NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

Master Thesis

Mathematical Models and Algorithms for Contextual Multi-armed Bandit Problems

---

Author:
Dimitrios ZACHARIS

Supervisor:
Prof. Apostolos BURNETAS

A thesis submitted in partial fulfillment of the requirements for the degree of
M.Sc in Statistics and Operational research
in the

Faculty of Science,
Department of Mathematics

Athens, September 2023

Η παρούσα Διπλωματική Εργασία εκπονήθηκε στα πλαίσια των σπουδών για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης «Στατιστική και Επιχειρησιακή Έρευνα»

που απονέμει το

Τμήμα Μαθηματικών

του

Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την…….. από Εξεταστική Επιτροπή

Αποτελούμενη από τους:

Ονοματεπώνυμο	Βαθμίδα	Υπογραφή
1. Απόστολος Μπουρνέτας(Επιβλέπων Καθηγητής)	Καθηγητής	…….....
2. Παναγιώτη Μερτικόπουλο	Καθηγητής	…….....
3. Αντώνιο Οικονόμου	Καθηγητής	……..……

Acknowledgements

First of all, I would like to thank my thesis supervisor, Professor Apostolos Burnetas, for his guidance through each stage of the process, for all his help and advice, and for inspiring my interest in the field of artificial neural networks and optimization.

In addition, I would like to express my deepest appreciation to my examination committee, Professor Antonis Economou and Professor Panagiotis Mertikopoulos, for their insightful comments and suggestions.

Finally, I would like to express my gratitude to my family and friends for all their encouragement and support during my studies.

Contents

1 Introduction

In this thesis, we will consider about mathematical models and algorithms for multi-armed bandit problems and especially for the contextual bandits. Contextual bandits belong to the field of reinforcement learning and in such a problem, the algorithm has to make decisions about the choice of actions based on contexts, which include information about the current state of the environment and possibly previous information collected. The goal of the algorithm is to learn a policy that, over time, will select actions with the greatest potential payoff. To achieve this, the learner uses different exploration-exploitation strategies that balance the trade-off between exploring new arms to collect new information and exploiting what is already known.

The structure of the thesis is as follows:

In chapter 2, we make an introduction to bandit problems by giving basic definitions (learner, environment, regret, etc), some explanatory notes and examples in order to make more concrete these ideas. We also mention some key applications of these problems, such as A/B Testing, advertisement placement, network routing, etc.

The next chapter refers to stochastic bandits, indicating their structure and a couple of algorithms, like ETC, UCB, Epsilon-greedy. Furthermore, there are theorems, which give important results about the bounds that the regret satisfies.

The chapter 4 refers to adversarial bandits, another main category of bandit problems. We present theorems, algorithms (Exp3), notes and examples, as previously. In addition, we record the similarities and differences of stochastic and adversarial environments.

In the next chapter, we analyze the contextual bandits, which are our main topic of the thesis. Firstly, we divide these problems into adversarial and stochastic and for each category we mention the basic algorithms and their results, Exp4 and LinUCB respectively. We give more attention to the LinUCB algorithm, which is actually the UCB algorithm applied to linear bandits.

In chapter 6, we present the Thompson Sampling algorithm. We give an example, in which we apply this algorithm to Bernoulli bandits with two arms, as Thompson initially did. Then,

we restrict the algorithm to contextual bandits and for the different cases we modify the pseudo-code and extract the corresponding results.

In the last chapter, we simulate in R-code an example of implementing, not only the Thompson Sampling algorithm for a Bernoulli bandit problem with two arms, but also the LinUCB algorithm, with the same data. The codes are shown in appendix.

2 Introduction to Bandit Problems

In 1933, William R. Thompson introduced a concept known as bandit problems through an article published in Biometrika. Thompson's primary focus was on medical trials and the ethical concerns associated with conducting trials without adapting the treatment allocations in response to the drug's varying effectiveness. The term “bandit problems” emerged later in the 1950s when Frederick Mosteller and Robert Bush conducted experiments on mice and humans to explore animal learning. In these experiments, mice were faced with the task of choosing between left and right directions in a T-shaped maze, uncertain of which end would lead to food (figure 1). To replicate this learning scenario in humans, a machine called a “two-armed bandit” was created. This machine allowed humans to choose between pulling the left or right arm, with each arm yielding a random payout, the distribution of which remained unknown to the player (figure 2). The name “two-armed bandit” was inspired by the term “one-armed bandit”, an old-fashioned reference to a lever-operated slot machine, with “bandit” alluding to the notion of money being taken away.

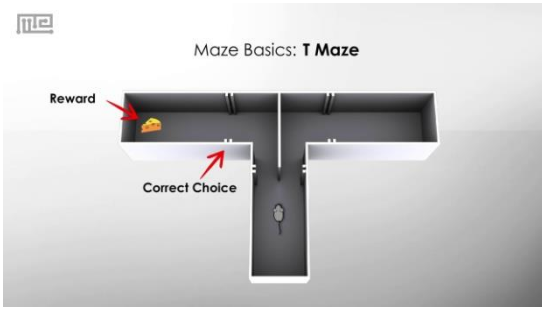

Figure 1: Mouse learning a T-maze

Figure 2: Two-armed bandit

There are many reasons to care about bandit problems. Decision-making with uncertainty is a challenge we all face, and bandits provide a simple model of this dilemma. Bandit

problems also have practical applications. Major tech companies use bandit algorithms for configuring web interfaces, where applications include news recommendation, dynamic pricing and ad placement. A bandit algorithm plays a role in Monte Carlo Tree Search, an algorithm made famous by the recent success of AlphaGo.

2.1 The Language of Bandits

A bandit problem is a sequential game between a **learner** and an **environment**. The game is played over n rounds, where n is a positive natural number called the **horizon**. In each round $t \in \{1, \dots, n\}$, the learner first chooses an action A_t from a given set $\mathcal{A}$, and the environment then reveals a reward $X_t \in \mathbb{R}$. Actions are often also called "arms". We talk about k**-armed bandits** when the number of actions is k, and about **multi-armed bandits** when the number of arms is at least two. There are also **one-armed bandits**, which are really two-armed bandits where the pay-off of one of the arms is a known fixed deterministic number.

Every action A_t should only depend on the **history** $\mathcal{H}_{t-1} = (A_1, X_1, \dots, A_{t-1}, X_{t-1})$**.** In the context of interactions between a learner and an environment, a **policy** can be understood as a function that maps histories to actions: "The learner utilizes this policy to make decisions on how to interact with the environment. On the other hand, the environment can be seen as a function that maps history sequences, which culminate in specific actions, to corresponding rewards". The most common objective of the learner is to choose actions that lead to the largest possible cumulative reward over all n rounds, which is equal to $\sum_{t=1}^{n} X_t$.

The challenge in bandit problems is that the environment is unknown to the learner, who only knows that the true environment lies in some set $\mathcal{E}$ called the **environment class**.

Definition 1: The regret of the learner relative to a policy π is the difference between the total expected reward using policy π for n rounds and the total expected reward collected under complete information over n rounds. The regret relative to a set of policies Π is the maximum regret relative to any policy $\pi \in \Pi$ in the set. The set Π is often called the **competitor class**.

Note: We usually measure the regret relative to a set of policies Π that is large enough to include the optimal policy for all environments in $\mathcal{E}$. In this case, the regret measures the loss suffered by the learner relative to the optimal policy.

For a fixed policy and competitor class, the regret depends on the environment. The environments where the regret is large are those where the learner is behaving worse. The ideal case is that the regret should be small for all environments (A large environment class corresponds to less knowledge by the learner). The **worst-case regret** is the maximum regret over all possible environments.

Example:

Suppose the action set is $\mathcal{A} = \{1, \ldots, k\}$. An environment is called a **stochastic Bernoulli bandit** if the reward $X_t \in \{0,1\}$ is binary valued and there exists a vector $\mu \in [0,1]^{\mathrm{k}}$ such that the probability that $X_t = 1$ given the learner chose action $A_t = \alpha$ is μ_α. The class of stochastic Bernoulli bandits is the set of all such bandits, which are characterized by their mean vectors. The optimal policy under complete information is to play the fixed action $a^* = argmax_{a \in \mathcal{A}}\, \mu_\alpha$. For this problem the natural competitor class is the set of k constant polices $\Pi = \{\pi_1, \ldots, \pi_\kappa\}$ where π_i chooses action i in every round. The regret over n rounds becomes:

$$R_n = n \max_{a \in \mathcal{A}} \mu_\alpha - \mathbb{E}\left[\sum_{t=1}^{n} X_t\right]$$

Note: 1) One straightforward problem scenario is that of **stochastic stationary bandits**. In this particular setting, the environment is constrained to produce rewards for each action based on a distribution unique to that action. Importantly, these reward distributions are independent of prior action choices and rewards. The environment class described in the previous example adheres to these conditions, but there are other possibilities as well. For example, instead of a Bernoulli distribution, the rewards could follow a Gaussian distribution. While this difference may seem large, it does not fundamentally alter the nature of the problem. A more drastic change is to assume the action set $\mathcal{A}$ is a subset of $\mathbb{R}^d$ and that the mean reward for choosing some action $\alpha \in \mathcal{A}$ follows a linear model, $X_t = < a, \theta > + \eta_t$ for $\theta \in \mathbb{R}^d$ and η_t a standard Gaussian (zero mean, unit variance). The unknown quantity in this case is θ and the environment class corresponding its possible values ($\mathcal{E} = \mathbb{R}^d$).

2) Another idea is to drop all assumptions on how the rewards are generated, except that

they are chosen without knowledge of the learner's actions and lie in a bounded set. In this case, the setting is called **adversarial bandits**. The trick is to restrict the competitor class. We usually choose Π to be the set of constant policies. By defining the regret in this way, the stationarity assumption is transformed to the definition of regret rather than constraining the environment.

3) Of course, except of the two previous situations there exist other cases. Sometimes we consider the case where the rewards are stochastic, but not stationary or may analyze the robustness of an algorithm for stochastic bandits to small adversarial perturbations.

4) Limitations of Bandit Framework: One of the features of all bandit problems is that the learner never needs to plan for the future. More precisely, problems with the assumption that the learner's available choices and rewards tomorrow are not affected by their decisions today fall into the realm of **reinforcement learning**. Another limitation of the bandit framework is the assumption that the learner observes the reward in every round. The setting where the reward is not observed is called **partial monitoring.**

2.2 Applications

i) ***A/B Testing***

A/B testing (also known as split testing or bucket testing) is a methodology for comparing two versions of a webpage or app against each other to determine which one performs better. A/B testing is essentially an experiment where two or more variants of a page are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal. For example, the designers of a company website are trying to decide whether the "buy it now" button should be placed at the top of the product page or at the bottom.

One way to apply bandits to this problem is to view the two versions of the site as actions. Each time t a user makes a request, a bandit algorithm is used to choose an action $A_t \in \mathcal{A} = \{siteA, siteB\}$, and the reward is $X_t = 1$ if the user purchases the product and $X_t = 0$ otherwise.

ii) ***Advertisement Placement***

Advertisement placement means the group of units that specify the areas on the website where advertisers can place. Ad placement criteria include the size, type, and location of the ads. One way to face this is to view it as a multi-armed bandit problem, where in each round a policy chooses an action $A_t \in \mathcal{A} = \{\text{all available adverts}\}$, and the reward $X_t = 1$ if the user clicked on the advert and $X_t = 0$ otherwise. This might work for specialized websites, where the advertisements are likely to be appropriate. But for companies, like Amazon, advertising should be targeted. Clearly, an algorithm should take into account previous purchases. This might mean clustering users and implementing a separate bandit algorithm for each cluster. For example, a user that recently purchased a boxing bag is more likely to buy boxing gloves than another user. Also, other metrics such as user satisfaction, diversity, freshness and fairness, just to mention a few, are important too.

iii) ***Recommendation Services***

Netflix needs to choose which movies to display on each user's "browse" page. Similar to placing advertisements, users visit the page one after another, and the success of the selection can be measured based on whether the user watches a movie and rates it positively. However, there are some difficulties. Netflix presents a vast list of movies, leading to a large number of possible actions. Moreover, each user is unique and only watches a few movies, so the algorithm's choices influence the available data.

iv) ***Network Routing***

Routing is the process of path selection in any network. A computer network is made of many machines, called nodes, and paths or links that connect those nodes. Communication between two nodes in an interconnected network can take place through many different paths. Routing is the process of selecting the best path using some predetermined rules. In this problem, the learner receives the start/end destinations for a packet of data in each round. The set of actions is the set of all paths starting and ending at the appropriate points on some known graph. The feedback in this case is the time it takes for the packet to be received at its destination, and the reward is the negative of this value.

v) ***Dynamic Pricing***

In a dynamic pricing scenario, a company aims to find the best price for a product. Users arrive one by one, and the learning algorithm sets the price. A user will buy the product only if the price is lower than their personal value for it. However, the algorithm never directly observes the user's value; it only receives a binary signal indicating whether the price was too low/too high for the user.

vi) ***Waiting Problems***

In these problems, the challenge is to devise a policy for choosing, for example, how long to wait at the bus stop before giving up and walking to minimize the time to get to the workplace or deciding the amount of inactivity required before putting a hard drive into sleep mode or powering off a car engine at traffic lights. The statistical part concerns estimating the cumulative distribution function of the bus arrival times from data.

vii) ***Tree Search***

The UCT (**U**pper **C**onfidence bounds applied to **T**rees) algorithm is a tree search algorithm commonly used in perfect-information game-playing algorithms. The idea is to build a search tree where in each iteration the algorithm takes three steps: (a) chooses a path from the root to a leaf, (b) expands the leaf, if possible, (c) performs a Monte Carlo roll-out to the end of the game. The contribution of a bandit algorithm is in selecting the path from the root to the leaves. At each node in the tree, a bandit algorithm is used to select the child based on the series of rewards observed through that node so far.

