea è che l'attaccante aggiorni le quantità $\mu_a - \mu_{a^*}$ in base ad una stima empirica della media di μ_a. Infatti l'attaccante osserva le tuple (a_r, r_t) per cui è in grado di calcolare il valore atteso $\mathbb{E}[r_t] = \tilde{\mu}_a(t)$ (unbiased, non distorto),

$$\tilde{\mu}_a(t) = \frac{1}{N_a(t)} \sum_1^t r_t I(a_t = a) \tag{62}$$

Dato $\delta \in (0, 1)$, definiamo la funzione $\beta(n)$ come:

$$\beta(n) = \sqrt{\frac{2\sigma^2}{n} log \frac{\pi^2 K n^2}{3\delta}} \tag{63}$$

tale che, possiamo definire l'evento E come:

$$E = (\forall a \in A, \forall t : |\tilde{\mu}_a(t) - \mu_a| \leq \beta(N_a(t)) \tag{64}$$

ovvero $\beta(\cdot)$ è il limite superiore della differenza tra la media stimata e la media ignota. Osserviamo come $\beta(\cdot)$ sia valida per qualunque t.

Lemma 1. *(Lemma 1 di [JLMZ18])* Per $\delta \in (0, 1), \mathbb{P}(E) \geq 1 - \delta$

Il risultato del Lemma 1 mostra come con una probabilità di $1 - \delta$:

$$\mu_a = \tilde{\mu}_a(t) \pm \beta(N_a(t)) \tag{65}$$

per cui abbiamo che:

$$\mathbb{P} = 1 - \delta : \tilde{\mu}_a(t) - \beta(N_a(t)) < \mu_a < \tilde{\mu}_a(t) + \beta(N_a(t)) \tag{66}$$

$$\mathbb{P} = 1 - \delta : \tilde{\mu}_{a^*}(t) - \beta(N_{a^*}(t)) < \mu_{a^*} < \tilde{\mu}_{a^*}(t) + \beta(N_{a^*}(t)) \tag{67}$$

da cui possiamo ricavare che (sottrarre i minimi con i massimi e viceversa):

$$\tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) - \beta(N_a(t)) - \beta(N_{a^*}(t)) < \mu_a - \mu_{a^*} < \tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) + \beta(N_a(t)) + \beta(N_{a^*}(t)) \tag{68}$$

$$\mu_a - \mu_{a^*} \in [\tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) \pm (\beta(N_a(t)) + \beta(N_{a^*}(t)))] \tag{69}$$

Sapendo che $C_a(t)$ è definito come il limite superiore allora possiamo ottimizzare il seguente attacco:

$$\epsilon_t = -I(a_t \neq a^*)C_{a_t} = -I(a_t \neq a^*)max(0, \tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) + \beta(N_a(t)) + \beta(N_{a^*}(t)) \quad (70)$$

Teorema 4. Dato un qualunque $\delta \in (0, 1/2)$, assumiamo che l'algoritmo del bandito raggiunga in $O(logT)$ il limite superiore con probabilità $1 - \delta$. Allora, con probabilità $1 - 2\delta$, l'attacco ACE forza l'algoritmo a scegliere il braccio a^* per $N_{a^*}(T)$ volte con $N_{a^*}(T) = T - o(T)$

3.3.3 Simulazioni numeriche:

Per le simulazioni numeriche sono stati utilizzati gli stessi parametri del paper, in particolare rumore normale con σ = 0.1, K è diverso dal paper, nel nostra caso $K = 5$, il braccio (scelta) target dell'attaccante ha μ_{a^*} = 0 e si trova nella posizione $a_5, k = 5$, mentre $\forall$ a $\in$ A, a $\neq$ a* , μ_a è estratta casualmente da una distribuzione uniforme $\sim U(0, 1)$. Con questa configurazione il braccio targhet non dovrebbe mai essere scelto se non fosse presente l'attaccante. δ = 0.05 e T = 10^4 time-step.

