POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Stochastic Linear Bandit with Global-Local Structure

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Francesco Fulco Gonzales**

Student ID: 968843
Advisor: Prof. Francesco Trovò
Co-advisors: Marco Mussi, Gianmarco Genalti, Prof. Marcello Restelli
Academic Year: 2021-22

Abstract

This work pertains to the field of Multi-Armed-Bandits (MAB), a framework in online learning where an agent sequentially chooses from a set of available actions, called arms, and receives feedback in the form of a scalar reward for each choice, with the goal of maximizing the cumulative reward over time. A well-known extension of the MAB is the stochastic linear bandit, where each arm is a vector and the expected reward of each arm is the dot product between the arm vector and an unknown parameter vector, which the agent aims to estimate through repeated pulls of the arms. We analyze a specific multi-agent bandit setting, where multiple linear bandits face different yet related tasks, with the additional assumption that the parameter vector of each agent is decomposed into a global component shared across all agents and a local component specific to each agent. We call the proposed problem scenario *Partitioned Linear Setting*. At each time step, an external context index determines which agent will be the actor of the current turn, and the selected agent chooses an arm from a set of available arms. To solve this setting we propose P-LinUCB, an algorithm that effectively maximizes the collective reward of multiple agents by leveraging both global and local information. We provide an experimental analysis of the algorithm's performance and show that it outperforms a well-known baseline for the multi-agent bandit setting across multiple scenarios.

Keywords: Multi-Armed Bandit, Linear Bandit, Multi-task Bandit, Clustering of Bandits

Abstract in lingua italiana

L'argomento di questo lavoro tratta il campo dei Multi-Armed-Bandits (MAB), un framework per l'apprendimento online in cui un agente sceglie ripetutamente un'azione da un'insieme di azioni disponibili, e riceve un feedback per ogni scelta, con l'obiettivo di massimizzare il reward cumulativo nel tempo. Una nota estensione del MAB è lo stochastic linear bandit, in cui ogni azione è un vettore e il valore atteso di ciascuna azione è il prodotto scalare tra il vettore dell'azione e un vettore di parametri sconosciuti, che l'agente stima scegliendo ripetutamente delle azioni. La tesi affronta una variante del MAB chiamata *Partitioned Linear Setting*, in cui abbiamo più linear bandit che affrontano compiti diversi ma correlati. Il vettore dei parametri di ciascun agente viene scomposto in una componente globale condivisa tra tutti gli agenti e una componente locale specifica per ciascun agente. Ad ogni istante di tempo, un indice di contesto esterno determina quale agente sarà designato per il turno corrente, e l'agente selezionato sceglie un'azione dall'insieme di azione disponibile. Per risolvere il problema descritto proponiamo P-LinUCB, un algoritmo che massimizza il reward collettivo di più agenti sfruttando sia le informazioni globali che quelle locali. Forniamo un'analisi sperimentale delle performance dell'algoritmo e dimostriamo che batte in vari scenari una baseline comunemente usata per il problema dei bandit multi-agente.

Parole chiave: Multi-Armed Bandit, Linear Bandit, Multi-task Bandit, Clustering di Bandit

Contents

1 | Introduction

The study of sequential decision-making under uncertainty is a fundamental area of research within the field of artificial intelligence. Its paramount importance lies in its applicability to a wide range of practical scenarios, making it a critical topic of investigation. The ability to make optimal decisions when faced with uncertain outcomes is fundamental to many real-world situations, ranging from mundane activities, such as the choice of which movie to watch or how to commute to work, to complex problems, such as financial investment or web content optimization.

An even more complex scenario is when the decision-maker is faced with multiple different yet related tasks. In this setting, a naive approach might prescribe solving each task independently using the same strategies that we would adopt when faced with a single task. However, knowledge is expensive, and it often takes a lot of trial and error to find what is the best choice to make in a certain scenario, and restarting the whole process for each new task would lead to an awful waste of time and resources. We know that there are indeed similarities between the tasks (our hard-earned knowledge), and we somehow want to apply as much as we learned from previous similar tasks that we solved. This is the fundamental principle that guides our research: leveraging the information that we learned from a decision-making setting to guide others. More specifically, we assume that all these tasks share some common structure but also have task-specific information. We propose an algorithmic solution to this problem that consistently outperforms the naive solution in a variety of different scenarios.

1.1. Applications

Multi-armed bandit problems have diverse applications across various domains [29]. The initial use case involved designing ethical medical trials with the objective of obtaining valuable scientific insights while minimizing harm to patients [32]. More recent applications include optimizing web-related tasks such as website design [35], content selection, and search results, as well as recommender systems [19, 28] for movies, songs, restaurants, and hotels. Additionally, these problems are relevant in economics for pricing [22, 23, 33]

Application	Action	Reward
Medical trials	Prescription drug	Health outcome
Web design	Page layout	#clicks
Content optimization	Emphasized items/articles	#clicks
Web search	Search results for a given query	#satisfied users
Advertisement	Displayed ad	Revenue from ad
Recommender systems	Recommended item (e.g., movie)	Clicks on item or not
E-commerce pricing	Price	Revenue
Procurement	Items to purchase and prices	#items procured
Auction design	Reserve price	Revenue
Crowdsourcing	Match task w/ worker & price	#completed tasks
Datacenter design	Server to route the job to	Job completion time
Internet	TCP settings	Connection quality
Radio networks	Radio frequency	#successful transmissions
Robot control	Strategy for a given task	Job completion time

Table 1.1: Applications of multi-armed bandits.

and auction optimization [24], as well as for improving task assignment and pricing on crowdsourcing platforms [26]. In computer systems, exploration can lead to better optimization of data centers and networking protocols rather than relying on a fixed design. Lastly, these problems can aid in teaching robots to perform tasks more efficiently. Table 1.1 reports some of the most common applications and the mapping of the actions and rewards in their context.

1.1.1. Motivating Examples

While in the bandit literature, it is more popular the scenario involving a single decision-maker solving a solitary bandit instance, there are many practical settings that contemplate a multitude of simultaneous bandit instances for closely related learning tasks. This presents an opportunity to learn not only within each individual bandit instance but also across comparable instances. We give a motivating example that illustrates a potential real-life instance of our problem setting.