3 Stochastic Bandits

3.1 Introduction

Stochastic bandits are a class of online learning problems where a decision maker or a learner has to repeatedly choose actions from a set of available options, also known as arms, and receive rewards that depend on the selected action. In contrast to traditional multi-

armed bandits, where the rewards are deterministic, stochastic bandits assume that the rewards are drawn from some unknown probability distributions associated with each arm.

The objective of the learner is to maximize the cumulative reward over a finite or infinite time horizon by learning the optimal arm selection strategy through trial and error. To achieve this objective, the learner uses various exploration-exploitation strategies that balance the trade-off between exploring new arms to learn their reward distributions and exploiting the already learned information to select the arms with the highest expected rewards.

Stochastic bandits have numerous applications in various fields, including recommendation systems, online advertising, clinical trials, and finance. The problem of designing efficient algorithms for stochastic bandits is an active research area in machine learning and optimization, and has led to the development of a variety of algorithms with different trade-offs between computational complexity, regret bounds, and practical performance.

3.1.1 Core Assumptions

A **stochastic bandit** is a collection of distributions $v = (P_a : \alpha \in \mathcal{A})$, where $\mathcal{A}$ is the set of available actions. In each round $t \in \{1, \ldots, n\}$, the learner chooses an action $A_t \in \mathcal{A}$, which is fed to the environment. The environment then samples a reward $X_t \in \mathbb{R}$ from distribution P_{A_t} and reveals X_t to the learner. The interaction between the learner and environment induces a probability measure on the sequence of outcomes $A_1, X_1, \ldots, A_n, X_n$. Usually the horizon n is finite, but sometimes we allow the interaction to continue indefinitely ($n = \infty$). The sequence of outcomes should satisfy the following assumptions:

(a) The conditional distribution of reward X_t given $A_1, X_1, \ldots, A_{t-1}, X_{t-1}, A_t$ is P_{A_t}, which captures the intuition that the environment samples X_t from P_{A_t} in round t.

(b) The law of action A_t given $A_1, X_1, \ldots, A_{t-1}, X_{t-1}$ is $\pi_t(\cdot \,|A_1, X_1, \ldots, A_{t-1}, X_{t-1})$, where $\pi_1, \pi_2, \ldots$ is a sequence of probability kernels that characterize the learner. The most important element of this assumption is the intuitive fact that the learner cannot use the future observations in current decisions.

3.1.2 Knowledge and Environment Classes

As mentioned, the learner's goal is to maximize the cumulative reward $S_n = \sum_{t=1}^{n} X_t$. However, very often the learner does not know ahead of time how many rounds are to be played. Even if the horizon is known in advance and we commit to maximizing the expected value of S_n, there is still the problem that the bandit instance $v = (P_a : a \in \mathcal{A})$ is unknown. A policy that maximizes the expectation of S_n for one bandit instance may behave quite badly on another. The learner usually has partial information about v, which we represent by defining a set of bandits $\mathcal{E}$ for which $v \in \mathcal{E}$ is guaranteed. The set $\mathcal{E}$ is called the **environment class**, which is distinguished between **structured** and **unstructured** bandits.

- **Unstructured Bandits**

An environment class $\mathcal{E}$ is unstructured if $\mathcal{A}$ is finite and there exist sets of distribution M_a for each $a \in \mathcal{A}$ such that: $\mathcal{E} = \{v = (P_a : a \in \mathcal{A}) : P_a \in M_a, \forall a \in \mathcal{A}\}$.

Name	Symbol	Definition
Bernoulli	$\mathcal{E}_{\mathcal{B}}^{k}$	$\{(\mathcal{B}(\mu_i))_i : \mu \in [0,1]^k\}$
Uniform	$\mathcal{E}_{\mathcal{U}}^{k}$	$\{(\mathcal{U}(a_i, b_i))_i : a, b \in \mathbb{R}^k \text{ with } a_i \leq b_i \text{ for all } i\}$
Gaussian (known var.)	$\mathcal{E}_{\mathcal{N}}^{k}(\sigma^2)$	$\{(\mathcal{N}(\mu_i, \sigma^2))_i : \mu \in \mathbb{R}^k\}$
Gaussian (unknown var.)	$\mathcal{E}_{\mathcal{N}}^{k}$	$\{(\mathcal{N}(\mu_i, \sigma_i^2))_i : \mu \in \mathbb{R}^k \text{ and } \sigma^2 \in [0,\infty)^k\}$
Finite variance	$\mathcal{E}_{\mathbb{V}}^{k}(\sigma^2)$	$\{(P_i)_i : \mathbb{V}_{X \sim P_i}[X] \leq \sigma^2 \text{ for all } i\}$
Finite kurtosis	$\mathcal{E}_{\mathrm{Kurt}}^{k}(\kappa)$	$\{(P_i)_i : \mathrm{Kurt}_{X \sim P_i}[X] \leq \kappa \text{ for all } i\}$
Bounded support	$\mathcal{E}_{[a,b]}^{k}$	$\{(P_i)_i : \mathrm{Supp}(P_i) \subseteq [a,b]\}$
Subgaussian	$\mathcal{E}_{\mathrm{SG}}^{k}(\sigma^2)$	$\{(P_i)_i : P_i \text{ is } \sigma\text{-subgaussian for all } i\}$

Note: 1) Some examples of unstructured bandits are presented in the table.

2) The Bernoulli, Gaussian, and uniform distributions are frequently utilized as examples to demonstrate specific characteristics of learning in stochastic bandit problems. In these problems, a bandit scenario is often referred to as a "distribution bandit", with the term "distribution" being replaced by the actual underlying distribution from which the pay-offs are sampled (Gaussian bandit, Bernoulli bandit, or subgaussian bandit). Additionally, we can refer to "bandits with X", where "X" represents a particular property of the underlying distribution from which the pay-offs are sampled. For example, bandits with finite variance

refer to the bandit environment where the learner's prior knowledge indicates that all pay-off distributions have finite variances.

3) Some environment classes, like Bernoulli bandits, are **parametric**, while others are **non-parametric,** like subgaussian bandits. The distinction is the number of degrees of freedom needed to describe an element of an environment class. When the number of degrees of freedom is finite, it is parametric, otherwise it is non-parametric. Furthermore, some environment classes are subsets of others. For example, Bernoulli bandits are a special case of bandits with finite variance or bandits with bounded support.

- **Structured Bandits**

Environment classes that are not unstructured are called structured. A significant feature of structured bandits is that the learner can often obtain information about some actions while never playing them. The following examples illustrate the flexibility of these problems.

Example 1: Let $\mathcal{A} = \{1,2\}$ and $\mathcal{E} = \{(B(\theta), B(1-\theta)) : \theta \in [0,1]\}$. The learner does not know the mean of either arm, but can learn the mean of both arms by playing just one. The difficulty of learning in this problem is changed by the knowledge of this structure.

Example 2(Stochastic linear bandit): Let $\mathcal{A} \subset \mathbb{R}^d, \theta \in \mathbb{R}^d$, $v_\theta = (N(< a, \theta >, 1) : a \in \mathcal{A})$ and $\mathcal{E} = \{v_\theta : \theta \in \mathbb{R}^d\}$. The reward of an action is Gaussian, and its mean is given by the inner product between the action and some unknown parameter. Notice that even if $\mathcal{A}$ is extremely large, the learner can deduce the true environment by playing just d actions that span $\mathbb{R}^d$.

3.1.3 The Regret

Let $v = (P_a : a \in \mathcal{A})$ be a stochastic bandit and define $\mu_\alpha(v) = \int_{-\infty}^{\infty} x dP_\alpha(x)$. Then let $\mu^*(v) = max_{a \in \mathcal{A}} \mu_\alpha(v)$ be the largest mean of all arms. The regret of policy π on bandit instance v is $R_n(\pi, v) = n\mu^*(v) - \mathbb{E}[\sum_{t=1}^{n} X_t]$, where the expectation is taken with respect

to the probability measure on outcomes induced by the interaction of π and v. Minimizing the regret is equivalent to maximizing the expectation of S_n.

The regret is always non-negative, and for every bandit v, there exists a policy π for which the regret is zero (the best possible outcome), i.e., $R_n(\pi, v) \geq 0$, for all policies π and the policy choosing $A_t \in argmax_\alpha \mu_\alpha$ for all t satisfies $R_n(\pi, v) = 0$. If $R_n(\pi, v) = 0$ for some policy π, then $\mathbb{P}(\mu_{A_t} = \mu^*) = 1$ for all $t \in [n]$, which means achieving zero is possible if and only if the learner knows which bandit it is facing or at least what is the optimal arm.

There is another candidate objective called the **Bayesian regret**. If Q is a prior probability measure on $\mathcal{E}$ (which must be equipped with a σ-algebra $\mathcal{F}$), then the Bayesian regret is the average of the regret with respect to the prior Q ,$BR_n(\pi, Q) = \int_{\mathcal{E}} R_n(\pi, v) dQ(v)$, which is only defined by assuming that the regret is a measurable function with respect to $\mathcal{F}$. An advantage of the Bayesian approach is that the problem of finding a policy that minimizes the Bayesian regret is just an optimization problem, although generally very difficult.

3.2 Stochastic Bandits with Finitely Many Arms

3.2.1 The Explore-Then-Commit(ETC) Algorithm And Regret Analysis

The **ETC** algorithm is characterized by the number of times it explores each arm, denoted by a natural number m. Because there are k actions, the algorithm will explore for mk rounds before choosing a single action for the remaining rounds. Let $\hat{\mu}_i(t)$ be the average reward received from arm i after round t, $\hat{\mu}_i(t) = \frac{1}{T_i(t)} \sum_{s=1}^{t} \mathbb{I}\{A_s = i\} X_s$, where $T_i(t) = \sum_{s=1}^{t} \mathbb{I}\{A_s = i\} X_s$ is the number of times action i has been played after round t. The **ETC** policy is given below:

Step 1: Input m

Step 2: In round t choose action $A_t = \begin{cases} (t \bmod k) + 1, if\ t \leq mk \\ argmax_i \hat{\mu}_i(mk), if\ t > mk \end{cases}$

Theorem 1:

According to Tor Lattimore and Csaba Szepesvari, when **ETC** is interacting with any 1-subgaussian bandit and $1 \leq \mathrm{m} \leq \frac{\mathrm{n}}{\mathrm{k}}$:

$$R_n \leq m \sum_{i=1}^{k} \Delta_i + (n - mk) \sum_{i=1}^{k} \Delta_i exp(-\frac{m{\Delta_i}^2}{4})$$

Note: 1) Let $v = (P_a : a \in A)$ be a stochastic bandit. We define $\Delta_\alpha(v) = \mu^*(v) - \mu_\alpha(v)$, which is called the **suboptimality gap** or **action gap** or **immediate regret** of action a.

2) **Definition 2**: A random variable X is σ-subgaussian if for all $\lambda \in \mathbb{R}$, it holds that $\mathbb{E}[exp(\lambda X)] \leq exp\left(\frac{\lambda^2\sigma^2}{2}\right)$.