Commento Risultati: Per come è stata impostata l'analisi, i risultati attesi dovrebbero essere molto simili a quanto mostrato dal paper. Il primo risultato che commenteremo riguarda l'attacco ACE su ϵ-greedy e la variante ϵ-greedy decrescente. Come si vede in Figura 13 il costo cumulato rispecchia quanto riportato in [Figura 4.a, pp 8] con attenzione alla scala utilizzata. Infatti per ϵ-greedy decrescente il costo dell'attacco cresce in modo rapido fino all'istante 10, dopodiché continua a cresce linearmente. Viceversa per ϵ-greedy con $\epsilon = 0.1$ la crescita dopo l'istante $t = 10$ continua in modo esponenziale e questo è rappresentato dalla retta sul grafico con assi logaritmici. In entrambi i casi quello che si osserva è che il decisore, inconsapevole di essere vittima dell'attacco, seleziona il braccio a^* come braccio ottimo. Leggera differenza nelle due strategie sta nel fatto che $\lim_{t\to\infty} 1/t \sim 0$ per cui dopo un certo t la probabilità che il decisore prenda una scelta casuale è praticamente nulla per cui prende sempre la scelta che ritiene ottima, in questo caso a^*, se non avessimo fatto l'attacco invece, non avrebbe mai scelto $a*$. Utilizzando ϵ-greedy con $\epsilon = 0.1$ costante nel tempo avremo che il 9% delle volte il decisore sceglierà in modo casuale una tra le K possibili scelte. Dal grafico si vede molto bene come per il 91% delle volte circa il braccio scelto sia a^* mentre il restante 10% è disperso tra le altre possibili $K - 1$ scelte. Anche in questo caso se non avessi fatto l'attacco il decisore non avrebbe mai scelto a^* se non con una probabilità inferiore del 2%. Per la variante ϵ-greedy riportiamo anche la variante in cui il reward sia distribuito come una binomiale con probabilità di successo $\overline{p} = \mu_a, \forall a \in A$ mostrata in Figura 14. Anche in questo caso il braccio scelto per quasi la totalità dei time-step a^*. Questo a indice dell'universalità della strategia ACE.

In Figura 15 le performance ottenute con ACE su un problema MAB che sfrutta UBC. Anche in questo caso i risultati ottenuti sono sovrapponibili con quanto riportato in [Figura 4.b, pp 8], il costo cumulato di attacco sembra crescere in modo esponenziale. Anche in questo caso il braccio targhet è scelto per quasi la totalità dei time-step.

Per ultimo, in Figura 16 le performance ottenute con ACE su un problema che utilizza Thompson Sampling. Qui possiamo notare come la funzione di costo mantenga la stessa forma ma con valori assoluti diversi. Questo perchè, in modo diverso da quanto fatto nel paper di riferimento è stato implementato il problema P_4 frutto di diversi rilassamenti al problema originale P. Nel paper a seguito di una soluzione trovata, viene eseguita la ricerca ottima.

4 Conclusione e Considerazioni

Gli autori forniscono una metodologia per implementare una possibile strategia di attacco a MAB nella sua versione offline semplice ed elegante, basata sull'ottimizzazione convessa di un problema quadratico. Per la formulazione online del problema MAB, operano per la costruzione di un algoritmo generico che possa adattarsi all'istanza del problema e dimostrano che al momento non ci siano algoritmi conosciuti che siano immuni a questa strategia. Questo articolo potrebbe essere quindi la base su cui costruire o aprire nuove strade per la ricerca di algoritmi sicuri. Le simulazioni numeriche fatte rispecchiano quanto fatto dagli autori.

Il lavoro di attività di analisi sul paper è stato molto stimolante e, per capirne appieno le potenzialità sono stati ricavati molti dei passaggi matematici mostrati per poi essere implementati al calcolatore.

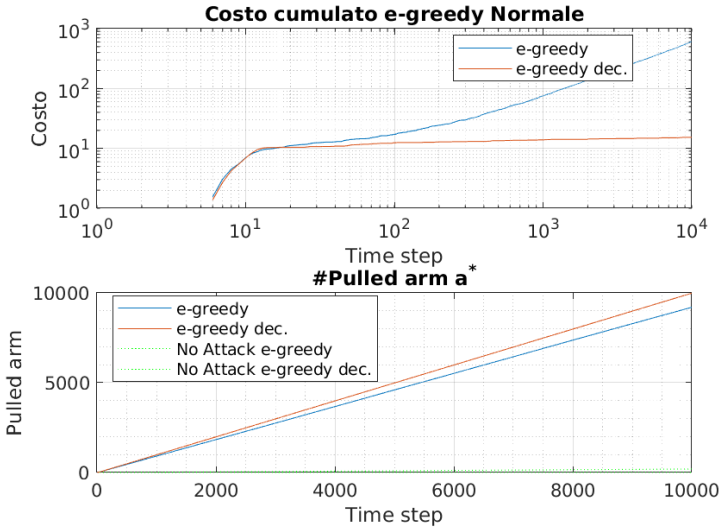

Figure 13: Performance attacco sul problema MAB-(ϵ-greedy) in due varianti, con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra l'errore cumulato in funzione di t. Il secondo grafico il numero delle volte in cui il decisore sceglie il braccio sub-ottimo $a^*(\mu_{a^*} = 0)$ in funzione del tempo t

