Regret Bound Balancing for Model Selection for Bandit

Lila Mekki, Théo Moret, Augustin Cablant
Ensae Paris
lila.mekki@ensae.fr, theo.moret@ensae.fr, augustin.cablant@ensae.fr
Source code can be found here: Github

Abstract

Bandit algorithms, particularly multi-armed bandits, are designed to balance exploration (gathering information about uncertain options) and exploitation (leveraging the best-known options) in sequential decision-making problems. However, these algorithms typically rely on assumptions that may be challenging to satisfy in practice, increasing the demand for more adaptive methods. In this report, we present the Regret Balancing strategy developed by Ref. [1], which addresses this need for greater flexibility. We highlight multiple use cases of this approach, with a particular emphasis on bandit algorithms and model selection.

1 Introduction

The multi-armed bandit framework is a widely studied approach to sequential decision-making, with application in a large panel of domains such as clinical trials or online advertising. This framework revolves around a sequence of T interaction rounds between a learning agent and an unknown environment. In each round, the agent selects an action from a set of available options, and the environment provides feedback in a form of a reward function, which value depend on the chosen action. The learning agent's objective is to accumulate a total reward over T rounds that is as close as possible to the reward of the best policy. There exists a wide range of bandit algorithms designed to address this problem, but determining which specific algorithm and parameter configuration is best suited to a given problem is often challenging. This raises the critical issue of model selection. In the context of bandit algorithms, model selection refers to the process of identifying the most appropriate algorithmic configuration or model for a problem where the reward structure is unknown or only partially known.

In this report, we address this issue by first providing a mathematical introduction to the multi-armed bandit framework. We then review common bandit algorithms, outlining their underlying assumptions and typical use cases. Building on these foundational algorithms, we introduce the Regret Balancing strategy that builds on these learners. The versatility of this new strategy will be illustrated through numerical experiments conducted in simulated environments. Finally, we dedicate the last section to showcasing the application of this method on real-world data.

2 Bandit Setting

The multi-armed bandit framework models decision-making in uncertain environments, where an agent selects actions (or "arms") over several rounds to maximize cumulative rewards. At each round t, the agent selects an arm, observes the associated reward r_t and updates its strategy. We consider stochastic rewards (each arm's reward is sampled from a fixed but unknown probability distribution). A fundamental challenge in this setting is balancing *exploration* (gathering information about uncertain arms) and *exploitation* (selecting the arm with the highest expected reward based on current knowledge).

Formally, the multi-armed bandit problem, can be defined as follows:

- **Arms:** A set of $\chi_t \subseteq \chi$ arms (that can be different at each round). In each round, the agent selects an arm $x_t \in \chi_t$.

- **Rounds:** The interaction occurs over T discrete rounds, indexed by $t = 1, 2, \ldots, T$ with T being the horizon.
- **Rewards:** At each round t, the agent selects an arm x_t and observes a reward r_t according to the distribution specified by the environment. The reward r_t is sampled from an unknown distribution associated with the selected arm x_t. Denote the mean reward of arm x as $\mu_x = \mathbb{E}[r_t | x_t = x]$.
- **Objective:** The goal is to maximize the cumulative reward over T rounds: $R_T = \sum_{t=1}^{T} r_t$. Alternatively, the performance can be measured by the *regret*, which quantifies the loss due to not always playing the optimal arm:

$$\text{Regret}_T = T\mu^* - \sum_{t=1}^{T} \mu_{x_t},$$

where $\mu^* = \max_{x \in \chi} \mu_x$ is the mean reward of the best arm. This quantity is useful in theoretical analysis of algorithms, especially when deriving rate of convergence.

2.1 Environments

The environment specifies the reward structure and the rules of interaction. We limit ourselves to stochastic rewards in the following. There exists two types of environments: non-contextual and contextual.

A non-contextual environment consists in a reward policy where the reward of each arm is determined by a fixed distribution, independent of any external context. Each arm $x \in \chi$ has an associated mean reward $\mu_x = \mathbb{E}[r_t | x_t = x]$, which the agent does not know. The agent's goal is thus to interact with the environment by selecting arms to minimize the regret over time. Since there is no additional context, the agent decides based only on past rewards from the arms it has pulled. This is for instance the case in a slot machine where each arm has a fixed probability of payout, and the agent tries to identify and exploit the best lever through trial and error. We implemented a non-contextual environment named `BernoulliBanditEnv`. This a simple bandit environment where each arm gives rewards according to a corresponding Bernoulli distribution.