3) (**Regret decomposition lemma**) For any policy π and stochastic bandit environment v with $\mathcal{A}$ finite and horizon $n \in \mathbb{N}$, the regret R_n of policy π in v satisfies: $R_n = \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}[T_a(n)]$. This tells us that to keep the regret small, the learner should try to minimize the weighted sum of expected action counts, where the weights are the respective suboptimality gaps.

4) The bound in the previous Theorem illustrates the trade-off between exploration and exploitation. i) If m is large, then the policy explores for too long, and the first term will be large. ii) If m is too small, then the probability that the algorithm commits to the wrong arm will grow, and the second term becomes large. The question is how to choose m.

Example:

Assume that $k = 2$ and that the first arm is optimal so that $\Delta_1 = 0$ and $\Delta = \Delta_2$. Then the bound simplifies to: $R_n \leq m\Delta + (n - 2m)\Delta \exp\left(-\frac{m\Delta^2}{4}\right) \leq m\Delta + n\Delta \exp\left(-\frac{m\Delta^2}{4}\right)$

For large n the quantity on the right-hand side is minimized up to a possible rounding error by $m = max\left\{1, \left\lceil \frac{4}{\Delta^2} \log\left(\frac{n\Delta^2}{4}\right)\right\rceil\right\}$ and for this choice and any n, the regret is bounded by $R_n \leq min\left\{n\Delta, \Delta + \frac{4}{\Delta}\left(1 + max\left\{0, \log\left(\frac{n\Delta^2}{4}\right)\right\}\right)\right\}$. Bounds like this are called **gap/problem/distribution/instance dependent**. The bound is close to optimal, but there is a caveat. The choice of m that defines the policy and leads to this bound depends on both the suboptimality gap and the horizon. While the horizon is sometimes known in advance, it is seldom reasonable to assume knowledge of the suboptimality gap.

5) It is proven that the previous relation can also be written as: $R_n \leq \Delta + C\sqrt{n}$, where $C > 0$ is a universal constant. Sometimes, is often assumed that $\Delta \leq 1$ and thus $R_n \leq 1 + C\sqrt{n}$. Bounds of this type are called **worst-case**, **problem free** or **problem independent.** The reason is that the bound only depends on the horizon and class of bandits for which the algorithm is designed, and not the specific instance within that class. Because the suboptimality gap does not appear, bounds like this are sometimes called **gap-free.** Note that without the condition $\Delta \leq 1$, the worst-case bound for **ETC** is infinite.

6) i) **Epsilon-greedy algorithm**: is a randomized analog of **ETC** with the only difference that exploration is spread more uniformly over time. This algorithm depends on a sequence of parameters $\varepsilon_1, \varepsilon_2, \ldots$. First, it chooses each arm once and subsequently chooses

$$\begin{cases} A_t = argmax_i \hat{\mu}_i(t-1), \text{with probability } 1-\varepsilon_t \\ \text{an arm uniformly at random, otherwise} \end{cases} \quad [6]$$

ii) **Elimination Algorithm**: A simple way to generalize the **ETC** policy to multiple arms and overcome the problem of tuning the commitment time is to use an elimination algorithm. The algorithm operates in phases and maintains an active set of arms that could be optimal.

Step 1: Input k and sequence $(m_l)_l$

Step 2: $A_1 = \{1,2, \ldots, k\}$

Step 3: For $l = 1,2,3 \ldots$ do

a) Choose each arm $i \in A_l$ exactly m_l times

b) Let $\hat{\mu}_{i,l}$ be the average reward for arm i from this phase only

c) Update active set: $A_{l+1} = \left\{i: \hat{\mu}_{i,l} + 2^{-l} \geq max_{j \in A_l} \hat{\mu}_{j,l}\right\}$

Step 4: End for

3.2.2 The Upper Confidence Bound Algorithm

The **UCB** algorithm is based on the principle of **optimism in the face of uncertainty**, which states that one should act as if the environment is as nice as plausibly possible. For bandits, the optimism principle means using the data observed so far to assign to each arm a value, called the **upper confidence bound** that with high probability is an overestimate of the unknown mean. Comparatively with **ETC**, this algorithm does not depend on advance

knowledge of the suboptimality gap, but on the horizon n and behaves well where there are more than two arms. [1]

Let $(X_t)_{t=1}^n$ be a sequence of independent 1-subgaussian random variables with mean μ and $\hat{\mu} = \frac{1}{n}\sum_{t=1}^n X_t$. A version of **UCB** algorithm, which takes as input the number of arms and the error probability δ, is given below:

Step 1: Input k and δ

Step 2: For $t \in 1, \dots, n$ do

a) Compute $UCB_i(t-1,\delta) = \begin{cases} \infty, & if\ T_i(t-1) = 0 \\ \hat{\mu}_i(t-1) + \sqrt{\frac{2\log\left(1/\delta\right)}{T_i(t-1)}}, & \text{otherwise} \end{cases}$

b) Choose action $A_t = argmax_i UCB_i(t-1,\delta)$

c) Observe reward X_t and update upper confidence bound

Step 3: End for

Definition 3: The value inside the $argmax$ is called the **index** of arm i. The index is the sum of the empirical mean of rewards experienced so far and the **exploration bonus**, which is also known as the **confidence width**.

Observation: After the initial period where the algorithm chooses each action once, action i can only be chosen if its index is higher than that of an optimal arm. This can only happen if:

(a) The index of action i is larger than the true mean of a specific optimal arm.

(b) The index of a specific optimal arm is smaller than its true mean.

Considering that the index of any arm typically serves as an upper bound for its mean with high probability, we can reasonably expect that the index of the optimal arm will not be below its mean. Additionally, if the suboptimal arm i is played sufficiently frequently, its exploration bonus decreases, and the empirical estimate of its mean converges to the true value. Consequently, this imposes an upper limit on the expected total number of occurrences when the index of the suboptimal arm remains above the mean of the optimal arm.

Note: At the start of round t the first arm has been played much more frequently than the rest. Because it has been played so often, we expect that $\hat{\mu}_1(t-1) \approx \mu_1$. The learner can be reasonably certain that arm i is worse than arm 1 if:

$\hat{\mu}_i(t-1) + \sqrt{\frac{2\log\left(1/\delta\right)}{T_i(t-1)}} \leq \mu_1 \approx \hat{\mu}_1(t-1) + \sqrt{\frac{2\log\left(1/\delta\right)}{T_1(t-1)}}$, where δ is called the **confidence level** and quantifies the degree of certainty. This means that choosing the arm with the largest upper confidence bound leads to a situation where arms are only chosen if their true mean could reasonably be larger than those of arms that have been played often. The confidence level should be small to ensure optimism with high probability, but not so large that suboptimal arms are explored excessively.

Theorem 2:

1) Consider **UCB**, as shown in previous algorithm, on a stochastic k-armed 1-subgaussian bandit problem. For any horizon n, if $\delta = 1/n^2$, then: $R_n \leq 3\sum_{i=1}^{k} \Delta_i + \sum_{i:\Delta_i>0} \frac{16\log(n)}{\Delta_i}$.

2) If $\delta = 1/n^2$, then the regret of **UCB**, as defined in previous algorithm, on any $v \in \mathcal{E}_{SG}^{k}(1)$ environment, is bounded by: $R_n \leq 8\sqrt{nk\log(n)} + 3\sum_{i=1}^{k} \Delta_i$. This result is close to optimal. [1]

3.2.3 The UCB Algorithm: Asymptotic Optimality

Step 1: Input k

Step 2: Choose each arm once

Step 3: Subsequently choose $A_t = argmax_i \left(\hat{\mu}_i(t-1) + \sqrt{\frac{2\log f(t)}{T_i(t-1)}}\right)$,

where $f(t) = 1 + t\log^2(t)$

Note: This algorithm differs from the one mentioned in the previous sector only by the choice of the confidence level, the choice of which is dictated by the analysis of its regret.

Theorem 3:

According to Tor Lattimore and Csaba Szepesvari, for any 1-subgaussian bandit, the regret of the previous algorithm satisfies:

$$R_n \leq \sum_{i:\Delta_i>0} inf_{\varepsilon\in(0,\Delta_i)}\Delta_i\left(1+\frac{5}{\varepsilon^2}+\frac{2(\log f(n)+\sqrt{\pi\log f(n)}+1}{(\Delta_i-\varepsilon)^2}\right)$$

Furthermore, by choosing $\varepsilon = \log^{-1/4}(n)$ and taking the limit as n tends to infinity, we take that: $\limsup_{n\to\infty}\frac{R_n}{\log(n)} \leq \sum_{i:\Delta_i>0}\frac{2}{\Delta_i}$. Choosing $\varepsilon = {\Delta_i}/{2}$ inside the sum shows that:

$$R_n \leq \sum_{i:\Delta_i>0}\left(\Delta_i+\frac{1}{\Delta_i}\left(8\log f(n)+8\sqrt{\pi\log f(n)}+28\right)\right)$$

Even more precisely, there exists some universal constant $C > 0$ such that:

$$R_n \leq C\sum_{i:\Delta_i>0}\left(\Delta_i+\frac{\log(n)}{\Delta_i}\right)$$

which leads to a worst-case bound of:

$$R_n \leq C\sum_{i=1}^{k}\Delta_i + 2\sqrt{Cnk\log(n)}$$

Note: The dominant terms in the two results have the same order, but the gain here is that in this result the leading constant, governing the asymptotic rate of growth of regret, is smaller.

3.2.4 Lower Bounds

In stochastic bandits, lower bounds refer to the minimum number of samples required to achieve a certain level of accuracy in estimating the mean reward of a given arm in a multi-armed bandit problem. Specifically, the lower bounds provide a theoretical guarantee on the performance of any algorithm used to solve the problem. These bounds can be established using information-theoretic arguments, which take into account the amount of information contained in the rewards obtained from the arms.

Lower bounds in stochastic bandits are useful because they give a fundamental limit on the performance of any algorithm, and can be used to assess the effectiveness of proposed algorithms. In particular, any algorithm that achieves a performance equal to the established lower bound is considered to be optimal. They can be derived using various techniques such as information theory, statistical learning theory, and Bayesian analysis. The specific

technique used depends on the assumptions made about the distribution of rewards and the algorithm being considered.

Definition 4: The **worst-case regret** of a policy π on a set of stochastic bandit environments $\mathcal{E}$ is $R_n(\pi, \mathcal{E}) = sup_{v \in \mathcal{E}} R_n(\pi, v)$. Let Π be the set of all policies. The **minimax regret** is:

$$R_n^*(\mathcal{E}) = inf_{\pi \in \Pi} R_n(\pi, \mathcal{E}) = inf_{\pi \in \Pi} sup_{v \in \mathcal{E}} R_n(\pi, v)$$

A policy is called **minimax optimal** for $\mathcal{E}$ if $R_n(\pi, \mathcal{E}) = R_n^*(\mathcal{E})$. The value $R_n^*(\mathcal{E})$ is of interest by itself. A small value of $R_n^*(\mathcal{E})$ indicates that the underlying bandit problem is less challenging in the worst-case sense. Minimax optimality is not a property of a policy alone. It is a property of a policy together with a set of environments and a horizon.

Note: 1) A policy is minimax optimal up to constant factors for finite-armed 1-subgaussian bandits with suboptimality gaps in $[0, 1]$, when there exists a $C > 0$ such that:

$$\frac{R_n(\pi, \mathcal{E}^k)}{R_n^*(\mathcal{E}^k)} \leq C, \text{for all } k \text{ and } n$$

where $\mathcal{E}^k$ is the set of k-armed 1-subgaussian bandits with suboptimality gaps in $[0, 1]$.

2) **Main Ideas Underlying Minimax Lower Bounds**:

Minimax lower bounds are a fundamental concept in the theory of algorithms and complexity. These bounds provide a theoretical guarantee on the minimum amount of resources (such as time, space, or samples) required to solve a given problem, regardless of the algorithm used. The main idea underlying minimax lower bounds is to consider the worst-case scenario for any algorithm solving the problem. That is, the bounds assume that an adversary is trying to make the algorithm fail by carefully selecting the input data to be as difficult as possible. More specifically, the minimax lower bound provides a guarantee on the best possible performance that any algorithm can achieve when facing the worst possible input data. This guarantee is expressed in terms of the resources required to achieve a certain level of performance, such as the number of comparisons or the number of samples required to estimate a parameter. The minimax lower bound is established by constructing an adversary that selects the input data to be as difficult as possible, and then showing that any algorithm that solves the problem using fewer resources than the lower bound will fail

on at least one instance of the problem. The lower bound is then the minimum amount of resources required achieving a certain level of performance on all possible input data. The minimax lower bound is an important concept in the analysis of algorithms and complexity because it provides a rigorous benchmark against which the performance of any algorithm can be measured. It also provides insights into the inherent difficulty of a problem, and can guide the design of algorithms that are provably optimal in terms of their resource usage. [1]

Figure: Given a policy and one environment, the evil antagonist picks another environment so that the policy will suffer a large regret in at least one environment.