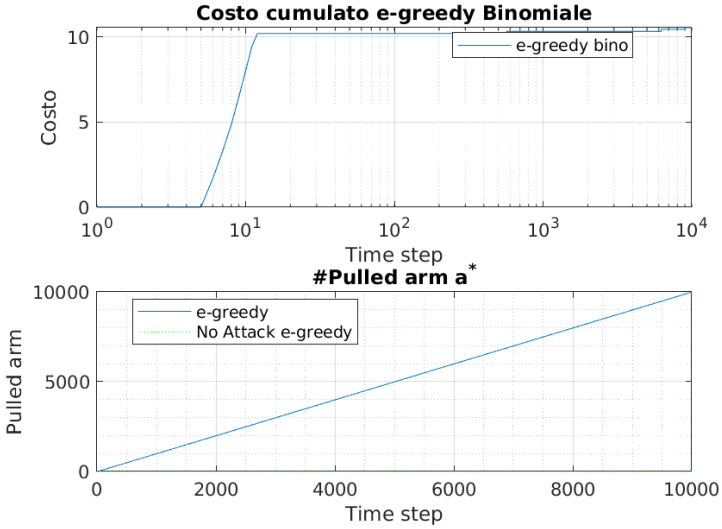

Figure 14: Performance attacco sul problema MAB-(-greedy) con distribuzione dei reward $B(\mu)$. Il primo grafico mostra l'errore cumulato in funzione di t. Il secondo grafico il numero delle volte in cui il decisore sceglie il braccio sub-ottimo $a^*(\mu_{a^*} = 0)$ in funzione del tempo t

L'attività è parsa subito interessante in quanto mi ha permesso di esplorare il settore degli algoritmi di apprendimento per *Reinforcement Learning* molto popolari a livello industriale e sopratutto ha dato luce al problema degli attacchi ai sistemi di apprendimento tramite *avvelenamento dei dati* di cui non avevo mai sentito parlare ne letto nulla. Va detto, che in alcuni punti, soprattutto di [KKM13] e [MJLZ18] è stato difficile interpretare quando detto dagli autori per la mancanza di strumenti matematici adeguati per affrontare i passaggi mostrati. Un altro aspetto particolarmente interessante è stato vedere come vengono "creati" gli algoritmi, che seppure semplici (come in questo caso, non più di 30 righe di codice) godono di una rigorosa trattazione teorica che ne definisce dapprima le caratteristiche ed i limiti dimostrati poi tramite simulazioni numeriche.

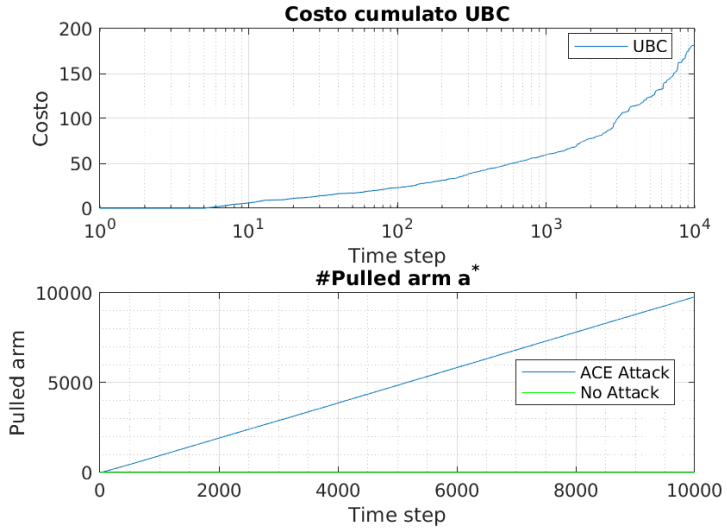

Figure 15: Performance attacco sul problema MAB-(UBC) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra l'errore cumulato in funzione di t. Il secondo grafico il numero delle volte in cui il decisore sceglie il braccio sub-ottimo $a^*(\mu_{a^*} = 0)$ in funzione del tempo t