Contrary to the previously described environment, the reward of each arm depends here both on the arm chosen, and an external context vector, which may vary across rounds. At each round t, the agent selects an arm x_t and observes a reward $r_t = f_*(x_t) + \epsilon_t$ where $f_* : \chi \to \mathbb{R}$ is an unknown regression function and ϵ_t is a centered noise, independent from previous data. Hence, the expected reward is a function of the context, e.g., $f_*(x_t) = \theta_*^\top x_t$ for a linear model. Then, θ_* is the unknown regression vector that represents the truth that optimizes the reward. Context enables the agent to make more informed decisions compared to the non-contextual case. The agent's goal is to infer θ_* and select actions to maximize rewards based on their linear relationship with θ_*. This is for example the case of online advertising. At each step, the system observes user features (context, such as the browsing history), and chooses an ad (arm) to display. The reward could be the user clicks on the ad in this example. To this end, we implemented a linear bandit environment : `LinearBanditEnv`. This is a contextual bandit framework in which the reward function is $f_*(x_t) = \theta_*^\top x_t$, with $\theta_* \in \mathbb{R}^d$ a regression vector unknown to the agent. The expected reward is then linear: $\mathbb{E}[r_t | x_t] = \theta_*^\top x_t$ and is observed with an additional Gaussian noise by the agent.

2.2 Agents

The agent interacts with the environment by selecting arms and updating its strategy based on observed rewards. Hence, different agents are designed for the two previously described environments. The non-contextual agent is well suited for non-contextual environments. The agent maintains estimates of each arm's mean reward (μ_x). It uses algorithms such as ϵ-greedy, Upper Confidence Bound (UCB), or Thompson Sampling to balance exploration and exploitation. The limitation of this kind of agent is that it is unable to incorporate additional information (context), potentially leading to suboptimal decisions. On the other hand, contextual agents can be developed, but they require to act on a contextual environment to ensure optimality. The agent learns a mapping between context and rewards, often assuming a parametric form for the reward function (e.g., linear or non-linear regression). It uses algorithms like LinUCB to predict expected rewards for each arm based on the observed context. By leveraging context, the agent can generalize knowledge about rewards across similar contexts, enabling better decision-making in dynamic or complex settings. We explain in greater details these algorithms in the next section.

3 Classical Bandit Algorithms

A fundamental challenge in the bandit framework is balancing *exploration* (sampling less-known arms to gather information about their rewards) and *exploitation* (selecting the arm believed to yield the highest reward based on current knowledge). We will see different strategies that address this trade off to minimize regret over time.

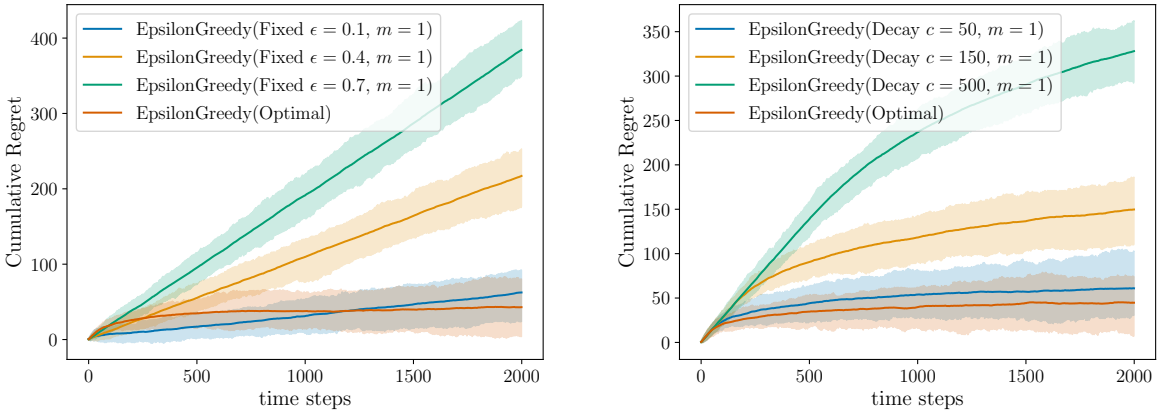
3.1 ε-greedy algorithms

The ε-greedy algorithm is a simple and widely used method in bandit problems. At each round t, the algorithm selects an action uniformly at random with probability ε_t and chooses the empirically best (greedy) action otherwise ($x_t = \text{argmax}_x \hat{\mu}_x$ with probability $1 - \epsilon$). The choice of ε_t is critical. If ε_t is too large, the algorithm over explores, leading to suboptimal performance. If ε_t is too small, the algorithm under explores, risking insufficient information about the reward distribution. The optimal exploration rate ε_t depends on the unknown reward distribution of each arm. It is known that the optimal value of ε_t is given by:

$$\varepsilon_t = \min\left\{1, \frac{5K}{t\Delta^2}\right\}, \tag{1}$$

where Δ is the smallest gap between the optimal reward and the suboptimal rewards, and K is the number of actions. For this optimal choice of ε_t, the regret scales as $\mathcal{O}(\sqrt{T})$ for $K = 2$, and $\mathcal{O}(T^{2/3})$ for $K > 2$. However, this requires knowledge of Δ which is not the case in practical applications.

At the beginning (initial phase), each arm is explored a minimal number of times, without accounting for past rewards. Then, ε_t is adjusted for a control of exploration probability compared to exploitation. For choosing ε_t, we display three cases. If Δ is known, we set $\varepsilon_t = \min\left\{1, \frac{5K}{t\Delta^2}\right\}$ and ε_t decreases along the iterations, ensuring that exploration diminishes. If Δ is unknown, one can set $\varepsilon_t = c/t$ for a given c. If none of those parameters are defined, ε_t is adjusted with the empirical values of the arms to mimic the optimal rate. The trade off betwen exploration and exploitation is assured by decreasing the value of ε_t through iterations, accounting less and less on exploration of new arms, and increasingly on the information acquired on the best arm.



(a) Evolution of cumulative regret over time for different ε values

(b) Cumulative regret over time for different decay values

Figure 1: Comparison of cumulative regrets through iterations based for fixed ε and for decreasing exploration rate $\varepsilon_t = c/t$

Fig. 1 displays result for a Bernoulli environment with $K = 2$ arms and means $\mu_1 = 0.2$ and $\mu_2 = 0.5$. On figure 1a, cumulative regret increases linearly over time due to the constant exploring rate. Note that for higher ε (closer to 1), the algorithm explores more frequently, resulting in more uniform sampling across arms which is useful at the beginning but less over time, as information on the arms has been gathered. Hence, an increases of the regret through iterations. When ε is closer to 0, the algorithm exploits more. On

figure 1b, the decay reduces the exploration rate over time. Initially, it is high to encourage exploration, but decreases as the algorithm gathers data. Note that the used Bernoulli environment is very simple, two arms which are very distinct in distributions. This favors the small values of c in $\varepsilon_t = c/t$ since only low exploration is needed to distinguish between the two given arms.

3.2 UCB algorithm

The UCB (Upper Confidence Bound) algorithm addresses the exploration-exploitation trade-off in the multi-armed bandit problem by constructing a confidence region for the expected reward of each arm. The confidence region is refined as iterations progress and the algorithm exploits in the refined region. This relies on Hoeffding's concentration inequality, hence assuming that each arm is independent (excluding by definition the case of contextual environments).

At each time step t, the algorithm maintains an estimate of the expected reward for each arm i. To account for uncertainty due to limited observations, it constructs a confidence region around the estimated mean reward. Assuming that rewards are $\sigma-$subgaussians, we have

$$P\left(\mu_x \leq \hat{\mu}_x(t) + \sqrt{\frac{6\sigma^2 \log(t)}{N_{t-1}^x}}\right) \geq 1 - \frac{1}{t^2}\,,$$

so that we define the Upper Confidence Bound for UCB(α) by

$$UCB_x^\alpha = \hat{\mu}_{t-1}^x + \sqrt{\frac{\alpha \log t}{N_{t-1}^x}}$$

where N_{t-1}^x gives the number of times each arm has been selected, $\sqrt{\frac{\alpha \log t}{N_{t-1}^x}}$ is the width of the confidence interval, which decreases as time goes on. The α parameter adjusts the level of confidence reached. A higher α leads to more exploration, as the algorithm is more cautious about the reward estimates. A lower α leads to more exploitation, as the algorithm trusts its reward estimates sooner. Hence, at each round, UCB_x^α is calculated and we act greedily with respect to it, *ie.* the arm x which provides the highest UCB is selected. This ensures a good Exploration-Exploitation trade off. Indeed, arms with fewer observations (N_{t-1}^x small) will have wider confidence intervals, since uncertainty is larger, encouraging exploration. But over time, as N_{t-1}^x increases, the uncertainty shrinks (the confidence interval is tight), leading to exploitation of the arm with the highest estimated reward $\hat{\mu}_{t-1}^x$. The regret bound scales as $\mathcal{O}(\sqrt{KT \log T})$ with K the number of actions and T the number of iterations.