Large retailers require accurate predictions of store-specific demand for their products to make informed decisions regarding pricing and inventory management [22, 23]. To achieve this, it is crucial to learn the demand model from sales data collected at the target store, which takes into account specific idiosyncrasies that are unique to the store and its customer population. These idiosyncrasies can include differences in customer preferences and trends, product placement within the store, and promotional decisions. As each store faces a distinct bandit learning problem, it is important to transfer knowledge across similar bandit instances where stores serve similar customer populations, which creates an opportunity to improve performance.

Recommender systems are yet another prominent application of multi-armed bandits since they allow to optimization of the recommendation process by dynamically exploring and exploiting different options. In the context of recommender systems, a bandit is allocated to each user, and the decisions correspond to the different items that are available for recommendation, e.g., movies, songs, or products. In these settings, to make an informed decision on what items to display to a user, we must take into account both the preferences unique to the user, but also embed the knowledge about global properties that are common to all users, for example, if a historical event happens, the system might want to bias towards movies relevant to the topic.

The reward associated with an arm can be defined as the probability of a user engaging with the recommended item. MAB algorithms aim to learn the underlying reward distribution by iteratively selecting and presenting arms to users, and updating the estimated reward distributions based on the observed feedback

There are various other examples where learning heterogeneous treatment effects across multiple experiments is desirable. These include customer promotion targeting, A/B testing on platforms, and identifying promising combination therapies in clinical trials. It is noteworthy that bandits are most useful in situations where there is limited historical data, for reasons such as the novelty or nonstationarity of the learning problem. In these scenarios, transfer learning from related data sources can be highly valuable in enhancing performance.

1.2. A brief overview of Multi-Armed-Bandits

The Multi-Armed Bandit is the mathematical framework that we use to model the aforementioned real-world phenomena. In the language of bandits, a sequential decision problem is a game played by an agent and an environment, the external entity that sets the rules of the game. The set of choices that the agent is given is called arms.

At each time step, an agent must choose an arm from the available arm set, which results in a reward associated with that arm at that time step. The agent's objective is to accumulate as much reward as possible over a given time horizon by learning how to make optimal decisions using past arms and reward information, referred to as the agent's history. The challenge then becomes how the agent can effectively learn from its history to make optimal decisions in the future. This is addressed by a policy, which is a mapping from history to arms.

The main problem is that the environment is unknown to the agent, that only knows the general reward-generating strategy of the environment but lacks key information on how the next reward is generated. For example, an agent may know that each reward is generated by an arm-specific is Bernoulli distribution but does not know their means, which is the key information that would guide the choice of the next arm. Therefore, not only will it need to choose the best arm by estimating this parameter with currently available data, but also occasionally choose apparently suboptimal arms that could have been simply underestimated due to an unfortunate previous draw, since there is a degree of randomness in the rewards. This example illustrates the exploration-exploitation dilemma in the context of Stochastic Multi-Armed Bandits, characterized by the fact that the rewards are associated with a distribution that generates rewards.

A class of particular interest within the field of Stochastic Multi-Armed Bandits are Stochastic Linear Bandits. In this setting the arms are vectors, and the distributions associated with each arm are Sub-Gaussian (a generalization of the well-known Gaussian distribution) with known variance. The peculiarity of the setting is that the mean of the distribution of each arm is given by the dot product between the arm and an unknown parameter vector. This parameter creates a relationship among the arms, thus giving the Linear Bandit a great advantage: it lets the agent learn the environment without necessarily having to pull every arm since it can gather information on the parameter from any arm pull. For this reason, linear bandits are exceptionally useful when dealing with continuous or very large arm spaces, where the agent couldn't possibly try out all arms.

A natural extension of the previous setting is to consider multiple agents at the same time. In a multi-agent bandit setting, there are multiple agents who need to make decisions. Each agent has its own set of arms to choose from and can only receive feedback on the reward it received from its selected arm. The agents need to balance their individual exploration and exploitation of arms while also considering the arms of other agents in the system. The goal is to maximize the collective reward of all agents. The challenge, in this case, is to determine how the agents can collaborate to improve their performance. Agents may or may not have similar unknown bandit parameters, and finding the best strategy for sharing observations is an open research problem.

1.3. Original Contributions

We tackle the Multi-Agent Stochastic Linear Bandit setting by imposing a structure on the interactions between agents. We consider a novel setting, called *Partitioned Linear Setting*, where multiple agents are involved in a decision-making process where each agent has its own set of arms. The agents receive rewards based on their arms and the value of a parameter vector. The unique feature of this setting is that the parameter vector is decomposed into two components, namely a global component and a local component. The global component is shared across all agents and depends only on the first components of the parameter vector, while the remaining components of the parameter vector are specific to each agent and constitute the local component. The objective of the agents, in typical bandit fashion, is to maximize their collective reward over time by choosing arms that balance the exploration-exploitation trade-off but to do so, they need to collaborate in an effective way by leveraging the structure given by the problem.

We propose a novel algorithm, called Partitioned LinUCB (P-LinUCB), for solving the multi-agent stochastic linear bandit problem by exploiting the structure of the problem. Specifically, the proposed algorithm leverages the assumption that the bandit parameter consists of shared and local components, and uses this information to learn the global component from the most accurate bandit instance and shares it across all bandits when the estimate reaches a high degree of accuracy. In contrast, the local components are learned separately for each instance. Our experimental results demonstrate that this approach outperforms the naive solution of running an independent linear bandit per context. Furthermore, we empirically show that P-LinUCB performs at least as well as the naive solution, with a moderate gain in most cases and a significant gain in common scenarios such as the long-tail distribution of contexts. The main original contribution of this work is the concept of using reliable estimates of the global parameter to speed up the learning process, which provides a new perspective on solving the partitioned linear bandit problem and contributes to the development of more efficient and effective algorithms for solving similar problems.

1.4. Thesis Structure

This thesis work is structured into six chapters, including the current one, each with a distinct focus.

Chapter 2 provides an overview of the mathematical foundations that underlie the research and lays a strong foundation for subsequent chapters. This chapter goes through the definitions of linear regression and its variants and then focuses on the Multi-Armed Bandit problem and the algorithmic approaches that are relevant to the research.