Theorem 4:

Let v_μ be the Gaussian bandit for which the ith arm has reward distribution $\mathcal{N}(\mu_i, 1)$, $k > 1$ and $n \geq k - 1$. Then, for any policy π, there exists a mean vector $\mu \in [0,1]^k$ such that:

$$R_n(\pi, v_\mu) \geq \frac{1}{27}\sqrt{(k-1)n}$$

Since $v_\mu \in \mathcal{E}_N^\kappa(1)$, it follows that the minimax regret for $\mathcal{E}_N^\kappa(1)$ is lower-bounded by the right-hand side of the above display as soon as $n \geq k - 1$:

$$R_n^*(\mathcal{E}_N^\kappa(1)) \geq \frac{1}{27}\sqrt{(k-1)n}$$

[1]

Definition 5: The **random pseudo-regret** is: $\bar{R}_n = \sum_{i=1}^k T_i(n)\Delta_i$

Theorem 5 (High-Probability Lower Bounds):

Let $n \geq 1$ and $k \geq 2$ and $B > 0$ and π be a policy such that for any $v \in \mathcal{E}^k$,

$$R_n(\pi, v) \leq B\sqrt{(k-1)n}$$

Let $\delta \in (0,1)$. Then there exists a bandit v in $\mathcal{E}^k$ such that:

$$\mathbb{P}\left(\bar{R}_n(\pi, v) \geq \frac{1}{4} min\left\{n, \frac{1}{B}\sqrt{(k-1)n}\log\left(\frac{1}{4\delta}\right)\right\}\right) \geq \delta$$

[1]

Corollary:

1) Let $n \geq 1$ and $k \geq 2$. Then, for any policy π and $\delta \in (0,1)$ such that

$$n\delta \leq \sqrt{n(k-1)log\left(\frac{1}{4\delta}\right)}$$

there exists a bandit problem $v \in \mathcal{E}^k$ such that:

$$\mathbb{P}\left(\bar{R}_n(\pi, v) \geq \frac{1}{4} min\left\{n, \sqrt{\frac{n(k-1)}{2}} log\left(\frac{1}{4\delta}\right)\right\}\right) \geq \delta$$

2) Let $k \geq 2$ and $p \in (0,1)$ and $B > 0$. Then, there does not exist a policy π such that for all $n \geq 1$, $\delta \in (0,1)$ and $v \in \mathcal{E}^k$,

$$\mathbb{P}\left(\bar{R}_n(\pi, v) \geq B\sqrt{(k-1)n}\log^p\left(\frac{1}{\delta}\right)\right) < \delta$$

4 Adversarial Bandits

4.1 Abstract

Adversarial bandits is a class of online decision-making problems where a learner, or decision-maker, must repeatedly choose one of several actions while facing an adversary who can dynamically manipulate the rewards associated with each action. This problem arises in many real-world applications, such as online advertising, recommendation systems, and cybersecurity.

The goal of the learner is to maximize its total reward over a finite time horizon, despite the adversary's attempts to minimize it. The key challenge in adversarial bandits is to balance the exploration of different actions to learn their rewards with the exploitation of actions that appear to be rewarding based on the agent's past experience. To address this challenge, various algorithms have been developed that use a combination of exploration and exploitation strategies to learn the rewards of the actions over time. These algorithms typically employ techniques such as optimism in the face of uncertainty, regret minimization, and exploration based on the principle of optimism under uncertainty.

4.1.1 Adversarial Bandit Environments

Let $k > 1$ be the number of arms. A ***k*-armed adversarial bandit** is an arbitrary sequence of reward vectors $(x_t)_{t=1}^{n}$, where $x_t \in [0,1]^k$ .In each round, the learner chooses a distribution over the actions $P_t \in P_{k-1}$. Then the action $A_t \in [k]$ is sampled from P_t, and the learner receives reward x_{tA_t}.

A policy in this setting is a function $\pi: ([k] \times [0,1]) \to P_{k-1}$ mapping history sequences to distributions over actions (regardless of measurability). The performance of a policy π in environment x is measured by the expected regret, which is the expected loss in revenue of the policy relative to the best fixed action in hindsight.

$$R_n(\pi, x) = \max_{i \in [k]} \sum_{t=1}^{n} x_{ti} - \mathbb{E}\left[\sum_{t=1}^{n} x_{tA_t}\right]$$

The worst-case regret over all environments is: $R_n^*(\pi) = sup_{x \in [0,1]^{n \times k}} R_n(\pi, x)$.

4.1.2 Similarities-Differences between Stochastic and Adversarial Environments

Stochastic and adversarial environments are two different types of environments that a learner can encounter in the context of decision-making problems. One key difference between stochastic and adversarial environments is the source of uncertainty. In a stochastic environment, the uncertainty arises from the randomness or probabilistic nature of the environment. For example, in a game of dice, the outcome of each roll is determined randomly, and the agent cannot predict it with certainty. In contrast, in an adversarial environment, the uncertainty arises from the actions of an adversary who actively tries to

manipulate the outcomes to its advantage. For example, in a game of chess, the opponent may make moves to try to undermine the agent's strategy. Another difference between stochastic and adversarial environments is the nature of the information available to the agent. In a stochastic environment, the agent typically has access to some probabilistic information about the environment, such as the probabilities of different outcomes. In contrast, in an adversarial environment, the agent may have very limited information about the adversary's actions and intentions.

Despite these differences, there are also some similarities. In both cases, the learner must make decisions under uncertainty and must balance the trade-off between exploration and exploitation to maximize its expected reward. Moreover, in both cases, the learner can use some form of learning to improve its decision-making over time.

4.1.3 Importance-Weighted Estimators

Definition 6: A key ingredient of all adversarial bandit algorithms is a mechanism for estimating the reward of unplayed arms. Recall that P_t is the conditional distribution of the action played in round t, and so for $i \in [k]$, $P_{ti} = \mathbb{P}(A_t = i | A_1, X_1, \dots, A_{t-1}, X_{t-1})$. The **importance-weighted estimator** of x_{ti} is:

$$\hat{X}_{ti} = \frac{\mathbb{I}\{A_t = i\} X_t}{P_{ti}} \text{ (I)}$$

An alternative estimator is $\hat{X}_{ti} = 1 - \frac{\mathbb{I}\{A_t = i\}}{P_{ti}}(1 - X_t)$. Rewriting the formula in terms of $y_{ti} = 1 - x_{ti}$ and $Y_t = 1 - X_t$ and $\hat{Y}_{ti} = 1 - \hat{X}_{ti}$ leads to:

$$\hat{Y}_{ti} = \frac{\mathbb{I}\{A_t = i\} Y_t}{P_{ti}} \text{ (II)}$$

This is the same as the previous formula except that Y_t has replaced X_t. The terms $y_{ti}, Y_t, \hat{Y}_{ti}$ should be interpreted as **losses**. The estimator in last equation is called the **loss-based importance-weighted estimator**.

Note: Let $\mathbb{E}[\cdot] = \mathbb{E}[\cdot | A_1, X_1, \dots, A_t, X_t]$ denote the conditional expectation given the history up to time t. The conditional mean of $\hat{X}_{ti}$ satisfies $\mathbb{E}_{t-1}[\hat{X}_{ti}] = \mathbb{E}_{t-1}\left[\frac{A_{ti}}{P_{ti}} x_{ti}\right] = \frac{x_{ti}}{P_{ti}} \mathbb{E}_{t-1}[A_{ti}] = \frac{x_{ti}}{P_{ti}} P_{ti} = x_{ti}$, which means that $\hat{X}_{ti}$ is an unbiased estimate of x_{ti}

conditioned on the history observed after $t-1$ rounds. The estimator in equation (II) is still unbiased.

The conditional variance of $\hat{X}_{ti}$ satisfies: $\mathbb{V}_{t-1}\left[\hat{X}_{ti}\right] = \mathbb{E}_{t-1}\left[\hat{X}_{ti}^2\right] - {x_{ti}}^2 = \mathbb{E}_{t-1}\left[\frac{A_{ti}}{{P_{ti}}^2}{x_{ti}}^2\right] - {x_{ti}}^2 = \frac{{x_{ti}}^2(1-P_{ti})}{P_{ti}}$.The conditional variance of $\hat{Y}_{ti}$ satisfies: $\mathbb{V}_{t-1}\left[\hat{Y}_{ti}\right] = \frac{{y_{ti}}^2(1-P_{ti})}{P_{ti}}$. The only difference is that the variance depends on ${y_{ti}}^2$ rather than ${x_{ti}}^2$. Which is better depends on the rewards for arm i, with smaller rewards suggesting the superiority of the first estimator and larger rewards (or small losses) suggesting the superiority of the second estimator.

4.2 The Exp3 Algorithm

Let $\hat{\mathrm{S}}_{\mathrm{ti}} = \sum_{\mathrm{s}=1}^{\mathrm{t}} \hat{\mathrm{X}}_{\mathrm{si}}$ be the total estimated reward by the end of round t, where $\hat{\mathrm{X}}_{\mathrm{si}} = 1 - \frac{\mathbb{I}\{A_s=i\}}{P_{si}}(1-X_s)$. . It seems natural to play actions with larger estimated reward with higher probability. While there are many ways to map $\hat{\mathrm{S}}_{\mathrm{ti}}$ into probabilities, a simple and popular choice is called **exponential weighting**, which for tuning parameter $\eta > 0$ sets:

$$P_{ti} = \frac{\exp\left(\eta \hat{S}_{t-1,i}\right)}{\sum_{j=1}^{k} \exp\left(\eta \hat{S}_{t-1,j}\right)}$$

The parameter η is called **learning rate**. When the learning rate is large, P_t concentrates about the arm with the largest estimated reward and the resulting algorithm exploits aggressively. For small learning rates, P_t is more uniform, and the algorithm explores more frequently. Note that as P_t concentrates, the variance of the importance-weighted estimators for poorly performing arms increases dramatically. The **Exp3** algorithm is given below:

Step 1: Input n, k, η

Step 2: Set $\hat{S}_{0i} = 0$ for all i

Step 3: For $t = 1, \dots, n$ do

a) Calculate the sampling distribution P_t: $P_{ti} = \frac{\exp\left[\eta \hat{S}_{t-1,i}\right]}{\sum_{\mathrm{j}=1}^{\mathrm{k}} \exp\left[\eta \hat{S}_{t-1,j}\right]}$

b) Sample $A_t \sim P_t$ and observe reward X_t

c) Calculate $\hat{S}_{ti}$: $\hat{S}_{ti} = \hat{S}_{t-1,i} + 1 - \frac{\mathbb{I}\{A_t=i\}(1-X_t)}{P_{ti}}$

Step 4: End for

4.2.1 Regret Analysis

Theorem 6:

Let $x \in [0,1]^{n\times k}$ and π be the policy of **Exp3** with learning rate $\eta = \sqrt{\log(k)/(nk)}$. Then,

$$R_n(\pi, x) \leq 2\sqrt{nk\log(k)}$$

Theorem 7:

Let $x \in [0,1]^{n\times k}$ be an adversarial bandit and π be the policy of **Exp3** with learning rate $\eta = \sqrt{2\log(k)/(nk)}$. Then,

$$R_n(\pi, x) \leq \sqrt{2nk\log(k)}$$

Note: The second theorem is an improved version of the first, for which the regret is smaller by a factor of $\sqrt{2}$. The algorithm is unchanged except for a slightly increased learning rate. [1]

4.2.2 The Exp3-IX Algorithm And Regret Analysis

The objective of this chapter is to modify **Exp3** so that the regret stays small in expectation and is simultaneously well concentrated about its mean. Such results are called **high-probability bounds**. The poor behavior of **Exp3** occurs because the variance of the importance-weighted estimators can become very large. In this chapter we modify the reward estimates to control the variance at the price of introducing some bias. Let $\gamma > 0$ be a small constant to be chosen later and define the biased estimator: $\hat{Y}_{ti} = \frac{\mathbb{I}\{A_t=i\}Y_t}{P_{ti}+\gamma}$ (3) . As γ increases, the predictable variance decreases, but the bias increases. When equation (3) is used in the exponential update in **Exp3**, the resulting algorithm is called **Exp3-IX**. The suffix *"IX"* stands for **i**mplicit e**x**ploration. Since small losses correspond to large rewards, the estimator is optimistically biased. The effect is a smoothing of P_t so that actions with large losses for which **Exp3** would assign negligible probability are still chosen occasionally. In fact, the smaller is P_{ti}, the larger the bias is. As a result, **Exp3-IX** will explore more than the standard **Exp3** algorithm. The **Exp-IX** algorithm is given below:

Step 1: Input n, k, η, γ

Step 2: Set $\hat{L}_{0i} = 0$ for all i, where $\hat{L}_{ni} = \sum_{t=1}^{n} \hat{Y}_{ti}$

Step 3: For $t = 1, \ldots, n$ do

a) Calculate the sampling distribution P_t: $P_{ti} = \frac{exp(-\eta \hat{L}_{t-1,i})}{\sum_{j=1}^{k} exp(-\eta \hat{L}_{t-1,j})}$

b) Sample $A_t \sim P_t$ and observe reward X_t

c) Calculate $\hat{L}_{ti} = \hat{L}_{t-1,i} + \frac{\mathbb{I}\{A_t = i\}(1 - X_t)}{P_{ti} + \gamma}$

Step 4: End for

Theorem 8:

Let $\delta \in (0,1)$ and define $\eta_1 = \sqrt{\frac{2\log(k+1)}{nk}}$ and $\eta_2 = \sqrt{\frac{\log(k) + \log\left(\frac{k+1}{\delta}\right)}{nk}}$ and $\hat{R}_n = \max_{a \in \mathcal{A}} \sum_{t=1}^{n} (y_{tA_t} - y_{ta})$.