3.3 LinUCB Algorithm

The LinUCB algorithm extends UCB to the contextual bandit setting, where rewards depend on context vectors. The expected reward of arm i at time t is supposed linear *ie.* $\mathbb{E}[r_{i,t}] = \theta_*^\top x_{i,t}$ with x_t the context vector of arm i at time t and θ_* the unknown parameter vector that determines the expected reward.

LinUCB constructs an estimate of θ_* by regularized least-square

$$\hat{\theta}_t^\lambda = (B_t^\lambda)^{-1} \sum_{s=1}^{t} r_s x_s\,, \tag{2}$$

with $B_t^\lambda = \lambda I + \sum_{s=1}^t x_s^\top x_s$ and $\lambda > 0$ the regularization parameter. An exploration bonus (β term) is added to this estimation:

$$\beta(t,\delta) = \sigma\sqrt{2\log\frac{1}{\delta} + d\log\left(1 + \frac{tL}{d\lambda}\right)} + \sqrt{\lambda}\,,$$

where d is the dimension of the feature vector ie arm x_t and L the upper bound on the norm of the feature vector. This choixe of β ensures that at some point, the estimate $\hat{\theta}$ will be β-close to θ_* with probability at least $1-\delta$. The upper confidence bound for arm i is then :

$$\text{UCB}_i(t) = \hat{\theta}(t)^\top x_{i,t} + \sqrt{x_{i,t}^\top (B_t^\lambda)^{-1} x_{i,t}}\ \ \beta(t,\delta),$$

where $x_{i,t}^\top \hat{\theta}(t)$ is the estimated reward for arm i and $\sqrt{x_{i,t}^\top (B_t^\lambda)^{-1} x_{i,t}} \;\; \beta(t,\delta)$ is the uncertainty term. At each round, we chose the arm with the highest confidence bound. This method still ensures the Exploration-Exploitation trade off. In the exploration phase, the algorithm selects the arms that provide the most information about the unknown parameter θ_* to improve future decisions. The exploration term $\sqrt{x_{i,t}^\top (B_t^\lambda)^{-1} x_{i,t}} \;\; \beta(t,\delta)$ is high when the context vector $x_{i,t}$ points in a direction of the feature space that has been poorly explored (i.e., when $(B_t^\lambda)^{-1}$ assigns high variance to $x_{i,t}$). Exploration favors arms with high uncertainty, ensuring that poorly explored arms or dimensions of the context space are sampled. The regret scales as $\mathcal{O}(\sqrt{KT \log T})$, with K being the number of arms. This sub-linearity in T implies that the average regret per time step decreases as T increases.

In this section, we have presented the usual bandit algorithms, but as seen, they rely on strong hypothesis on the underlying reward environment. Most of the time, such information is not provided to the experimentalist, hence improving the need of more flexible method. In the following section, we present one of them, called the Regret Balancing Method.

4 Regret Balancing Method

As presented in the previous section, all usual bandit algorithms often rely on strong hypothesis, making their use is practice somewhat limited in many cases. For instance, we may not know how the environment interacts, and more specifically if the underlying reward scheme is linear or not. Hence, how to choose between an UCB or LinUCB algorithm ? Similarly, when using ε-greedy algorithm, one needs to specify the decay rate of ε_t. It is well-known that the optimal value of this parameter can be fixed knowing Δ, the smallest gap between the optimal reward and the suboptimal rewards. Once again, this suppose a complete knowledge of the reward distribution of each arm, which is usually not available. All these different assumptions make achieving an optimal reward in practice very hard. However, we almost always know what kind of algorithms is better tuned for our problem : UCB/LinUCB, ε-greedy,... Hence, Ref. [1] proposes a method to achieve suboptimal regret relying on the set of base algorithm (eg. multiple ε-greedy algorithms with different decay rate). They present an effective model selection strategy that adapts to the best learning algorithm in an online fashion, among all the base learner. To do so, they make sure that estimate the regret of each algorithm at each round and play the algorithm such that all empirical regrets of each learner are ensured to be of the same order, hence introducing the Regret Balancing strategy. By knowing an upper bound on the optimal base learner, this method can be tuned to achieve a near-optimal regret *ie.* a regret that is optimal up to a multiplicative constant being the number of base learner used. Note that this balancing strategy encapsulates the trade-off between exploration and exploitation: if a base algorithm is played only for a small number of rounds, or if it plays good actions, then its empirical regret will be small and will be chosen by the model selection procedure. We present in the following the mathematical formulation of the decision process at each round. First, suppose a set of M learners *ie.* at each round t, the regret balancing strategy will choose a base algorithm $i_t \in [M]$. Let's introduce some notations:

- $N_{i,t}$:= number of rounds that base i is played up to but not including round t, and as a abuse of notation also the set of round that base i has been played;
- $R_{i,t}$:= the total reward of the base i during these $N_{i,t}$ rounds;
- $S_{i,t} := \{(s_t, a_t, r_t) : t \in N_{i,t}\}$;
- $G_{i,t} := \sum_{\tau \in N_{i,t}} \mu_{*,\tau} - R_{i,t}$ is the regret of base i during the $N_{i,t}$ rounds.

We assume that a high probability upper bound on the regret of the optimal base algorithm is known: a function $U : \mathbb{R} \times \mathcal{H} \to \mathbb{R}$ is given so that for any $\delta \in (0,1)$, with probability at least $1-\delta$,

$$G_{i_*,t} \leq U(\delta, S_{*,t}) \;, \quad \forall t. \tag{3}$$

The $\mathcal{H}$ space is the set of possible histories *ie.* $\{S_{i,t} : \forall i \in [M], \forall t \in [T]\}$, allowing the given bound to be data-dependent. The model decision strategy is then: at round t, let j_t be the optimistic base and b_t be the optimistic value:

$$j_t = \arg\max_{i \in [M]} \frac{R_{i,t} + U(\delta, S_{i,t})}{N_{i,t}}, \quad b_t = \frac{R_{j_t,t} + U(\delta, S_{j_t,t})}{N_{j_t,t}}. \tag{4}$$

The variable b_t estimates the value of the best action. Define the empirical regret of base i by

$$\widehat{G}_{i,t} = N_{i,t} b_t - R_{i,t} \ , \tag{5}$$

And at time t, we choose the base with the smallest empirical regret:

$$i_t = \arg\min_{i \in [M]} \widehat{G}_{i,t} \ . \tag{6}$$

Hence, base i will see its regret increased, ensuring the regret balancing strategy. The next theorem proved by Ref. [1] ensures the near-optimal regret of this strategy.

Theorem 4.1. *If $\mu_{*,t} = \mu_*$ for a constant μ_* regardless of time t, and if with probability at least $1 - \delta$, $G_{i_*,t} \leq U(\delta, S_{i_*,t})$ for any t, then:*

$$\textit{Regret}_T \leq M \max_{i \in [M]} U(\delta, S_{i,T}) \ ,$$

with probability at least $1 - \delta$.

The first condition can be interpreted as having an underlying optimal learner of the problem. The second one is the requirement of the upper bound on the regret. The obtained equation shows that a near-optimal regret is achieved with this strategy, and that this regret is controlled by the tightness of the bound $U(\delta, S_{i,T})$. Note that the condition of having a (true) bound on base learners can be relaxed, as shown in Ref. [2]. In this paper, only candidate bounds are needed, and base learners experience an elimination process at each round based on their performance, to detect if the provided bound is violated or not.

In the following section, we present numerical experiences on simulated environment to illustrate the regret balancing method and its possible application cases, mainly using it at a bandit algorithm itself and as a representation learning strategy.

5 Numerical Experiences

In this section, we first provide an overview of the source code, more specifically how to use it to run bandit experiments. We then run two different numerical experiences on simulated environments to illustrate the Regret Balancing method. Note that the implementation of the linear environment and the `experiment` method as been taken from Claire Vernade's website[1]. We take inspiration of the structure to make our own environments and agents.