Chapter 3 deals with the classical literature relevant to our approach and recent advancements that are adjacent to our research problem.

In Chapter 4 we formally define the problem setting. We start with a general formulation, the Multi-Agent Stochastic Linear Bandit Setting. The core of this chapter is the reward-generating function that characterizes the Partitioned Linear Bandit problem. We also go through the mathematical formulation of the regret performance measure and the ideas behind it.

Chapter 5 contains the main contribution of the thesis, where we describe in detail the newly developed algorithm that solves the setting outlined in Chapter 4. We provide an intuitive explanation of the algorithm and include the pseudocode, along with a thorough breakdown of each phase of the algorithm.

Chapter 6 presents the experimental analysis of the algorithm, using synthetically generated data, with the goal of demonstrating its strengths and weaknesses in different settings. For each setting, we highlight the goal of the experiment, its main parameters, and the results, which are presented in the form of graphs and tables.

Finally, in Chapter 7, we draw conclusions, summarize the main results of the work, and propose some future developments.

2 | Preliminaries

This chapter provides a brief overview of the fundamental mathematical groundings that form the basis of our research and establishes a strong foundation for the subsequent chapters.

We start by introducing linear regression and its variants, which will serve as a basis for later concepts, and then we introduce formally the problem of Multi-Armed Bandits (MAB), the main topic of this thesis. We explain the more general Reinforcement Learning paradigm, and how the Multi-Armed Bandit problem fits into this setting. We expanded upon the MAB framework by giving an intuition of the Optimism in Face of the Uncertainty principle, which provides a useful and common approach for solving the exploration-exploitation trade-off. We then use these concepts to delve into the topic of linear bandits, which is a specific type of Multi-Armed Bandit problem where the reward function is assumed to be linear with respect to some underlying parameters.

2.1. Linear Regression

Regression analysis aims to predict the value of one or multiple continuous target variables, represented by the vector t, based on an input vector x of d-dimensional variables. Simple linear regression models are linear functions of the input variables, but a more powerful class of functions can be obtained by combining fixed nonlinear functions of the input variables, known as basis functions. Although these models remain linear functions of their parameters, they can achieve nonlinear behavior with respect to input variables. The term *linear* in the model's name refers to its linearity with respect to parameters. Despite being linear in the parameter space, these models have the ability to estimate nonlinear input-output relationships. This feature makes them particularly useful for modeling complex systems with non-trivial interactions among input variables. Furthermore, these models possess simple analytical properties due to their linear parameterization, which facilitates model interpretation and analysis.

A basic linear model for regression involves a linear combination of input variables [7].

If we have a set of M-1 observations represented by x_m, where $m = 1, ..., M-1$, we can express the model as follows:

$$y(x, w) = w_0 + w_1 x_1 + ... + w_{M-1} x_{M-1} = w_0 + \sum_{n=1}^{M-1} (w_n x_n) = \boldsymbol{w}^\top \boldsymbol{x},$$
$$\text{where} \quad \boldsymbol{w}^\top = \begin{bmatrix} w_0 & w_1 & \ldots, & w_{M-1} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_{M-1} \end{bmatrix}. \tag{2.1}$$

Notice that the vector notation contains the dummy sample $x_0 = 1$ in order to include w_0 in $\boldsymbol{w}^\top \boldsymbol{x}$. This model is commonly known as linear regression. The key property of this model is that it is a linear function of the parameters $w_0, ..., w_{M-1}$.

Loss functions

In order to assess the quality of the parameters, it is necessary to define a measure of performance in the given task, which is often accomplished through the use of loss functions.

$L(t, y(\boldsymbol{x}))$	Loss function
$\mathbb{E}[L] = \int\int L(t, y(\boldsymbol{x}))p(\boldsymbol{x}, t) d\boldsymbol{x} dt$	Expected loss function

Our goal is to find the model $y(\boldsymbol{x})$ that minimizes the loss function $L(t, y(\boldsymbol{x}))$. If we take the Minkowsky loss:

$$\mathbb{E}[L] = \int\int (t - y(\boldsymbol{x}))^q p(t, \boldsymbol{x}) dt d\boldsymbol{x}.$$

Based on q the model that minimize $\mathbb{E}[L]$ is:

- $q = 2:$ $y(x) = \mathbb{E}[t|\boldsymbol{x}]$ Conditional mean. Note $\mathbb{E}[t|\boldsymbol{x}] = \int tp(t|\boldsymbol{x})dt$;
- $q = 1:$ $y(\boldsymbol{x}) = median(t|\boldsymbol{x})$ Conditional median;
- $q \to 0:$ $y(\boldsymbol{x}) = mode(t|\boldsymbol{x})$ Conditional mode.

For $q = 2$ we have the squared loss function, which is the most commonly used.

2.1.1. Least square minimization

The least squares method is a widely used technique in regression analysis that involves approximating a model by minimizing the sum of squared residuals. Specifically, given a dataset of N samples, we define the following loss (error) function:

$$L(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(\underbrace{x_n}_{n^{th}input}, w) - \underbrace{t_n}_{n^{th}target})^2.$$

This loss function is called the Squared Sum of Errors (SSE) or half Residual Sum of Squares (RSS) and can be also formulated as:

$$RSS(\boldsymbol{w}) = \|\boldsymbol{\varepsilon}\|_2^2, \quad \text{where} \quad \boldsymbol{\varepsilon} = \begin{bmatrix} y(x_1, w) - t_1 \\ \vdots \\ y(x_N, w) - t_N \end{bmatrix}.$$

The notation above indicates the square of the L_2 norm of $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \dots, & x_M \end{bmatrix}^\top$, which is defined as:

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{M} x_i^2}.$$

Ordinary Least Squares

Ordinary Least Squares (OLS) is a closed-form solution for linear regression that is utilized for estimating the unknown parameters of a linear regression model. This method involves minimizing the sum of the squared differences between the observed dependent variable and the corresponding predicted values, resulting in a solution that is mathematically derived and does not require iterative numerical optimization.