1) If **Exp3-IX** is run with parameters $\eta = \eta_1$ and $\gamma = \eta/2$, then:

$$\mathbb{P}\left(\hat{R}_n \geq \sqrt{8nk\log(k+1)} + \sqrt{\frac{nk}{2\log(k+1)}}\log\left(\frac{1}{\delta}\right) + \log\left(\frac{k+1}{\delta}\right)\right) \leq \delta$$

2) If **Exp3-IX** is run with parameters $\eta = \eta_2$ and $\gamma = \eta/2$, then:

$$\mathbb{P}\left(\hat{R}_n \geq 2\sqrt{nk(2\log(k+1) + \log(1/\delta)} + \log\left(\frac{k+1}{\delta}\right)\right) \leq \delta$$

Note: The value of η_1 is independent of δ, which means that using this choice of learning rate leads to a single algorithm with a high-probability bound for all δ. On the other hand, η_2 does depend on δ, so the user must choose a confidence level from the beginning. The advantage is that the bound is improved, but only for the specified confidence level.

5 Contextual Bandits

5.1 Abstract

Contextual bandits are a type of online learning algorithm that aims to balance the exploration-exploitation tradeoff in decision-making problems. In a contextual bandit setting, the learning agent is presented with a set of contexts, or features that describe the state of the environment, and must select an action to take based on those features. Unlike traditional bandit algorithms, which assume that the environment remains stationary and the learner can sample actions from a fixed set of alternatives, contextual bandits must adapt to changing environments and select actions from a potentially infinite set of alternatives. The goal of the contextual bandit algorithm is to maximize the cumulative reward obtained over time, while also minimizing the regret of not selecting the optimal action. To achieve this, the algorithm must balance the need to gather new information with the desire to use what is already known.

5.1.1 Contextual Bandits: One Bandit Per Context

While contextual bandits can be studied in both the adversarial and stochastic frameworks, in this section we focus on the k-armed adversarial model. The interaction protocol is given below:

Step 1: Adversary secretly chooses rewards $(x_t)_{t=1}^n$ with $x_t \in [0,1]^k$

Step 2: Adversary secretly chooses contexts $(c_t)_{t=1}^n$ with $c_t \in \mathcal{C}$

Step 3: For rounds $t = 1, \dots, n$:

a) Learner observes context $c_t \in \mathcal{C}$, where $\mathcal{C}$ is an arbitrary fixed set of contexts

b) Learner selects distribution $P_t \in P_{k-1}$ and samples A_t from P_t

c) Learner observes reward $X_t = x_{tA_t}$

A natural way to define the regret is to compare the rewards collected by the learner with the rewards collected by the best context-dependent policy in hindsight:

$$R_n = \mathbb{E}\left[\sum_{c\in\mathcal{C}} \max_{i\in[k]} \sum_{t\in[n]:c_t=c} (x_{ti} - X_t)\right]$$

If the set of possible contexts is finite, then a simple approach is to use a separate instance of **Exp3** for each context. Let:

$$R_{nc} = \mathbb{E}\left[\max_{i\in[k]} \sum_{t\in[n]:c_t=c} (x_{ti} - X_t)\right]$$

be the regret due to context $c \in \mathcal{C}$. Combining these equations we conclude to:

$$R_n = \sum_{c\in\mathcal{C}} R_{nc} \leq 2\sum_{c\in\mathcal{C}} \sqrt{k\log(k)\sum_{t=1}^{n} \mathbb{I}\{c_t = c\}}$$

where the sum inside the square root counts the number of times context $c \in \mathcal{C}$ is observed and the magnitude of the right-hand side depends on the distribution of observed contexts.

5.1.2 Bandits with Expert Advice

Provided that the context set $\mathcal{C}$ is extensive, it is usually not a good idea to use a single bandit algorithm per context, unless there is an enormous amount of data available. However, in practice, the context space is often not just vast but structured, such as in the case of a movie recommendation system, where user demographics and movie genres provide some structure that can be exploited to improve the efficiency of the bandit algorithm.

The bandits with expert advice setting is a k-armed adversarial bandit, but with M experts making recommendations to the learner. At the beginning of each round, the experts announce their predictions about which actions are the most promising. The experts report a probability distribution over the actions. The interpretation is that the expert, if the decision were left to them, would choose the action for the round at random from the probability distribution it reported.

The predictions of the M experts in round t are represented by a matrix $E^{(t)} \in [0,1]^{M\times k}$, where the m-th row $E_m^{(t)}$ is a probability vector over $[m]$ representing the recommendations of expert m in round t. The interaction protocol is given below:

Step 1: Adversary secretly chooses rewards $x \in [0,1]^{n\times k}$

Step 2: Experts secretly choose predictions $E^{(1)}, \dots, E^{(n)}$

Step 3: For rounds $t = 1, \dots, n$:

a) Learner observes predictions of all experts, $E^{(t)} \in [0,1]^{M\times k}$

b) Learner selects a distribution $P_t \in P_{k-1}$

c) Action A_t is sampled from P_t and the reward is $X_t = x_{tA_t}$

5.1.3 The Exp4 Algorithm And Regret Analysis

The number 4 in Exp4 is not just an increased version number, but indicates the four e's in the long name of the algorithm, which is **e**xponential weighting for **e**xploration and **e**xploitation with **e**xperts. The pseudocode of **Exp4** is given below:

Step 1: Input n, k, M, η, γ

Step 2: Set $Q_1 = (1/M\ , \dots, 1/M) \in [0,1]^{1\times M}$ (a row vector)

Step 3: For rounds $t = 1, \dots, n$ do

a) Receive advice $E^{(t)}$

b) Choose the action $A_t \sim P_t$, where $P_t = Q_t E^{(t)}$

c) Receive the reward $X_t = x_{tA_t}$

d) Estimate the action rewards: $\hat{X}_{ti} = 1 - (1 - X_t)\frac{\mathbb{I}\{A_t=i\}}{P_{ti}+\gamma}$

e) Propagate the rewards to the experts: $\tilde{X}_t = E^{(t)}\hat{X}_t$

f) Update the distribution Q_t using the exponential weighting:

$$Q_{t+1,i} = \frac{\exp(\eta\tilde{X}_{ti})Q_{ti}}{\sum_j \exp(\eta\tilde{X}_{tj})Q_{tj}} \text{ for all } i \in [M]$$

Step 4: End for

The regret measures the cumulative rewards collected by the learner relative to the best expert in hindsight:

$$R_n = \mathbb{E}\left[\max_{m\in[M]} \sum_{t=1}^{n} E_m^{(t)} x_t - \sum_{t=1}^{n} X_t\right]$$

Theorem 9:

Let $\gamma = 0$ and $\eta = \sqrt{2\log(M)/(nk)}$ denote by R_n the expected regret of **Exp4** after n rounds. Then, $R_n \leq \sqrt{2nk\log(M)}$.

Note: This is the same bound we derived using an independent copy of **Exp3** for each context.

Theorem 10:

Assume the same conditions as the previous theorem, except let $\eta_t = \sqrt{\log(M)/E_t^*}$, where $E_t^* = \sum_{s=1}^{t} \sum_{i=1}^{k} \max_{m \in [M]} E_{mi}^{(s)}$, which shows if the experts have a high degree of agreement. Then there exists a universal constant $C > 0$ such that:

$$R_n \leq C\sqrt{E_n^* \log(M)}$$

Note: The bound tells us that **Exp4** with the suggested learning rate is able to adapt to degree of disagreement between the experts, which seems like quite an encouraging result. As a further benefit, the learning rate does not depend on the horizon.

5.2 Stochastic Contextual Bandits

At the beginning of round t, the learner observes a context $C_t \in \mathcal{C}$, which may be random or not. Having observed the context, the learner chooses their action $A_t \in [k]$ based on the information available. So far everything is the same as the adversarial setting. The difference comes from the assumption that the reward X_t satisfies $X_t = r(C_t, A_t) + \eta_t$, where $r: \mathcal{C} \times [k] \to \mathbb{R}$ is called the **reward function** and η_t is the noise, which we will assume is conditionally 1-subgaussian.

If r was given, then the action in round t with the largest expected return is $\mathrm{A_t^*} \in \arg\max_{\mathrm{a} \in [\mathrm{k}]} \mathrm{r}(\mathrm{C_t}, \mathrm{a})$. Notice that this action is now a random variable because it depends on the context C_t. The loss due to the lack of knowledge of r makes the learner incur the regret,

$$R_n = \mathbb{E}\left[\sum_{t=1}^{n} \max_{a \in [k]} \mathrm{r}(\mathrm{C_t}, \mathrm{a}) - \sum_{\mathrm{t}=1}^{\mathrm{n}} \mathrm{X_t}\right]$$

In stochastic linear bandits, the definition of regret assumes that the learner's actions do not significantly affect the subsequent contexts. However, in some practical scenarios, the context may depend on the learner's previous actions. In such cases, the definition of regret may not accurately reflect the learner's performance. To overcome this issue, a more general framework such as the contextual bandit framework can be used, which allows the

learner to select an action based on the context and receive a reward that depends on both the chosen action and the context. This approach can handle the dependence between the contexts and the actions more effectively than the traditional stochastic linear bandit framework.

Definition 7: A simple assumption to capture further information about the dependence of rewards on context is to assume that the learner has access to a map $\psi: \mathcal{C} \times [k] \to \mathbb{R}^d$, and for an unknown parameter vector $\theta_* \in \mathbb{R}^d$, it holds that: $r(c, a) = \langle \theta_*, \psi(c, a) \rangle$, for all $(c, a) \in \mathcal{C} \times [k]$. The map ψ is called **feature map**. The subspace Ψ spanned by the **feature vectors** $\{\psi(c, a)\}_{c,a}$ in $\mathbb{R}^d$ is called the **feature space**.

Note: To understand the concept of feature maps, let's consider an example of a website selling books. The context in this scenario is the website visitor, the actions are the recommended books, and the reward is the revenue generated from the sale of a book. In order to make a recommendation to the visitor, we need to understand their interests and preferences, as well as the domain and topic of the books. One way to represent this information is through feature maps. Feature maps are a way to transform the input data (in this case, information about the visitor and the books) into a set of features that can be used to make predictions. For example, we could use indicator variables to represent the visitor's interests and preferences, as well as the domain and topic of the books. These indicator variables would be combined to create a feature map that captures the relevant information for making book recommendations. [1]

5.3 Contextual Bandits with Linear Payoff Functions

Contextual bandits with linear payoff functions refer to a class of reinforcement learning problems where an agent must make decisions based on contextual information to maximize some linear reward function. In this setting, the agent receives a set of features that describe the current state of the environment and must select an action from a set of available actions. The goal of the agent is to learn a policy that maps the contextual features to actions that maximize the expected reward.