5.1 Code Structure

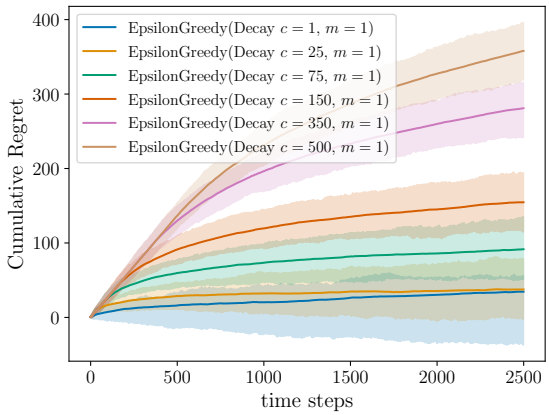
The source code contains the implementation of mathematical concepts presented in Sec.2 and Sec.3. First, two bandit environments are implemented. The first one is `BernoulliBanditEnv` which provides an environment with K arms, each following a Bernoulli distribution with mean μ_i for $i \in [K]$. You can act on this environment by choosing an action $i \in [K]$, and receive a reward using the `get_reward` method, which pulls the arm i *ie.* returns a number drawn from a Bernoulli $\mathcal{B}(\mu_i)$. The second environment is `LinearBandit` which implements a contextual bandit environment. The reward model is supposed linear, described by an underlying $\theta^* \in \mathbb{R}^d$ parameter, hidden to the player. At round t, after having chosen the action $x_t \in \mathbb{R}^d$ among K possible ones, you can receive a reward using the `get_reward` method to obtain a reward drawn from a normal distribution $\mathcal{N}(x_t^\top \theta^*, \sigma^2)$. Multiple agents have been implemented to act on these environments, namely `EpsilonGreedy`, `UCB`, `LinUCB` and `RegretBalancingAgent`. All needed parameters can be adjusted (see source code for more details). After having defined an environment and a list of agents that you wish to test, you can run an experiment by calling the `experiment` method. This will basically run each chosen agent on the given environment using Monte-Carlo approximation : `Nmc` trajectories will be simulated up to horizon `T` each, then regrets will be tracked and returned along with the agent name in an array. Results can be visualized using `plot_regret` method, that also displayed confidence intervals.

5.2 Optimizing the Exploration Rate of ε_t-greedy algorithms

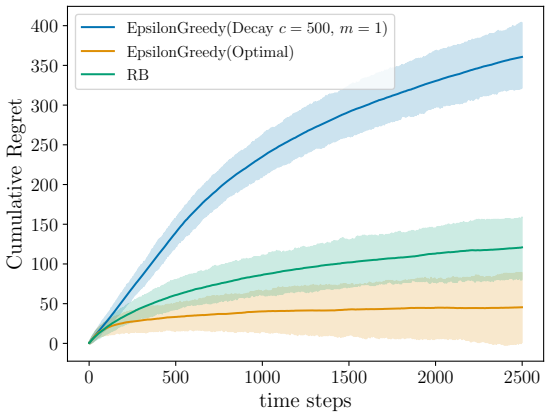
In this section, we run an experiment to show that the Regret Balancing method achieved a near-optimal regret in an Bernoulli environment when the sub-optimality gap Δ, hence the optimal exploration rate

[1] https://www.cvernade.com/teaching

is unknown. As explained in Ref. [1], the Regret Balancing method as to be applied with regret bound $U(t) = \sqrt{t}$. We initiate a Bernoulli environment with $K = 2$ arms and means $\mu_1 = 0.2$ and $\mu_2 = 0.5$. Then, the optimal ε_t-greedy algorithm will be run with an exploration rate of $\varepsilon_t = \min\left(1, \frac{5K}{t\Delta^2}\right)$, where $\Delta = \mu_2 - \mu_1 = 0.55$. However, since this information is hidden to a player, we choose to define 6 base learners with $\varepsilon_t = c/t$ for $c \in \{1, 25, 75, 150, 350, 500\}$. The performance of these learners are displayed in Fig. 2a. The Regret Balancing algorithm take these six algorithms as base learners. The cumulative regret of this agent and the optimal one are displayed in Fig. 2b



(a) Cumulative regret of the six base learners. The parameter $m = 1$ ensures that each arm is pulled at least once.



(b) Cumulative regret of the optimal algorithm, the Regret Balancing one and the worst base learner.

Figure 2: The experiment has been run for 250 Monte-Carlo trajectories over horizon $T = 2500$. Confidence intervals at 90% are displayed.

As Fig. 2 shows, the Regret Balancing method achieves performance close to the optimal algorithm, given the six described base learner. More accurate performance could be achieved by increasing the number of base learner, but we did not do it for clarity. Also note that the used Bernoulli environment is very simple, two arms which are very distinct in distributions. This favors the small values of c in $\varepsilon_t = c/t$ since only low exploration is needed.