Given N samples and M parameters, we construct $X = \begin{bmatrix} \boldsymbol{x}_1 & \dots, & \boldsymbol{x}_N \end{bmatrix}^\top$ and
$\boldsymbol{t} = \begin{bmatrix} t_1 & \dots, & t_N \end{bmatrix}^\top$.

We can rewrite the SSE in matrix notation

$$L(\boldsymbol{w}) = \frac{1}{2}RSS(\boldsymbol{w}) = \frac{1}{2}(\boldsymbol{t} - X\boldsymbol{w})^\top(\boldsymbol{t} - X\boldsymbol{w})$$

To minimize $L(\boldsymbol{w})$ we have to calculate the first and the second derivative

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial(\frac{1}{2}(\boldsymbol{t} - X\boldsymbol{w})^\top(\boldsymbol{t} - X\boldsymbol{w}))}{\partial \boldsymbol{w}} = -X^\top(\boldsymbol{t} - X\boldsymbol{w})$$

Note that $\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$ are the directions of every w_i to minimize $L(\boldsymbol{w})$.

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} = X^\top X$$

$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top}$ is a semi-definite positive matrix i.e., $x^\top X^\top X x \geq 0, \quad \forall x \in \mathbb{R} \setminus$, so all eigenvalues ≥ 0. It also means that for $\frac{\partial L(\boldsymbol{w})}{\partial w} = 0$ we find the minimum of $L(\boldsymbol{w})$

Now we have to find w imposing the first derivative to zero

$$\begin{aligned}
\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} &= 0 \\
-X^\top(\boldsymbol{t} - X\boldsymbol{w}) &= 0 \\
-X^\top\boldsymbol{t} + X^\top X\boldsymbol{w} &= 0 \\
X^\top X\boldsymbol{w} &= X^\top\boldsymbol{t} \\
\hat{\boldsymbol{w}} &= (X^\top X)^{-1}X^\top\boldsymbol{t}
\end{aligned}$$

Second derivative can give us some information about feature importance. If we have some eigenvalues close to zero we could have the following situations:

- $N < M$, fewer samples than dimensions (parameters);
- The feature with null eigenvalue is linearly dependent from other features.

Underfitting and Overfitting

The success of an algorithm is significantly influenced by the complexity of the model. Model complexity can be defined as the number of parameters and features incorporated in the model. The complexity of the model impacts its ability to generalize to unseen data. An ideal model is one that can effectively generalize and provide reasonable outputs for new input sets. Poor generalization can manifest in two ways:

- **Underfitting** The model is too simple and therefore cannot accurately estimate the true model due to a lack of parameters.
- **Overfitting** The model is too complex and becomes too heavily reliant on the dataset at hand. As a result, the model performs exceptionally well on the given

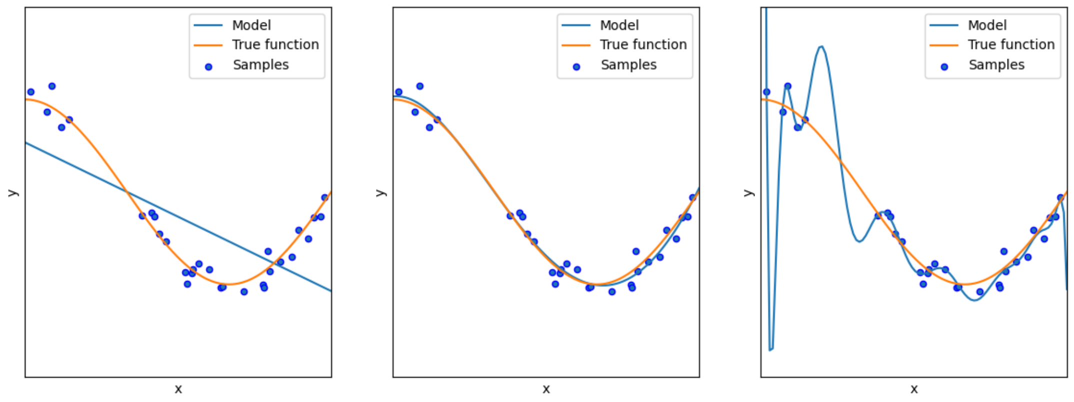


Figure 2.1: Examples of underfitting (left), proper fitting (center), and overfitting (right).

dataset but fails to generalize to other datasets. Overfitting can occur when the model is attempting to interpolate the dataset, learning both the true model and the noise of the initial dataset due to the excess number of parameters.

One simple way to evaluate the performance of a model is to examine its loss function value, with lower values indicating better performance. However, relying solely on the loss function value can lead to overlooking important information related to model complexity. As previously mentioned, overfitting occurs when the model is attempting to interpolate the dataset, which can produce a loss function value that approaches zero, but the model's true performance may still be suboptimal. An example of overfitting and underfitting is provided in Figure 2.1.

To overcome this problem we can split the dataset into two subsets,

- **Training set** This set is used to learn the model's parameters.
- **Test set** This set is used to test model performance on unseen data. This is very helpful for complexity selection.

A good error function for testing the complexity is the Root Mean Square Error (RMSE):

$$E_{RMS} = \sqrt{\frac{2RSS(\hat{\boldsymbol{w}})}{N}}.$$

Unlike the loss function used at training time, E_{RMS} does not monotonically decrease with model complexity, but rather it has a U shape, where the minimum corresponds to the optimal model complexity. We can see the two curves compared

Large parameter values can be an indication of poor generalization in the model, as it suggests that the model may be overfitting the training data. When the model has very large parameters, it tends to oscillate rapidly, which can result in poor performance on new and unseen data.