The challenge in contextual bandits with linear payoff functions is to balance the exploration of the environment to learn about the true reward function and the exploitation of the current knowledge to maximize immediate rewards. To address this challenge, various algorithms have been developed, including the contextual linear bandit (**CLB**) algorithm and the **LinUCB** algorithm. Both algorithms use a linear regression model to estimate the expected payoff for each action based on contextual features, and they differ in the way they incorporate exploration into the decision-making process.

5.3.1 The LinUCB Algorithm

In round t, the learner is given the decision set $\mathcal{A}_t \subset \mathbb{R}^d$, from which it chooses an action $A_t \in \mathcal{A}_t$ and receives reward $X_t = \langle \theta_*, A_t \rangle$ where η_t is 1-subgaussian given $\mathcal{A}_1, A_1, X_1, \dots, \mathcal{A}_{t-1}, A_{t-1}, X_{t-1}, \mathcal{A}_t$ and A_t . The random pseudo-regret and regret are defined by: $\hat{R}_n = \sum_{t=1}^{n} \max_{a \in \mathcal{A}_t} \langle \theta_*, a - A_t \rangle$ and $R_n = \mathbb{E}[\hat{R}_n] = \mathbb{E}\left[\sum_{t=1}^{n} \max_{a \in \mathcal{A}_t} \langle \theta_*, a \rangle - \sum_{t=1}^{n} X_t\right]$.

The first step is to construct a confidence set $\mathcal{C}_t \subset \mathbb{R}^d$ based on $(A_1, X_1, \dots, A_{t-1}, X_{t-1})$ that contains the unknown parameter vector θ_* with high probability. For any given action $a \in \mathbb{R}^d$, let $UCB_t(a) = \max_{\theta \in \mathcal{C}_t} \langle \theta, a \rangle$ be an upper bound on the mean pay-off $\langle \theta_*, a \rangle$ of a. The **UCB** algorithm that uses the confidence set $\mathcal{C}_t$ at time t then selects:

$$A_t = arg \max_{a \in \mathcal{A}_t} UCB_t(a)$$

UCB applied to linear bandits is known by various names, such as **LinRel** (**lin**ear **re**inforcement **l**earning), **LinUCB** and OFUL (**o**ptimism in the **f**ace of **u**ncertainty for **l**inear bandits).

To apply the concept of **UCB**, we require a method to estimate the unknown quantity, represented by θ_*. There are various approaches one could take to obtain such an estimate. Currently, we are utilizing the **regularized least-squares estimator**, which is:

$$\hat{\theta}_t = arg \min_{\theta \in \mathbb{R}^d} \left(\sum_{s=1}^{t} (X_s - \langle \theta, A_s \rangle)^2 + \lambda \|\theta\|_2^2 \right)$$

where $\lambda \geq 0$ is called the **penalty factor**. The solution to this equation is obtained easily by differentiation and is:

$$\hat{\theta}_t = V_t^{-1} \sum_{s=1}^{t} A_s X_s$$

where $(V_t)_t$ are $d \times d$ matrices given by:

$$V_0 = \lambda I \text{ and } \mathrm{V_t} = \mathrm{V_0} + \sum_{\mathrm{s=1}}^{\mathrm{t}} \mathrm{A_s A_s^T}$$

So $\hat{\theta}_t$ is an estimate of θ_*, which makes it natural to choose $\mathcal{C}_t$ to be centered at $\hat{\theta}_{t-1}$. Thus, the confidence set $\mathcal{C}_t$ satisfies:

$$\mathcal{C}_t \subseteq \mathcal{E}_t = \left\{\theta \in \mathbb{R}^d : \left\|\theta - \hat{\theta}_{t-1}\right\|_{V_{t-1}}^2 \leq \beta_t\right\} \quad (1)$$

where $(\beta_t)_t$ is an increasing sequence of constants with $\beta_1 \geq 1$. The set $\mathcal{E}_t$ is an ellipsoid centered at $\hat{\theta}_{t-1}$ and with principle axis being the eigenvectors of V_t with corresponding lengths being the reciprocal of the eigenvalues. Notice that as t grows, the matrix V_t has increasing eigenvalues, which means the volume of the ellipse is also shrinking.

Computation:

The computation of A_t can also be written as:

$$\left(A_t, \tilde{\theta}_t\right) = arg \max_{(a,\theta) \in \mathcal{A}_t \times \mathcal{C}_t} \langle \theta, a \rangle \quad (2)$$

This is a bilinear optimization problem over the set $\mathcal{A}_t \times \mathcal{C}_t$. In general, not much can be said about the computational efficiency of solving this problem. There are two notable special cases, however.

(a) Suppose that $a(\theta) = arg \max_{a \in \mathcal{A}_t} \langle \theta, a \rangle$ can be computed efficiently for any θ and that $\mathcal{C}_t = \mathrm{co}(\varphi_1, \ldots, \varphi_m)$ is the convex hull of a finite set. Then A_t can be computed by finding $a(\varphi_1), \ldots, a(\varphi_m)$ and choosing $A_t = a(\varphi_i)$, where i maximizes $\langle \varphi_i, a(\varphi_i) \rangle$.

(b) Assume that $\mathcal{C}_t = \mathcal{E}_t$ is the ellipsoid given in Eq. (1) and $\mathcal{A}_t$ is a small finite set. Then the action A_t from Eq. (2) can be found using:

$$A_t = arg \max_{a \in \mathcal{A}_t} \langle \hat{\theta}_{t-1}, a \rangle + \sqrt{\beta_t} \|a\|_{V_{t-1}^{-1}}$$

which may be solved by simply iterating over the arms and calculating the term inside the $argmax$. The term $\langle \hat{\theta}_{t-1}, a \rangle$ may be interpreted as an empirical estimate of the reward from choosing action a, and $\sqrt{\beta_t}\|a\|_{V_{t-1}^{-1}}$ is a bonus term that ensures sufficient exploration.

Let $\beta_t(\delta) = \lambda + \sqrt{2\log\left(1/\delta\right) + d\log\left(1 + \frac{t}{\lambda d}\right)}$ [2]. The confidence ellipsoid is defined as:

$$C_t(\delta) = \left\{\theta \in \mathbb{R}^d : \left\|\theta - \hat{\theta}_t\right\|_{V_{t-1}} \leq \beta_{t-1}(\delta)\right\}$$

With this choice of confidence ellipsoid the previous optimization problem is equivalent to maximizing:

$$A_t = arg\max_{a \in \mathcal{A}_t}\left(a^{\intercal}\hat{\theta}_t + \beta_{t-1}(\delta)\|a\|_{V_{t-1}^{-1}}\right)$$

The **LinUCB** algorithm is given below:

Step 1: Input probability δ, dimension d, regularization λ

Step 2: $b = 0_{\mathbb{R}^d}, V = \lambda\mathbb{I}_d, \hat{\theta} = 0_{\mathbb{R}^d}$

Step 3: For $t \geq 1$ do

a) Receive $\mathcal{A}_t$

b) Compute $\beta_{t-1}(\delta) = \lambda + \sqrt{2\log\left(1/\delta\right) + d\log\left(1 + \frac{t-1}{\lambda d}\right)}$

c) For $a \in \mathcal{A}_t$ do

i) Compute $UCB(a) = a^{\intercal}\hat{\theta} + \beta_{t-1}\sqrt{a^{\intercal}V^{-1}a}$

ii) $A_t = arg\,max_a(UCB(a))$

iii) Play action A_t and receive reward X_t

iv) Update phase: $V = V + A_t{A_t}^{\intercal}, b = b + X_tA_t, \hat{\theta} = V^{-1}b$

5.3.2 Regret Analysis

Theorem 11:

Under the assumptions of the algorithm, with probability $1 - \delta$ the regret of **LinUCB** satisfies:

$$R_T \leq \sqrt{dT}\sqrt{8\beta_T(\delta)\log\left(1+\frac{TL^2}{\lambda d}\right)}$$

Assumption: The following hold:

a) $1 \leq \beta_1 \leq \beta_2 \leq \cdots \beta_n$

b) $\max_{t\in[n]} sup_{a,b\in\mathcal{A}_t}\langle\theta_*, a-b\rangle \leq 1$

c) $\|a\|_2 \leq L$ for all $a \in \bigcup_{t=1}^{n}\mathcal{A}_t$

d) There exists a $\delta \in (0,1)$ such that with probability $1-\delta$, for all $t \in [n]$, $\theta_* \in \mathcal{C}_t$ where $\mathcal{C}_t$ satisfies Eq. (1).

Theorem 12:

Under the conditions of the assumption with probability $1-\delta$, the regret of **LinUCB** satisfies:

$$\hat{R}_n \leq \sqrt{8n\beta_n\log\left(\frac{detV_n}{detV_0}\right)} \leq \sqrt{8dn\beta_n\log\left(\frac{d\lambda+nL^2}{d\lambda}\right)}$$

Note: The bound given in previous theorem is essentially a worst-case style of bound, with little dependence on the parameter θ_* or the geometry of the action set. In the worst-case, the upper bound is tight up to logarithmic factors. [4]

Corollary:

Under the conditions of the assumption, the expected regret of **LinUCB** with $\delta = {}^{1}/_{n}$ satisfies:

$$R_n \leq Cd\sqrt{n}\log(nL)$$

where $C > 0$ is a suitably large universal constant.

Theorem 13:

Let $\delta \in (0,1)$. Then, with probability at least $1-\delta$, it holds that for all $t \in \mathbb{N}$,

$$\left\|\hat{\theta}_t - \theta_*\right\|_{V_t(\lambda)} < \sqrt{\lambda}\|\theta_*\|_2 + \sqrt{2\log\left(\frac{1}{\delta}\right) + \log\left(\frac{detV_t(\lambda)}{\lambda^d}\right)}$$

Furthermore, if $\|\theta_*\|_2 \leq m_2$, then $\mathbb{P}(\text{exists } t \in \mathbb{N}^+ : \theta_* \notin \mathcal{C}_t) \leq \delta$ with,

$$\mathcal{C}_t = \left\{\theta \in \mathbb{R}^d : \left\|\hat{\theta}_{t-1} - \theta\right\|_{V_{t-1}(\lambda)} < m_2\sqrt{\lambda} + \sqrt{2\log\left(\frac{1}{\delta}\right) + \log\left(\frac{detV_{t-1}(\lambda)}{\lambda^d}\right)}\right\}$$

Note: The choice of β_n may be: $\sqrt{\beta_n} = m_2\sqrt{\lambda} + \sqrt{2\log\left(\frac{1}{\delta}\right) + d\log\left(\frac{d\lambda+nL^2}{d\lambda}\right)}$. Empirically, the choice of β_n in the theorem is never worse than the upper value, and sometimes better, typically by a modest amount. [8]

5.3.3 Lower Bound

Theorem 14 (Hypercube):

Let $\mathcal{A} = [-1,1]^d$ and $\Theta = \left\{-n^{-1/2}, n^{-1/2}\right\}^d$. Then, for any policy, there exists a vector $\theta \in \Theta$ such that:

$$R_n(\mathcal{A},\theta) \geq \frac{\exp(-2)}{8} d\sqrt{n}$$

Note: Except for logarithmic factors, this shows that the **LinUCB** algorithm is near optimal for this action set. The same works when $\mathcal{A} = \{-1,1\}^d$ is restricted to the corners of the hypercube, which is a finite-armed linear bandit.

Theorem 15 (Unit Ball):

Assume $d \leq 2n$ and let $\mathcal{A} = \{x \in \mathbb{R}^d : \|x\|_2 \leq 1\}$. Then there exists a parameter vector $\theta \in \mathbb{R}^d$ with $\|\theta\|_2^2 = d^2/(48n)$ such that:

$$R_n(\mathcal{A},\theta) \geq d\sqrt{n}/\left(16\sqrt{3}\right)$$

Note: When the action set is the unit ball, determining the lower bound for minimax regret poses a greater challenge compared to the hypercube. Unlike the hypercube, where the actions taken in one dimension do not limit choices in other dimensions due to its product structure, the unit ball operates differently. In the unit ball, actions taken in one dimension impose constraints on choices in other dimensions, making the analysis more complex.