5.3 Is context needed ?

As presented in Sec.2 , UCB considers each arm as independent and is a context-free bandit algorithm. In contrast, LinUCB is a contextual bandit algorithm which assumes linear reward model with context. But in some settings such as personalization problems, the context might not be predictive of the user behavior, and even if context is available, using a non-contextual bandit as UCB might lead to better performance. Without knowing in advance which algorithms is better suited for the specific problem using the Regret Balancing method with base learners being UCB and LinUCB will avoid committing to a non-optimal algorithm and near-optimal rate will still be achieved. To illustrate this, we initiate a linear bandit environment, with a linear reward model and informative context represented by the unknown vector $\theta^* \in \mathbb{R}^{10}$. The action space is described by $K = 5$ arms, that are vectors in $\mathbb{R}^{10}$. On this environment, the LinUCB algorithm is perfectly suited, while the UCB will have poor performance. We initialize these two agents : UCB with $\alpha = 2$ and LinUCB with regularization parameter $\lambda = 1$. The Regret Balancing agent takes these two algorithms as base learners. Results are displayed in Fig. 3.

As expected, the LinUCB algorithm is better suited for this environment, but the Regret Balancing agent achieves good performance without assuming anything on the structure of the reward model and the context. This provides a flexible way to make bandit algorithms in uncertain conditions.

After having developed code to test the different algorithms on multiple environments, we now present in the next section an application of the Regret Balancing method on a real data set, namely the Movielens 25m dataset.

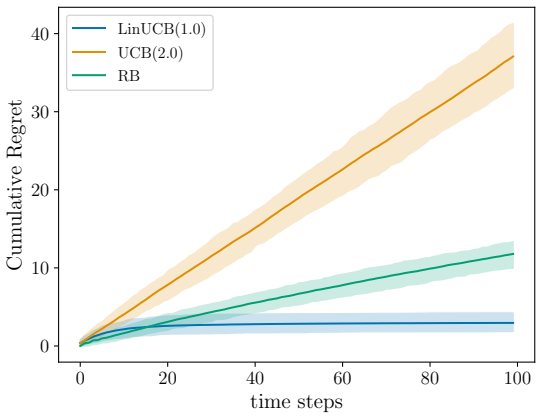


Figure 3: The experiment has been run for 250 Monte-Carlo trajectories over horizon $T = 100$. Confidence intervals at 90% are displayed.

6 Test on Real Data

Multi-armed bandit algorithms are gaining renewed interest due to their ability to address major challenges faced by online companies, such as personalization tasks. Indeed, they need to explore a constantly evolving landscape while avoiding showing low-quality content to users. Additionally, advancements in contextual bandits, combined with reinforcement learning, have further increased their relevance. However, these algorithms are difficult to work with on real-world datasets, as evaluating them online can be risky. It is crucial to assess them offline for two reasons: limited access to large-scale production environments and the need for caution before deploying untested algorithms. In this section, we deployed the Regret Balancing on a real data set, and assessed its performance in a coherent offline fashion.

6.1 Data

We use the Movielens 25m dataset, which contains ratings for movies from users. We first need to preprocess this dataset to make it usable for bandit algorithms. To do so, we used the preprocessing steps given by James LeDoux[2]. The dataset is transformed into a binary problem, where a movie is considered "liked" if it receives a rating of 4.5 stars or higher. To simulate a bandit environment, the data is shuffled and given pseudo-timestamps, simplifying the problem by ignoring non-rating biases. Additionally, movies with fewer than 1,500 ratings are excluded.

6.2 Recommendations and Target Metrics

To train the bandit, the data is treated as a time-evolving process, where each bandit decision involves using past data to update the model, make predictions, and evaluate the rewards. This process is computationally intensive, so instead of updating after every single event, updates are batched every n events. The problem is expanded to a slate recommendation task. Indeed, in real data settings, it is not provided that we can access the feedback that the user would have given to each recommendation the bandit may make. Hence, by recommending bandit's top movies to a user, increasing the likelihood of receiving feedback that we can use to improve the algorithm. The number of recommended movies is set to 5 in our experiment. To train the model offline, a "history" dataset is created that accumulates data from previous events, ensuring the bandit only sees past data during training. This dataset is updated at each time step during training to simulate real-world scenarios where the bandit model learns over time. Then, to assess the performance in an online fashion, the replay evaluation method is used [3]. Replay evaluation involves examining your historical event stream and the algorithm's recommendations at each time step. It filters out all instances except those where the model's recommendation matches the one the user actually encountered in the historical dataset. This approach closely mimics real-world learning scenarios, where the bandit refines its policy based on feedback from its own recommendations rather than relying on a