2.1.2. Regularization

A significant factor in effectively training a machine learning model is avoiding overfitting, which occurs when the model attempts to capture noise in the training dataset, leading to reduced accuracy. The process of constraining or shrinking parameter estimates towards zero is another method for reducing overfitting, which is called regularization. This can be accomplished by modifying the loss function:

$$L(\boldsymbol{w}) = \underbrace{L_D(\boldsymbol{w})}_{\text{error on data}} + \underbrace{\lambda L_W(\boldsymbol{w})}_{\text{error on complexity}} \quad .$$

In most cases, we have:

$$L_D(\boldsymbol{w}) = \frac{1}{2} RSS \qquad\qquad L_W(\boldsymbol{w}) = \frac{1}{2}\sum_{j=1}^{M} |w_j|^q, \quad q \in \mathbb{N}^+.$$

The loss function L_W is dependent on the absolute value of the parameters, which discourages high parameter values. This means that a parameter must have a significant contribution to L_D to justify its high value, or else the model will be penalized. The strength of the penalization can be controlled using the hyperparameter λ, the regularization coefficient, and q. L_W can also be viewed as a constraint on the parameter space, restricting parameters to a certain distance from the origin. This distance is determined by λ and q, with q influencing the shape of the boundary or constraint in parameter space, as shown in Figure 2.2. The most commonly used values for q are 1 and 2.

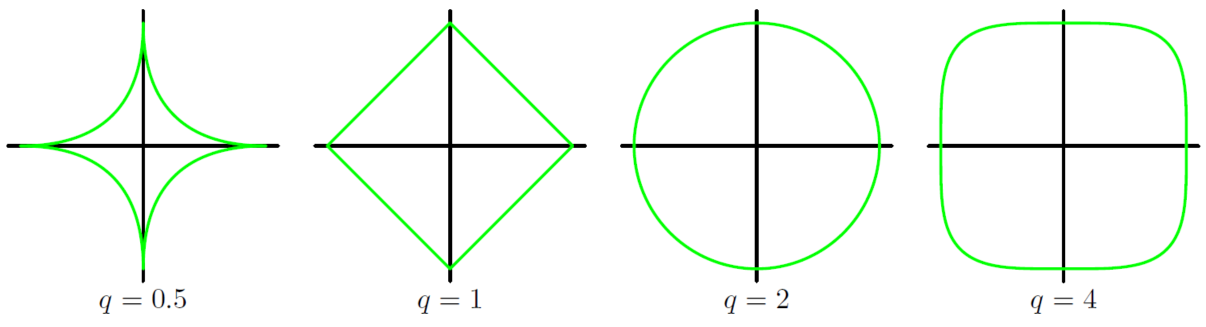


Figure 2.2: Shape of loss function regularizations.

Regularization can help in training complex models on small datasets without causing overfitting by limiting the effective model complexity. However, instead of finding the right number of basis functions, the task of determining the optimal model complexity is shifted to identifying an appropriate value of the regularization coefficient λ.

Ridge regression

For $q = 2$ we have the so-called Ridge regression:

$$\begin{aligned} L_W(\boldsymbol{w}) &= \frac{1}{2} w^\top \boldsymbol{w} = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ L(\boldsymbol{w}) &= \frac{1}{2}\sum_{j=1}^{N}(t_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \frac{\lambda}{2}\boldsymbol{w}^\top\boldsymbol{w} \\ &= \frac{1}{2}(\boldsymbol{t} - X\boldsymbol{w})^\top(\boldsymbol{t} - X\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^\top\boldsymbol{w}. \end{aligned}$$

One of the benefits of ridge regression is that the loss function retains its quadratic form in $\boldsymbol{w}$, enabling us to retain a closed-form solution:

$$\hat{\boldsymbol{w}}_{ridge} = (\lambda I + X^\top X)^{-1} X^\top \boldsymbol{t}.$$

The eigenvalues of $(\lambda I + X^\top X)$ remain non-negative since $X^\top X$ is semi-definite positive, and the addition of λI only sets a positive lower bound on the eigenvalues. Therefore, $(\lambda I + X^\top X)$ is always semi-definite positive and invertible, which is a very useful property that we often use in practice.

Lasso

Lasso regression instead, uses the absolute value of the coefficients instead of their squared values in the regularization term. We obtain its formulation using $q = 1$:

$$\begin{aligned} L_W(\boldsymbol{w}) &= \frac{1}{2}\sum_{j=1}^{M}|w_j| = \frac{1}{2}\|\boldsymbol{w}\|_1 \\ L(\boldsymbol{w}) &= \frac{1}{2}\sum_{j=1}^{N}(t_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j| \\ &= \frac{1}{2}\sum_{j=1}^{N}(t_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_1 \end{aligned}$$

Unlike ridge regression, lasso is a non-linear model. A closed-form solution does not exist due to the presence of the absolute value operator in L_W. However, an advantage of lasso

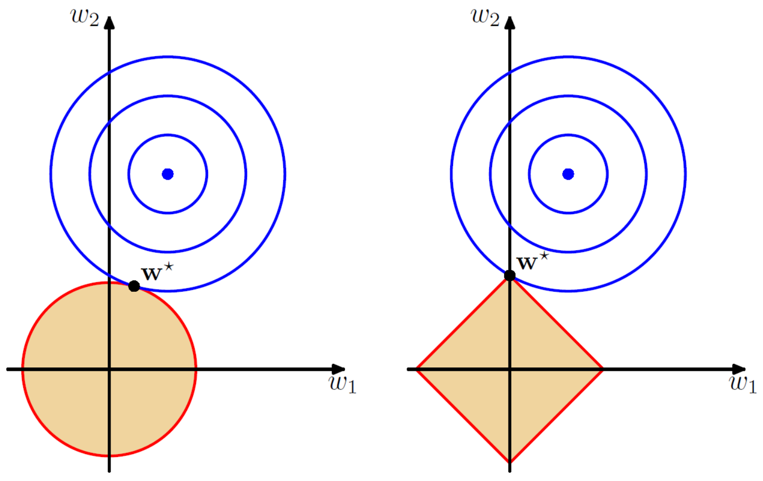


Figure 2.3: Ridge and Lasso loss functions.

is that it is capable of setting some weights to exactly zero for sufficiently large values of λ. As a result, lasso can lead to sparse models, where some parameters tend to zero and certain features are eliminated from the model.

In Figure 2.3, the lines in red represent $\sum_{j=1}^{M} |w_j|^q$ respectively for $q = 2$ (Ridge) and $q = 1$ (Lasso), while the blue lines are the unregularized error functions.