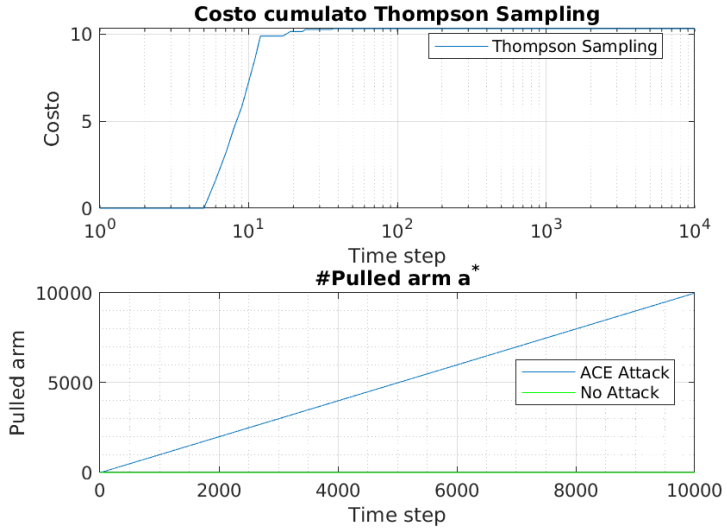

Figure 16: Performance attacco sul problema MAB-(TS) con distribuzione degli errori $N(0, \sigma^2)$. Il primo grafico mostra l'errore cumulato in funzione di t. Il secondo grafico il numero delle volte in cui il decisore sceglie il braccio sub-ottimo $a^*(\mu_{a^*} = 0)$ in funzione del tempo t

5 Bibliografia e testi consultati

[AZB16] Scott Alfeld, Xiaojin Zhu e Paul Barford: Data poisoning attacks against autoregressive models. n *AAAI*, pp. 1452–1458, 2016.

[BNL12] Battista Biggio, Blaine Nelson e Pavel Laskov: Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 1467–1474, 2012.

[CMR15] Olivier Chapelle, Eren Manavoglu e Rómer E Rosales: Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2015.

[GHRZ19] Zijun Gao, Yanjun Han, Zhimei Ren e Zhengqing Zhou: Batched Multi-armed Bandits Problem. *arXiv preprint 1904. 01763*, 2019.

[GLK16] Aurélien Garivier, Emilie Kaufmann e Tor Lattimore: On explore-then-commit strategies. In *Advances in Neural Informa-tion Processing Systems*, pp. 784–792, 2016.

[GSS15] Ian J. Goodfellow, Jonathon Shlens e Christian Szegedy: Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[GKT19] Anupam Gupta, Tomer Koren e Kunal Talwar: Better algorithms for stochastic bandits with adversarial corruptions. *arXiv preprint 1902. 08647*, 2019.

[HPGDA17] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan e Pieter Abbeel: Adversarial attacks on neural network policies. *arXiv preprint 1702. 02284* , 2017.

[JLMZ18] Kwang-Sung Jun, Lihong Li, Yuzhe Ma e Xiaojin Zhu: Adversarial attacks on stochastic bandits. *arXiv preprint 1810. 12188* , 2018.

[KKM13]Nathaniel Korda, Emilie Kaufmann e Remi Munos : Thompson sampling for 1-dimensional exponential family bandits. In *Advances in Neural Information Processing Systems*, pp. 1448–1456, 2013.

[LWSV16] Bo Li, Yining Wang, Aarti Singh e Yevgeniy Vorobeychik: Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems*, pp. 1885–1893, 2016.

[LCLS10] Lihong Li, Wei Chu, John Langford e Robert E. Schapire: A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670. ACM,2010.

[LHLSLS17] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu e Min Sun: Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint 1703.06748* , 2017.

[LWBSU18] Fang Liu, Sinong Wang, Swapna Buccapatnam e Ness Shroff: Ucboost, a boosting approach to tame complexity and optimality for stochastic bandits. *arXiv preprint 1804. 05929* , 2018.

[LMP18] Thodoris Lykouris, Vahab Mirrokni e Renato Paes Leme: Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 114–122. ACM, 2018.

[MJLZ18] Yuzhe Ma, Kwang-Sung Jun, Lihong Li e Xiaojin Zhu: Data poisoning attacks in contextual bandits. *arXiv preprint 1808. 05760* , 2018.

[MZ15] Shike Mei e Xiaojin Zhu: Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pp. 2871–2877, 2015.

[Tho33] William R Thompson: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[WC18] Yizhen Wang e Kamalika Chaudhuri: Data poisoning attacks against online learning. *arXiv preprint 1808. 08994* , 2018.