6 Thompson Sampling

Thompson sampling is a decision-making algorithm that works by choosing a prior distribution over a set of possible scenarios, or "bandit environments". In each round, the algorithm samples an environment from the posterior distribution and selects the optimal action for that environment. Initially, Thompson only applied this approach to simple scenarios, such as Bernoulli bandits with two arms, and relied on hand calculations to demonstrate its effectiveness. However, recent advances have expanded the approach to a wider range of scenarios and theoretical guarantees now demonstrate that the algorithm is often near-optimal. [15]

The key to Thompson sampling is the randomization it introduces through sampling from the posterior distribution. If the posterior distribution is not well-concentrated, the algorithm is more likely to explore different options. As more data is collected, the posterior distribution becomes more concentrated, and exploration is reduced. Overall, Thompson sampling is a practical and effective algorithm for decision-making in uncertain environments. The algorithm's exploration rate is adjusted dynamically through the posterior distribution, and it can be used in a wide range of scenarios, making it an important tool for many real-world applications.

Example (Bernoulli-Bandit Thompson Sampling):

The Bernoulli-Bandit T.S. problem is a multi-armed bandit problem where there are a total of k actions. Each action, when played, yields either a success (which gives reward 1) or a failure (which gives reward 0) with certain probability, i.e. the reward of each arm follows an independent Bernoulli distribution. The success probabilities for all arms are unknown to the

agent, but are fixed over time. The goal is to maximize the cumulative number of successes over T periods, with $T > k$. We define: $N_i(t)$ (denotes the number of pulls of arm i up to time $t-1$), $S_i(t)$ (denotes the number of successful pulls of arm i up to time $t-1$), $F_i(t)$ (denotes the number of failed pulls of arm i up to time $t-1$), $i(t)$ (denotes the arm played at t) and $r_i(t)$ (denotes the reward of arm i at time t). So we always have $N_i(t) = S_i(t) + F_i(t)$. The Bernoulli-Bandit Thompson Sampling algorithm is given below:

Step 1: (Initialization) for each arm $i \in [k]$, set $S_i = 0, F_i = 0$

Step 2: For $t = 1, \dots, T$ do

a) for each arm i do

sample $\theta_i(t) \sim Beta(S_i + 1, F_i + 1)$

end

b) Play arm $i(t) := arg\max_j \theta_j(t)$

c) Observe reward $r_i(t)$

d) If $r_i(t) = 1$ then

$S_i(t) = S_i(t) + 1$

else

$F_i(t) = F_i(t) + 1$

end

Step 3: End for

The reason to use Beta distribution for Bernoulli rewards is that the beta distribution is a conjugate prior for the Bernoulli distribution: if the prior is a $Beta(a, \beta)$ distribution, then after observing a Bernoulli trial, the posterior distribution is $Beta(a+1, \beta)$ if the trial is a success or $Beta(a, \beta+1)$ if the trial is a failure. The reason that we have 1 added to both parameters of Beta distribution is that when $S_i(t) = F_i(t) = 0$, the distribution $Beta(1,1)$ is uniform. It's natural to have a uniform prior. [11]

6.1 T.S. for Contextual Bandits with Linear Payoffs

Problem Settings:

There are N arms. At time $t = 1,2, \dots$, a context vector $b_i(t) \in \mathbb{R}^d$, is revealed for every arm i. These context vectors are chosen by an adversary in an adaptive manner after observing the arms played and their rewards up to time $t-1$, i.e. $\mathcal{H}_{t-1}$,

$$\mathcal{H}_{t-1} = \{a(\tau), r_{a(\tau)}(\tau), b_i(\tau), i = 1, \dots, N, \tau = 1, \dots, t-1\}$$

where $a(\tau)$ denotes the arm played at time τ. Given $b_i(t)$, the reward for arm i at time t is generated from an unknown distribution with mean $b_i(t)^\intercal \mu$, where $\mu \in \mathbb{R}^d$ is a fixed but unknown parameter.

$$\mathbb{E}\left[r_i(t)\middle|\{b_i(t)\}_{t=1}^N, \mathcal{H}_{t-1}\right] = \mathbb{E}[r_i(t)|b_i(t)] = b_i(t)^\intercal \mu$$

An algorithm for the contextual bandit problem needs to choose, at every time t, an arm $a(t)$ to play, using history $\mathcal{H}_{t-1}$ and current contexts $b_i(t), i = 1, \dots, N$. Let $a^*(t)$ denote the optimal arm at time t, i.e. $a^*(t) = arg\max_i b_i(t)^\intercal \mu$. And let $\Delta_i(t)$ be the difference between the mean rewards of the optimal arm and of arm i at time t, i.e.

$$\Delta_i(t) = b_{a^*(t)}(t)^\intercal \mu - b_i(t)^\intercal \mu$$

Then the regret at time t is defined as:

$$\text{regret}(t) = \Delta_{a(t)}(t)$$

The objective is to minimize the total regret $\mathcal{R}(T) = \sum_{t=1}^T \text{regret}(t)$ in time T. The time horizon T is finite but possibly unknown. We assume that $\eta_{i,t} = r_i(t) - b_i(t)^\intercal \mu$ is conditionally R-subgaussian for a constant $R \geq 0$. We also assume that $\|b_i(t)\|_2 \leq 1, \|\mu\|_2 \leq 1 \; and \; \Delta_i(t) \leq 1 \text{for all } i, t.$

Thompson Sampling Algorithm:

We use Gaussian likelihood function and Gaussian prior to design a version of **Thompson Sampling** algorithm. More precisely, suppose that the likelihood of reward $r_i(t)$ at time t, given context $b_i(t)$ and parameter μ, were given by the pdf of Gaussian distribution $\mathcal{N}(b_i(t)^\intercal \mu, v^2)$. Here, $v = \mathcal{R}\sqrt{\frac{24}{\varepsilon} d ln\left(\frac{1}{\delta}\right)}$, with $\varepsilon \in (0,1)$ which parameterizes this algorithm. Let

$$B(t) = I_d + \sum_{\tau=1}^{t-1} b_{a(\tau)}(\tau)\, b_{a(\tau)}(\tau)^\intercal$$

$$\hat{\mu}(t) = B(t)^{-1}\left(\sum_{\tau=1}^{t-1} b_{a(\tau)}(\tau)\, r_{a(\tau)}(\tau)\right)$$

Then, if the prior for μ at time t is given by $\mathcal{N}(\hat{\mu}(t), v^2 B(t)^{-1})$, it is easy to compute the posterior distribution at time $t+1$,

$$\Pr(\tilde{\mu}|r_i(t)) \propto \Pr(r_i(t)|\tilde{\mu})\Pr(\tilde{\mu})$$

$$\propto exp\left\{-\frac{1}{2v^2}\left(r_i(t)-\tilde{\mu}^\intercal b_i(t)\right)^2+\left(\tilde{\mu}-\hat{\mu}(t)\right)^\intercal B(t)\left(\tilde{\mu}-\hat{\mu}(t)\right)\right\}$$

$$\propto exp\left\{-\frac{1}{2v^2}\left(r_i(t)^2+\tilde{\mu}^\intercal b_i(t)b_i(t)^\intercal\tilde{\mu}+\tilde{\mu}^\intercal B(t)\tilde{\mu}-2\tilde{\mu}^\intercal b_i(t)r_i(t)\right.\right.$$

$$\left.\left.-2\tilde{\mu}^\intercal B(t)\hat{\mu}(t)\right)\right\} \propto exp\left\{-\frac{1}{2v^2}\left(\tilde{\mu}^\intercal B(t+1)\tilde{\mu}-2\tilde{\mu}^\intercal B(t+1)\hat{\mu}(t+1)\right)\right\}$$

$$\propto exp\left\{-\frac{1}{2v^2}\left(\tilde{\mu}-\hat{\mu}(t+1)\right)^\intercal B(t+1)\left(\tilde{\mu}-\hat{\mu}(t+1)\right)\right\}$$

as $\mathcal{N}(\hat{\mu}(t+1), v^2B(t+1)^{-1})$. At every time step t, we will generate a sample $\tilde{\mu}(t)$ from the distribution $\mathcal{N}(\hat{\mu}(t), v^2B(t)^{-1})$ and play the arm i that maximizes $b_i(t)^\intercal\tilde{\mu}(t)$. The **Thompson Sampling** algorithm for contextual bandits is given below:

Step 1: Set $B = I_d, \hat{\mu} = 0_d, f = 0_d$

Step 2: For all $t = 1,2, \ldots,$ do

a) Sample $\tilde{\mu}(t)$ from distribution $\mathcal{N}(\hat{\mu}, v^2B^{-1})$

b) Play arm $a(t) \coloneqq arg\max_i\ b_i(t)^\intercal\tilde{\mu}(t)$ and observe reward r_t

c) Update $B = B + b_{a(t)}(t)b_{a(t)}(t)^\intercal, f = f + b_{a(t)}(t)r_t,\ \hat{\mu} = B^{-1}f$

Step 3: End for

Every step t of the algorithm consists of generating a d-dimensional sample $\tilde{\mu}(t)$ from a multivariate Gaussian distribution, and solving the problem $arg\max_i b_i(t)^\intercal\tilde{\mu}(t)$. Therefore, even if the number of arms N is large (or infinite), the above algorithm is efficient as long as the problem $arg\max_i b_i(t)^\intercal\tilde{\mu}(t)$ is efficiently solvable. [3]

6.2 Regret Analysis

Theorem 16:

The total regret in time T for **Thompson Sampling** for stochastic contextual bandit problem with linear payoff function, with probability $1-\delta$ satisfies:

$$\mathcal{R}(T) \leq C\frac{d^2}{\varepsilon}\sqrt{T^{1+\varepsilon}}\left(\ln(\mathrm{Td})\ln\frac{1}{\delta}\right), \text{for all } \varepsilon \in (0,1), \delta \in (0,1)$$

where $C > 0$ is a universal constant and ε is a parameter used by **Thompson Sampling** algorithm. [12]

Remark: 1) The parameter ε can be chosen to be any constant in $(0,1)$. If T is known, one could choose $\varepsilon = \frac{1}{\ln T}$, to get: $\mathcal{R}(T) \leq Cd^2\sqrt{T}$, where $C > 0$ a universal constant.

2) Our regret bound in theorem does not depend on N, and is applicable to the case of infinite arms.

Now we state two additional results. The first one is for the setting where each of the N arms is associated with a different d-dimensional parameter $\mu_i \in \mathbb{R}^d$, so that the mean reward for arm i at time t is $b_i(t)^\intercal \mu_i$. For this setting **Thompson Sampling** will maintain a separate posterior distribution for each arm i and will only update them whenever i is chosen to be played. At every time step t, instead of a single sample $\tilde{\mu}(t)$, N independent samples will have to be generated: $\tilde{\mu}_i(t)$ for each arm i. We appropriately modify some of the previous definitions:

$$B_i(t) = I_d + \sum_{\tau=1:a(\tau)=i}^{t-1} b_i(\tau)\, b_i(\tau)^\intercal$$

$$\hat{\mu}_i(t) = B_i(t)^{-1}\left(\sum_{\tau=1:a(\tau)=i}^{t-1} b_i(\tau)\, r_i(\tau)\right)$$

The posterior distribution for each arm i at time t will be $\mathcal{N}(b_i(t)^\intercal \hat{\mu}_i(t), v^2 b_i(t)^\intercal B_i(t)^{-1} b_i(t))$. And, the **Thompson Sampling** algorithm is now stated as follows:

Step 1: Set $B_i = I_d, \hat{\mu}_i = 0_d, f_i = 0_d, i = 1, \ldots, N$

Step 2: For all $t = 1,2, \ldots$, do

a) For each arm $i = 1, \ldots, N$, sample $\theta_i(t)$ independently from the posterior distribution

b) Play arm $a(t) \coloneqq arg\max_i \theta_i(t)$ and observe reward r_t

c) Update $B_{a(t)} = B_{a(t)} + b_{a(t)}(t) b_{a(t)}(t)^\intercal$, $f_{a(t)} = f_{a(t)} + b_{a(t)}(t) r_t$, $\hat{\mu}_{a(t)} = {B_{a(t)}}^{-1} f_{a(t)}$

Step 3: End for

The optimal arm $a^*(t)$ is now the arm that maximizes $b_i(t)^\intercal \mu_i$, and the regret at time t is defined as:

$$\text{regret}(t) = b_{a^*(t)}(t)^\intercal \mu_{a^*(t)} - b_{a(t)}(t)^\intercal \mu_{a(t)}$$

Theorem 17:

For this setting, with probability $1-\delta$, the total regret in time T for **Thompson Sampling** satisfies:

$$\mathcal{R}(T) \leq Cd\sqrt{\frac{NT^{1+\varepsilon}\ln N}{\varepsilon}}\left(\ln T \ln\frac{1}{\delta}\right), \text{for all } \varepsilon \in (0,1), \delta \in (0,1)$$

where $C > 0$ a is a universal constant.