2.2. Reinforcement Learning

Reinforcement Learning (RL) [31] is a paradigm within the field of Machine Learning that studies algorithms capable of improving their behavior through interaction with an environment. Unlike Supervised and Unsupervised Learning, which focus on learning from labeled and unlabeled data, respectively, RL concerns the relationship between agents and environments where the agents learn how to act based on feedback in the form of rewards. The core of RL is based on trial-and-error exploration, where agents are not given explicit instructions on how to act in a given state, but must instead infer the optimal actions to achieve predefined goals by drawing from their experiences of previous rewards and possible actions and states.

The concept of Reinforcement Learning can be formalized through the use of the Markov Decision Process (MDP), a mathematical framework used to model decision-making problems in uncertain environments. An MDP is characterized by a set of states, a set of actions, a reward function that assigns a scalar reward to each state-action pair, and a transition function that defines the probability of moving from one state to another based on the current state and selected action. The next state depends only on the current state and the selected action, and not on previous states and actions, which is referred to as the Markov property.

Definition 2.1. *A Markov Decision Process is a tuple* $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu >$ *where:*

- $\mathcal{S}$ *is a set of states;*
- $\mathcal{A}$ *is a set of actions;*
- $\mathcal{P}$ *is a transition function,* $\mathcal{P} = (p_a : a \in \mathcal{A})$*, where* $p_a = P\left(s'|s, a\right) \quad s, s' \in \mathcal{S}$*;*
- $\mathcal{R}$ *is a reward function,* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$*;*
- μ *is a set of initial probabilities,* $\mu_i = P\left(s_0 = s_i\right) \quad \forall s_i \in \mathcal{S}$*.*

The agent interacts with the environment by observing his current state s_t, then taking an action $a_t \in \mathcal{A}$, which results in a reward y_t, generated by the reward function $\mathcal{R}(s_t, a_t)$. The interaction leads to a transition from the current state s_t to a new state s_{t+1}, with a probability determined by the transition function $p_{a_t}(s_t)$.

In essence, the MDP provides the mathematical formulation of decision-making problems in uncertain environments, while RL provides the algorithms and methods to solve these problems by finding the optimal policy for an agent in a given MDP, whose structure may be unknown.

2.3. The Multi-Armed Bandits Problem

A multi-armed bandit is a classical problem in the field of reinforcement learning that models a decision-making scenario where a learner must choose among several options, referred to as "arms," to maximize their total reward over a limited number of trials. Each arm has an unknown reward distribution and the learner must balance exploration of the different arms to gather information about their expected reward and exploitation of the arm with the highest estimated reward. The multi-armed bandit problem has been extensively studied, both theoretically and experimentally, and has numerous real-world applications, such as online advertising, clinical trials, recommendation systems, and dynamic pricing [8, 25]. The challenge in solving the multi-armed bandit problem lies in finding an optimal balance between exploration and exploitation that maximizes the total reward.

A Multi-Armed Bandit (MAB) can be considered as a simplified version of the Markov Decision Process (MDP), where there is a single state, $\mathcal{S} = \{s\}$, and the agent-environment interaction is limited to the selection of an action a_t (referred to as an arm), that results in some reward y_t.

Definition 2.2. *A Stochastic Multi-Armed Bandit is a tuple* $< \mathcal{A}, \mathcal{R} >$ *where:*

- $\mathcal{A}$ *is a set of arms;*
- $\mathcal{R}$ *is the set of unknown distributions, such that* $\mathcal{R}(s, a_i) = \mathcal{R}(a_i)$.

From Definition 2.2, it is evident that the state-related components of Definition 2.1 has been removed, since we have a unique state s.

Formally, in the standard stochastic bandit setting, given a set of K arms, where each arm a_i has an unknown expected reward μ_i. At each time step t, the algorithm selects an arm a_t to play and receives a reward y_t drawn from the unknown distribution of the selected arm $\mathcal{R}(a_i)$. The goal of the algorithm is to maximize the total expected reward over a fixed time horizon T, while balancing the trade-off between exploration (trying out different arms to gather information about their expected rewards) and exploitation (selecting the arm with the highest estimated expected reward).

Algorithm 2.1 summarizes the interaction protocol between the bandit algorithm and the environment.

Algorithm 2.1 MAB Interaction Protocol

```
1: for $t = 1, 2, \ldots T$ do
2:     Agent chooses an action $a_t$ from a set $\mathcal{A}$ of available actions. The chosen action is sent to the environment
3:     The environment generates a response in the form of a real-valued reward $y_t \in \mathbb{R}$, which is sent back to the learner
4: end for
```

2.3.1. Objective

The goal of the learner is to maximize the sum of rewards that it receives $\sum_{t=1}^{T} y_t$. In the standard stochastic setting, as described by Definition 2.2, this objective function can be reformulated as the minimization of the expected regret, defined as:

$$R_T = T \cdot \max_{i \in [K]} \mu_i - \mathbb{E}\left[\sum_{t=1}^{T} y_t\right],$$

where $T \cdot \max_{i \in [K]} \mu_i$ is the expected reward obtained by always playing the arm with the highest expected reward, and $\mathbb{E}\left[\sum_{t=1}^{T} y_t\right]$ is the expected reward obtained by the algorithm. The goal of the algorithm is to minimize its regret, since maximizing reward is the same as minimizing regret. This formulation is particularly helpful because it lets us evaluate the algorithm's learning speed. In fact, we expect good learning to have a regret sublinear in the number of rounds, i.e., $R_T = \widetilde{\mathcal{O}}(\log T)$, or equally $\lim_{T \to \infty} \frac{R_T}{T} = 0$. This means that at every successive rounds, the algorithm incurs in an immediate regret (regret calculated over a single round) that is slightly smaller than the previous round. Conversely, if the regret is linear, the algorithm doesn't learn anything, since it repeats the same mistakes, and does not improve its performance over time.

2.4. Optimism in the Face of Uncertainty

The principle of Optimism in the Face of Uncertainty (OFU) is the cornerstone of a wide range of Multi-Armed Bandit algorithms, aimed at resolving the exploitation-exploration dilemma. The earliest formalization of this principle dates back to 1952 [27], was further theoretically analyzed and improved upon under the name of Upper Confidence Bound (UCB) [2, 5, 17].