[XBBFER15] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert e Fabio Roli: Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pp. 1689–1698, 2015.

6 Appendice

In questa sezione viene descritta l'implementazione MATLAB, poichè, come descritto nei capitoli precedenti, sono tutte molto simili, vengono riportate solo le implementazioni degli attacchi per il

problema *TS* offline e per l'algoritmo online adattativo. Il codice implementato si trova al seguente indirizzo: https://github.com/jacopoRodeschini/TakeHome-AI/tree/master/Code.

Per le altre implementazioni offline, in accordo a quanto specificato ai paragrafi 3.2.1, 3.2.2, 3.2.3, 3.2.4: cambia il vettore dei termini noti b.

```
K = size(succ,2);
T = size(succ,1);
R = size(succ,3);

sigma = opts.sigma; % varianza dell'errore
opt = opts.opt;     % opzioni per programmazione linerare
ski = opts.ski;     % variabile surplus programmazione linerare
delta = opts.delta; % p = 1 - delta

ch_next = NaN(R,1);
cost = NaN(R,1);


for j = 1:R

    % info che servono al manipolatore
    cnt = nansum(ch(:,:,j)); % frequenza delle scelte
    avg = nanmean(succ(:,:,j)); % media stimata
    % inizializzazione problema di ottimizzazione (linear programming)

    H = eye(T);

    % vincoli sul problema di ottimizzazione,
    % poichè ho scelto un sigma che soddisfa la disequazione, posso evitare
    % di inserire il secondo vincolo, poichè risulta che il primo è molto
    % piu stringente
    A = zeros(K-1,T);
    of = 0;
    % attenzione: il branch K è il nostro traget

    % matrice dei coefficienti A
    % attenzione a come sono ordinati i coeff. all'interno della matrice, i
    % risultati non cambiano ma cambia l'interpretazione
    for i = 1:(K-1)
        A(i,of+1:of+cnt(i)) =   ones(1,cnt(i))./cnt(i);
        A(i,T-cnt(K)+1:T) =   -ones(1,cnt(K))./cnt(K);
        of = of + cnt(i);
    end
    % va agg. un vincolo sulla riga K

    % vettore dei termini noti
    b = ones(K-1,1)*avg(K);
    b = b - avg(1:end-1)' + ...
        norminv(delta/(K-1))*sigma^3*sqrt(1./cnt(1:K-1) + 1/cnt(K))';

    % risolutore
    % x: soluzione ottima ovvero overmo tutti i "rumori" dallo step 1 allo
    % step T, ordinati come la matrice A da applicare ai reward ottenuti

    x = quadprog(H,[],A,b,[],[],[],[],[],opt); % SOLUZIONE AMMISSIBILE

    % verifichiamo la soluzione
    % ovvero che al tempo T+1 il braccio scelto sia appunto il braccio K,
    % per far cio ricalcoliamo
```

```
55
56      C = zeros(K,T); % matrice dei coeff.
57      C(1:end-1,1:T-cnt(K)) = A(:,1:T-cnt(K));
58      C(end,T-cnt(K)+1:end) = ones(1,cnt(K))/cnt(K);
59
60
61      % costo tot
62      c = 0;
63      for i = 1:K-1
64          c = c + normcdf(((avg(i) + C(i,:)*x) - (avg(i) + C(K,:)*x))...
65              / (sigma^3 * sqrt(1/cnt(i) + 1/cnt(K))) );
66      end
67      c = c - delta;
68      if c <= 0
69          % calocolo nuove scelta tramite l'update delle medie
70          [m, t] = max(avg + (C*x)');
71          % scelta al passo al T + 1
72          ch_next(j) = t;
73
74          % ratio
75          cost(j) = x' * x;
76      end
77  end
```

Nel primo frame riportato, si implementa l'algoritmo per TS. In accordo con 3.2.4 la matrice A è composta dai termini $1/m_a$ e $-1/m_a^*$ per tutti gli $a \neq a^*$. Il vettore dei termini noti b, implementa la costante: $avg(a^*) - avg(a) + \sigma^3 \sqrt{\frac{1}{m_a} + \frac{1}{m_{a^*}}} \Phi^{-1}\left(\frac{\sigma}{K-1}\right)$. Per calcolare Φ^{-1} si utilizza il comando $norminv(p)$ che calcola l'inverso della distribuzione cumulata normale per la probabilità p. Per risolvere il problema di ottimizzazione quadratica si utilizza il comando $quadprog(H, [], A, b)$ dove A e b, in accordo con 25, implementano il vincolo $Ax \leq b$. Successivamente viene verificato il primo vincolo definito in 51, per cui, se è soddisfatto il vincolo di $c - \delta \leq 0$ viene calcolata la media dei rewar per le diverse scelte con l'aggiunta dell'errore introdotto dall'attaccante. Alla riga 70 si nota come la media dei reward è composta dai due termini: il contributo portato dal reward generato dall'ambiente: $avg(a)$ e il contributo dell'attaccante $\frac{1}{m_a}\epsilon_a I(a_t = a)$. Il decisore quindi, seleziona il massimo della quantità appena calcolata che diventa la scleta all'istante $T + 1$. Infine viene calcolato il costo dell'attacco come norma-l^2.