Note: Unlike first theorem, the second one has dependence on N in its regret bound, which is reasonable since this theorem deals with a setting where there are N different parameters to learn. However, the bound in last theorem has a better dependence on d. This improvement results from the independence of $\theta_i(t) = b_i(t)^\intercal \tilde{\mu}_i(t)$ in the algorithm for this setting. On the other hand in the algorithm, a single $\tilde{\mu}(t)$ is generated, and thus $\theta_i(t) = b_i(t)^\intercal \tilde{\mu}_i(t)$ are not independent.

Considering the note, we modify the algorithm for the single parameter setting by entailing the $\theta_i(t)$'s to be independently generated, each with marginal distribution $b_i(t)^\intercal \tilde{\mu}_i(t)$. The arm with the highest value of $\theta_i(t)$ is played at time t. Although, this modified algorithm could be inefficient compared to the first algorithm if N is large compared to d, the better dependence on d in regret bounds could be useful if d is large. The modified **Thompson Sampling** algorithm is given below:

Step 1: Set $B = I_d, \hat{\mu} = 0_d, f = 0_d$

Step 2: For all $t = 1,2, \ldots$, do

a) For each arm $i = 1, \ldots, N$, sample $\theta_i(t)$ from distribution $\mathcal{N}(b_i(t)^\intercal \hat{\mu}, v^2 b_i(t)^\intercal B^{-1} b_i(t))$

b) Play arm $a(t) \coloneqq arg\max_i \theta_i(t)$ and observe reward r_t

c) Update $B = B + b_{a(t)}(t) b_{a(t)}(t)^\intercal$, $f = f + b_{a(t)}(t) r_t$, $\hat{\mu} = B^{-1} f$

Step 3: End for

Theorem 18:

For the modified algorithm in single parameter setting, with probability $1-\delta$ the total regret in time T satisfies:

$$\mathcal{R}(T) \leq Cd\sqrt{\frac{T^{1+\varepsilon}\ln N}{\varepsilon}}\left(\ln T \ln\frac{1}{\delta}\right), \text{for all } \varepsilon \in (0,1), \delta \in (0,1)$$

where $C > 0$ is a universal constant.

7 Simulation

To make all these more concrete, we give an example of implementing the **Thompson Sampling** algorithm for a Bernoulli bandit problem with two arms in R-code, which is presented in appendix 1. To better understand it, a detailed explanation is given below:

We start by loading the necessary package "MASS", which contains the "rbeta" function used for sampling from the beta distribution. The "simulate_bandit" function takes a success probability "p" and returns a binary outcome (0 or 1) based on a Bernoulli distribution.

The "thompson_sampling" function implements the **Thompson Sampling** algorithm. It takes two arguments: "n_arms" (the number of arms in the bandit) and "n_trials" (the number of trials to run the algorithm). Inside the function, we initialize two variables: "arm_success" and "arm_failures" to keep track of the number of successes and failures for each arm.

For each trial, we sample success probabilities for each arm using the beta distribution with parameters '("arm_success" + 1, "arm_failures" + 1)'. These parameters represent the number of successes and failures observed for each arm plus a pseudo-count of 1 to ensure exploration. The "rbeta" function is used to generate random samples from the beta distribution.

Next, we select the arm with the highest sample by finding the index of the maximum value in "arm_samples". We then simulate a reward for the chosen arm using the "simulate_bandit" function, which takes the true success probabilities "p" and returns a binary reward (0 or 1) based on the Bernoulli distribution.

Finally, we update the arm success/failure counts based on the observed reward. If the reward is 1, we increment the success count for the chosen arm; otherwise, we increment the failure count. After running the **Thompson Sampling** algorithm for the specified number

of trials, we return the estimated success probabilities for each arm by dividing the success count by the total count (successes + failures).

In the main code, we set the number of arms (“n_arms”) to 2 and the number of trials (“n_trials”) to 1000. We also define the true success probabilities of the arms (“p”) as “c (0.3, 0.5)”.

We then run the **Thompson Sampling** algorithm 100 times by calling the “thompson_sampling” function inside the “replicate” function. This allows us to obtain an average estimate of the arm success probabilities across multiple runs.

Finally, we print the estimated success probabilities for each arm by calculating the mean of the results matrix for each arm separately.

Now, we are using the **LinUCB** algorithm in the same example. The R-code is shown in appendix 2. More specifically, in this implementation, we define the “simulate_bandit” function, which is the same as before. The “lin_ucb function” implements the **LinUCB** algorithm. It takes three arguments: “n_arms” (the number of arms in the bandit), “n_trials” (the number of trials to run the algorithm), and “alpha” (the exploration parameter). Inside the function, we initialize variables to keep track of the number of successes, failures, counts, and rewards for each arm. We also initialize the UCB values for each arm to the specified “alpha”. For each trial, we select the arm with the highest UCB value using the “which.max” function.

We then simulate a reward for the chosen arm using the “simulate_bandit” function, which takes the true success probabilities “p” and returns a binary reward (0 or 1) based on the Bernoulli distribution. Next, we update the arm success/failure counts and rewards based on the observed reward. We also increment the arm count.

After that, we update the UCB values for all arms. For arms that have been pulled at least once (“arm_counts [arm]> 0”), we calculate the mean reward and an exploration term based on the exploration parameter “alpha” and the logarithm of the current trial number “t”. The UCB value for an arm is the sum of the mean reward and the exploration term.

After running the **LinUCB** algorithm for the specified number of trials, we return the estimated success probabilities for each arm by dividing the success count by the total count (successes + failures).

In the main code, we set the number of arms ("n_arms") to 2, the number of trials ("n_trials") to 1000, and the exploration parameter ("alpha") to 2.0. We then run the LinUCB algorithm 100 times by calling the "lin_ucb" function inside the "replicate" function. This allows us to obtain an average estimate of the arm success probabilities across multiple runs.

Finally, we print the estimated success probabilities for each arm by calculating the mean of the results matrix for each arm separately.

Appendix

1) *# Required packages*

```
library(MASS)

# Function to simulate Bernoulli bandit
simulate_bandit ← function (p) {
  return (rbinom (1, 1, p))
}

# Thompson Sampling function
thompson_sampling ← function (n_arms, n_trials) {
  # Initialize variables
  arm_success ← numeric (n_arms)
  arm_failures ← numeric (n_arms)

  for (t in 1:n_trials) {
    # Sample success probabilities for each arm
    arm_samples ← rbeta(n_arms, arm_success + 1, arm_failures + 1)

    # Select arm with highest sample
    chosen_arm ← which.max (arm_samples)
```

```
    # Simulate reward for chosen arm
    reward ← simulate_bandit (p [chosen_arm])

    # Update arm success/failure counts
    if (reward == 1) {
      arm_success [chosen_arm] ← arm_success [chosen_arm] + 1
    } else {
      arm_failures [chosen_arm] ← arm_failures [chosen_arm] + 1
    }
  }

  # Return arm success probabilities
  return (arm_success / (arm_success + arm_failures))
}

# Number of arms
n_arms ← 2

# Number of trials
n_trials ← 1000

# Run Thompson Sampling
p ← c (0.3, 0.5) # True success probabilities of arms
results ← replicate (100, thompson_sampling (n_arms, n_trials))

# Print arm success probabilities
for (arm in 1:n_arms) {
  cat (paste ("Arm", arm, "success probability:", mean(results[arm, ]), "\n"))
}
```

2) *# Function to simulate Bernoulli bandit*

```
simulate_bandit ← function (p) {
  return (rbinom (1, 1, p))
}

# LinUCB function
lin_ucb ← function (n_arms, n_trials, alpha) {
```

```
 # Initialize variables
 arm_success ← numeric(n_arms)
 arm_failures ← numeric(n_arms)
 arm_counts ← numeric(n_arms)
 arm_rewards ← numeric(n_arms)
 arm_UCB ← rep(alpha, n_arms)

 for (t in 1:n_trials) {
    # Select arm with highest UCB
    chosen_arm ← which.max(arm_UCB)

    # Simulate reward for chosen arm
    reward ← simulate_bandit (p [chosen_arm])

    # Update arm success/failure counts and rewards
    if (reward == 1) {
      arm_success [chosen_arm] ← arm_success [chosen_arm] + 1
    } else {
      arm_failures [chosen_arm] ← arm_failures [chosen_arm] + 1
    }
    arm_counts [chosen_arm] ← arm_counts [chosen_arm] + 1
    arm_rewards [chosen_arm] ← arm_rewards [chosen_arm] + reward

    # Update UCB values for all arms
    for (arm in 1:n_arms) {
       if (arm_counts [arm] > 0) {
          mean_reward ← arm_rewards [arm] / arm_counts [arm]
          exploration_term ← sqrt (alpha * log (t) / arm_counts [arm])
          arm_UCB [arm] ← mean_reward + exploration_term
       }
    }
 }

# Return arm success probabilities
return (arm_success / (arm_success + arm_failures))
}
```

```
# Number of arms
n_arms ← 2

# Number of trials
n_trials ← 1000

# Exploration parameter
alpha ← 2.0

# Run LinUCB
p ← c (0.3, 0.5) # True success probabilities of arms
results ← replicate (100, lin_ucb (n_arms, n_trials, alpha))

# Print arm success probabilities
for (arm in 1:n_arms) {
   cat (paste ("Arm", arm, "success probability:", mean(results[arm, ]), "\n"))
}
```

References

[1] Tor Lattimore and Casba Szepesvari, “Bandit Algorithms”, Cambridge University Press, 2020.

[2] Yoan Russac. “Introduction to Linear Bandits”, ENS Paris, 2019.

https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=0CAIQw7AJahcKEwiIsNK-87D_AhUAAAAAHQAAAAAQAw&url=https%3A%2F%2Fwww.yoanrussac.com%2Fen%2Ftalk%2Ftalk1-ens%2Fintro_linear_bandits.pdf&psig=AOvVaw2qVagBgWEsm9LUpjBUR3rT&ust=1686218484468248

[3] Shipra Agrawal and Navin Goyal, “Thompson Sampling for Contextual Bandits with Linear Payoffs”, Proceedings of Machine Learning Research (PMLR) 28(3):127-135, 2013.

[4] Wei Chu, Lihong Li and Lev Reyzin and Robert E. Schapire, “Contextual Bandits with Linear Payoff Functions”, PMLR 15:208-214, 2011.

[5] Chi Jin, Zhuoran Yang, Zhaoran Wang and Michael I. Jordan, “Provably Efficient Reinforcement Learning with Linear Function Approximation”, PMLR 125:1-7, 2020.

[6] Alexander Slivkins, “Introduction to Multi-Armed Bandits”, Foundations and Trends in Machine Learning, Vol. 12:No.1-2, pp 1-286.

http://dx.doi.org/10.1561/2200000068

[7] Li, Lihong, et al. “A Contextual Bandit Approach to Personalized News Article Recommendation”, Proceedings of the 19th International Conference on World Wide Web, ACM, 2010.

[8] Yasin Abbasi-Yadkori, David Pal and Casba Szepesvari, “Improved Algorithms for Linear Stochastic Bandits”, Advances in Neural Information Processing Systems 24, 2011.

[9] Kevin Jamieson, “Linear bandits”, February 2018.
https://courses.cs.washington.edu/courses/cse599i/18wi/resources/lecture9/lecture9pdf

[10] Alekh Agarwal, “Exploration in Contextual Bandits”, October 2017.
http://alekhagarwal.net/bandits_and_rl/exploration.pdf

[11] Shipra Agrawal and Navin Goyal, “Analysis of Thompson Sampling for the multi-armed bandit problem”. JMLR Workshop and Conference Proceedings 23:39.1-39.26, 2012.

[12] Shipra Agrawal and Navin Goyal, “Further optimal regret bounds for Thompson Sampling”, Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, PMLR 31:99-107, 2013.

[13] Aditya Gopalan, Shie Mannor, and Yishay Mansour, “Thompson sampling for complex online problems”, PMLR, 2014.

[14] Branislav Kveton, Csaba Szepesvari, Sharan Vaswani, Zheng Wen, Tor Lattimore, and Mohammad Ghavamzadeh, “Garbage in, reward out: Bootstrapping exploration in multi-armed bandits”, PMLR, 2019.

[15] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen, “A tutorial on Thompson Sampling”, 2017.