A core algorithm that belongs to this family is UCB1, which assigns upper confidence bound $\hat{\mu}_i(t) + \sigma_i(t)$ to each expected reward μ_i of distribution $\mathcal{R}(a_i)$. Here, $\hat{\mu}_i(t)$ is an estimate of the true expected reward μ_i, and $\sigma_i(t)$ is an *exploration bonus* chosen such that $\hat{\mu}_i(t) - \sigma_i(t) \leq \mu_i \leq \hat{\mu}_i(t) + \sigma_i(t)$ with high probability. The algorithm selects the arm with the maximal upper confidence bound $\hat{\mu}_i(t) + \sigma_i(t)$ at each trial t.

At each round t, the algorithm chooses the arm $a_i(t)$ to play as follows:

$$a_i(t) = \arg\max_{i \in [K]} \left(\hat{\mu}_i + \sqrt{\frac{2 \ln t}{T_i(t)}} \right),$$

where $T_i(t)$ is the number of pulls of arm a_i at time t.

Exploration trials occur when an arm with a large $\sigma_i(t)$ is selected, as the estimate $\hat{\mu}_i(t)$ is unreliable. Conversely, exploitation trials occur when an arm with a large $\hat{\mu}_i(t)$ is selected. The decrease in $\sigma_i(t)$ over time ensures that the number of exploration trials is limited. If $\sigma_i(t)$ is small, then $\hat{\mu}_i(t)$ is close to μ_i, and the optimal arm with the maximum μ_i is selected in exploitation trials. Confidence bounds and a generalization of the concepts discussed here is the essence of the OFU principle.

2.5. Linear Bandit

Stochastic linear bandits are subclass of bandit problems that tackle the problem of large action spaces by incorporating a structure that enables the algorithm to generalize across actions.

The problem was first studied by Auer [4] and then improved upon by Abbasi-Yadkori et al. [1], Dani et al. [13], Li et al. [19]. For a comprehensive overview, refer to [18, 29, 37].

Mathematically, we can describe the problem as follows: at each round t, the learner chooses a vector arm $\boldsymbol{a}_t$ from a set $\mathcal{A}_t \subseteq \mathbb{R}^d$ and receives a reward:

$$y_t = \boldsymbol{\theta}_*^\top \boldsymbol{a}_t + \eta_t,$$

where θ_* is some parameter in $\mathbb{R}^d$ unknown to the agent and η_t a zero-mean conditionally σ-Sub-Gaussian, where $\sigma \geq 0$ is a fixed constant, which is equivalent to the following condition: $\forall \lambda \in \mathbb{R}, \mathbb{E}\left[e^{\lambda \eta_t} \mid a_{1:t}, \eta_{1:t-1}\right] \leq \exp\left(\frac{\lambda^2 R^2}{2}\right)$ [18].
The agent seeks to maximize the cumulative expected reward given by $\sum_{t=1}^{T} \langle \boldsymbol{a}_t, \theta_* \rangle$. As in the general case, we can state this objective function in terms of regret. The goal in this formulation becomes to minimize the regret over a fixed time horizon T. In the stochastic linear bandit setting the random pseudo-regret and regret are respectively defined as:

$$\hat{R}_T = \sum_{t=1}^{T} \max_{\boldsymbol{a} \in \mathcal{A}_t} \langle \boldsymbol{\theta}_*, \boldsymbol{a} - \boldsymbol{a}_t \rangle \,, \tag{2.2}$$

$$R_T = \mathbb{E}\left[\hat{R}_T\right] = \mathbb{E}\left[\sum_{t=1}^{T} \max_{\boldsymbol{a} \in \mathcal{A}} \langle \boldsymbol{\theta}_*, \boldsymbol{a} \rangle - \sum_{t=1}^{T} y_t\right] . \tag{2.3}$$

Equation 2.2 defines the random pseudo-regret, which is the cumulative loss incurred by the algorithm in terms of the difference between the reward obtained by the optimal action and the reward obtained by the action chosen by the algorithm at each time step t. Specifically, the equation sums over all T time steps and takes the maximum inner product, denoted by the angle brackets, between the true parameter vector $\boldsymbol{\theta}_*$ and the difference between the action taken by the algorithm $\boldsymbol{a}_t$ and the optimal action.

Equation 2.3 defines the expected regret, which is the expected value of the random pseudo-regret. This equation is obtained by taking the expectation of the random pseudo-regret over all possible values of the reward y_t obtained by the algorithm at each time step t. The expected regret is thus the expected difference between the cumulative reward obtained by the algorithm and the cumulative reward obtained by the optimal action, summed over all T timesteps. While the random regret is affected by the noise in the rewards, the expectation effectively filters it out, making it arguably a better measurement of an algorithm's skill.

3 | Related Works

In this chapter, we review the relevant literature on the MAB areas related to our research topic. In particular, in the first section, we start by analyzing a prominent algorithm for linear bandits, which is very relevant to our research since it serves as a foundation for our algorithm. In the following section, we outline the existing approaches and algorithms for Multi-Agent MABs.

3.1. LinUCB

The linear bandit problem described in Section 2.5 can be addressed through an algorithmic approach that utilizes the aforementioned OFU principle. There are different terms used to refer to the application of UCB to linear bandits, such as LinRel (linear reinforcement learning), LinUCB, and OFUL (optimism in the face of uncertainty for linear bandits).

LinUCB [12, 19] builds on this principle, and extends the UCB algorithm to the linear setting. The core of the idea is to maintain a confidence set for the vector of coefficients θ of the linear function that generates the rewards. At each round, the algorithm selects an estimate of the coefficients of the linear function from the confidence set and chooses an action that maximizes the predicted reward. The problem, therefore, becomes constructing confidence sets for the vector of coefficients of the linear function based on observed action-reward pairs in previous time steps.