Viene ora riportata l'implementazione per l'attacco ACE (Attacks Constant Estimation). Una leggera differenza con quella riportata nella repository è che in questo contesto sono state tolte le parti in cui si computa il risultato nella situazioe in cui non ci fosse stato l'attacco, per mostrare con più chiarezza il ruolo dell'attaccante. La strategie *ACE* è generale, ovvero si adatta all'istanza del problema. In particolare, sino alla riga 40 si simula l'agente (in questo caso TS) dalla riga 44 in poi si simula l'attaccante. Per cui per simularlo con altre istanze del problema MAB è sufficiente che siano modificate solo le prime 40 righe.

```
1   clear all
2   rng default
3
4   R = 10;  % round
5   T = 1e4; % timestep
6   K = 5;    % number of branch
7   target = K; % il targhet è sempre l'ultima braccio
8   ind = ones(K,1); % funzione indicatrice
9   ind(target) = 0;
10
11  sigma = 0.1;  % e-greedy
12  eps = 0.1;    % e-greedy tradeoff (static)
13  delta = 0.05; % attack
```

```
14
15  % universal B(n) functions
16  beta = @(n) sqrt(2*sigma^2/n*log(pi^2*n*K^2/(3*delta)));
17
18  mu = [rand(1,K-1) 0]; % random expected values U(0,1);
19
20
21  rewa = NaN(T,K,R);
22  Target = NaN(T,R);
23
24
25  for r = 1:R
26     cnt = zeros(1,K);
27     cntNot = zeros(1,K);
28     avgU = zeros(1,K);
29     for t = 1:T
30          if t<=K
31              dec = t;
32
33          else
34              rd = randn;
35              avg = nanmean(rewa(:,:,r)); % mean branch (bias)
36              avg(isnan(avg)) = 0;
37              [m ,dec] = nanmax(avg/(sigma^2) + rd*sqrt(sigma^2./cnt));
38
39          end
40
41          rd = randn;
42          rw = mu(dec) + rd*sigma;
43
44          avgU(dec) = (avgU(dec) * (cnt(dec)) + rw) / (cnt(dec) + 1);
45
46          cnt(dec) = cnt(dec)+1;
47
48          if t <= K
49              att = 0; % non esguo nessun attocco
50          else
51              att = -ind(dec) *nanmax(0,avgU(dec) - avgU(K) ...
52                        + beta(cnt(dec)) + beta(cnt(K)));
53          end
54          c(t,r) = abs(att); % norma 1
55          rewa(t,dec,r) = rw + att;
56          Target(t,r) = dec;
57
58      end
59  end
```

Come possiamo osservare, nel secondo frame di codice, a differenza di quanto visto per la versione offline, l'attacco viene eseguito ad ogni turno. Alla riga 42 si simula il reward generato dall'ambiente. L'attaccante (riga 44) calcola la media *unbiased* dei reward prima che siano affetti dalla manipolazione. Alla riga 51, in accordo con 70, viene calcolata la stretagia di attacco: $\epsilon_t = -I(a_t \neq a^*)C_{a_t} = -I(a_t \neq a^*)max(0, \tilde{\mu}_a(t) - \tilde{\mu}_{a^*}(t) + \beta(N_a(t)) + \beta(N_{a^*}(t))$ dove $\beta(n)$ è definita in 63. Il termine $-I(a_t \neq a^*)$ è codificato tramite $-ind(dec)$ che è un vettore "indicatrice" del fatto che il braccio sia il braccio target , in tal caso assume il valore 0, oppure uno degli altri possibli $K-1$ bracci e in tal caso assume il valore 1. Poichè siamo nella strategia online, il costo è calcolato come norma-l^1. Infine, alla riga 55, viene calcolato il reward manipolato dall'attaccante che viene "passato" all'agente.