[2]Source code can be found here : https://github.com/jldbc/bandits/tree/master

fixed, pre-determined policy. The replay method also helps eliminate bias present in the historical dataset, providing a more accurate simulation of how the bandit would perform in a live production environment. Finally, as opposed to theoretical settings where the regret is used as a metric, in real data settings, we don't know the ground truth of the environment. We then rely on the cumulative reward obtained. The cumulative reward can be calculated without requiring knowledge of the counterfactual outcomes

6.3 Results

Along with the preprocessing steps, James LeDoux provides some algorithms that use this dataset. We take those as inspirations to make our own ϵ_t-greedy algorithms and associated Regret Balancing agent. Due to the time needed to run the algorithms, we decided to run 4 different ϵ_t-greedy algorithms, with $\epsilon_t = c/t$ for $c \in \{10, 10^3, 10^5, 10^8\}$. These algorithms are the one used as base learners for the Regret Balancing method. As before, we used $U(t) = \log(Kt)$ as the target bound, with K the number of unique movies that can be recommended to users. Results are displayed in Fig. 4.

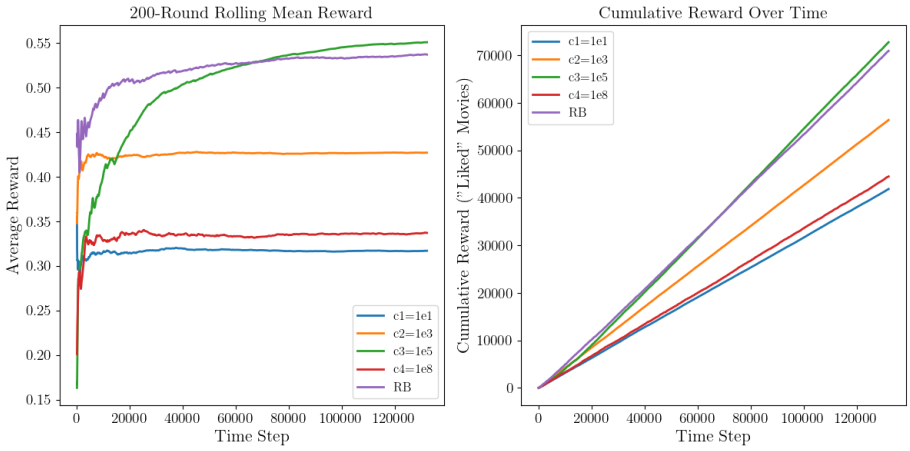


Figure 4: Results for each base learners and the Regret Balancing method. Average reward is computed aswell as the cumulative reward obtained.

As we see from the Average Reward, three of the four ϵ_t-greedy algorithms recommend movies that are not liked most of the time, while the one with $c = 10^5$ is the only one crossing the 50% threshold. Even in this harsh conditions, the Regret Balancing method manages to cross this threshold too, and to obtain a cumulative reward that is closed to the best learners among base ones. Note that this has only been run for 4 base learners, and that the performance could be improved by considering more base algorithms. However, this method already beats more complex approaches on this dataset, such as EXP3 that didn't cross the 50% threshold is the same conditions (see James LeDoux GitHub for details). Hence, the Regret Balancing method is useful in real data setting by not needing to over commit on miss-tuned hyper-parameters (see bad performances of some base learners). This may also be a good approach to deploy more confidently the algorithm in an online fashion.

7 Conclusion

In this project, we developed the foundational theory of multi-armed bandits, starting with an introduction to standard environments and their associated algorithms. We then implemented these environments and algorithms to illustrate their properties through numerical simulations. Highlighting the significant assumptions each algorithm requires to achieve optimal regret bounds, we motivated the need for a more flexible approach and introduced the Regret Balancing method. To validate this method, we developed source code for numerical testing, demonstrating its ability to achieve near-optimal regret while relying on fewer assumptions. Additionally, we tested the method on real-world data, yielding conclusive results. An extension of this work involves implementing an elimination scheme, where underperforming learners can be removed from the pool, thereby improving the efficiency of the Regret Balancing method, as detailed in Ref. [2].

References

[1] Yasin Abbasi-Yadkori, Aldo Pacchiano, and My Phan. Regret balancing for bandit and rl model selection. *arXiv preprint arXiv:2006.09479*, 2020. The exact preprint URL or journal details need verification if available.

[2] Aldo Pacchiano, Christoph Dann, Claudio Gentile, and Peter Bartlett. Regret bound balancing and elimination for model selection in bandits and rl. *arXiv preprint arXiv:2006.05491*, 2020.

[3] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM'11, page 297–306. ACM, February 2011.