Let $H_t \in \mathcal{R}^{t\times d}$ and $\boldsymbol{y}_t \in \mathcal{R}^t$ be the vector of historical rewards up to time t, where each row j of H_t represents the feature vector of size d of the arm pulled at time j and the j^{th} element of $\boldsymbol{y}_t$ is the corresponding reward. Since the reward is generated by the linear function:

$$\boldsymbol{y}_t = \boldsymbol{\theta}_*^\top H_t,$$

we can estimate the ridge regressor of θ_* in closed-form:

$$\hat{\boldsymbol{\theta}}_t = (H_t^\top H_t + \lambda I_d)^{-1} H_t^\top \boldsymbol{y}_t. \tag{3.1}$$

This same estimator is reported in Algorithm 3.1 in incremental form. Note that V_t is the incremental regularized Gram matrix $\lambda \mathrm{I}_d + H_t^\top H_t$ and $\boldsymbol{b}_t$ corresponds to $H_t^\top \boldsymbol{y}_t$. Moreover, $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$ is a constant, where δ is the error probability.

Algorithm 3.1 LinUCB

```
1: Input: $\alpha > 0, \lambda > 0$
2: $V_0 = \lambda \mathrm{I}_d$, $\boldsymbol{b}_0 = \mathbf{0}_d$
3: for t = 1, 2, ..., T do
4:     $\boldsymbol{\theta}_t = V_{t-1}^{-1} \boldsymbol{b}_{t-1}$
5:     for $\boldsymbol{a} \in \mathcal{A}$ do
6:         $\mathrm{UCB}_t(\boldsymbol{a}) = \hat{\boldsymbol{\theta}}_t^\top \boldsymbol{a} + \alpha \sqrt{\boldsymbol{a}^\top V_{t-1}^{-1} \boldsymbol{a}}$
7:     end for
8:     Choose arm $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}} \mathrm{UCB}_t(\boldsymbol{a})$
9:     Receive reward $y_t$
10:    $V_t = V_{t-1} + \boldsymbol{a}_t \boldsymbol{a}_t^\top$
11:    $\boldsymbol{b}_t = \boldsymbol{b}_{t-1} + \boldsymbol{a}_t y_t$
12: end for
```

Among the different available formulations, we chose to report the version of Li et al. [19], since it seems the most intuitive.

3.1.1. Regret

We report a regret bound for LinUCB [18], valid under the following assumptions.

Assumption 3.1. *The following hold:*

1. $1 \le \beta_1 \le \beta_2 \le \cdots \le \beta_T$.
2. $\max_{t\in[T]} \sup_{a,b\in\mathcal{A}_t} \langle \boldsymbol{\theta}_*, \boldsymbol{a} - \boldsymbol{b} \rangle \le 1$.
3. $\|\boldsymbol{a}\|_2 \le L$ *for all* $\boldsymbol{a} \in \bigcup_{t=1}^T \mathcal{A}_t$.
4. *There exists a* $\delta \in (0,1)$ *such that with probability* $1-\delta$*, for all* $t \in [T]$*,* $\boldsymbol{\theta}_* \in \mathcal{C}_t$ *where* $\mathcal{C}_t$ *satisfies:*

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

Assumption 1 specifies that $(\beta_t)_t$ is an increasing sequence of constants with $\beta_1 \ge 1$, and assumptions 2-3 impose the boundedness of the rewards and action set respectively. Assumption 4 instead states that the optimal parameter $\boldsymbol{\theta}_*$ falls within the set $\mathcal{E}_t$ with probability $1-\delta$. The set $\mathcal{E}_t$ is an ellipsoid centered at $\hat{\boldsymbol{\theta}}_{t-1}$ and with principle axis being the eigenvectors of V_t with corresponding lengths being the reciprocal of the eigenvalues.

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

$$\hat{R}_T \leq \sqrt{8T\beta_T \log\left(\frac{\det V_T}{\det V_0}\right)} \leq \sqrt{8dT\beta_T \log\left(\frac{d\lambda + TL^2}{d\lambda}\right)}. \tag{3.2}$$

where a suitable choice for β_T may be:

$$\sqrt{\beta_T} = \sqrt{\lambda} m_2 + \sqrt{2\log\left(\frac{1}{\delta}\right) + d\log\left(\frac{d\lambda + TL^2}{d\lambda}\right)}, \tag{3.3}$$

where m_2 is an upper bound on $\|\theta_*\|_2$.

The variables on which the regret depends are the key elements to note in the equation above. To make them more evident, under the same assumptions, we can simplify the bound above by choosing $\delta = 1/T$. We obtain that the expected regret of LinUCB is bounded by:

$$R_T \leq Cd\sqrt{T}\log(TL),$$

where $C > 0$ is a suitably large universal constant. This shows that the expected regret depends on the horizon T, the dimension d, and arm bound L. Furthermore, the constant C also hides a dependency on the upper bound on $\|\boldsymbol{\theta}_*\|_2$ and the regularization hyperparameter λ.

3.2. Clustering of Bandits

The problem of clustering bandits has been actively researched in the field of recommendation systems, where users and/or items, along with their interactions are represented as a graph. The goal of these works is to design a feedback-sharing mechanism to leverage user similarities embedded in the graph. The baselines these works are usually compared against are called LinUCB-ONE and LinUCB-IND, which learn respectively one single instance of LinUCB across all users, and one independent instance per user. Most of the works try to find the sweet spot by employing one linear bandit per cluster and differentiate themselves based on how they perform the clustering, i.e., how they determine which users are "similar" enough to be clustered together.

The *gang of bandits* model introduced by Cesa-Bianchi et al. [11], represents users as nodes on a graph and assigns a linear bandit instance to each node, enabling information exchange between various nodes. After serving the current user using the UCB policy, its bandit weights are locally updated, while its feedback is also, to a lesser extent, shared with nearby nodes by reproducing kernel Hilbert space, used in the literature for solving online multitask classification problems [9], defined in terms of the Laplacian matrix of the graph. The main limitation of this approach is that it scales quadratically with the nodes, Vaswani et al. [34] show that the computational workload can be reduced, using ideas from Gaussian Markov Random Fields and Thompson Sampling, thus resulting in a more scalable algorithm that can be applied to larger graphs.

CLUB [14] also represents the similarity of users by a graph, and obtains the clustering through a partition of the graph, which is induced by repeatedly deleting edges between users whose bandit weights differ significantly. The algorithm works under the assumption that nodes within the same cluster exhibit the same behaviour.

In the follow-up work [20], a "two-sided" clustering method is introduced, which clusters users based on their expected payoffs on a particular item (as the previous works), but also clusters items based on the similarity of user preferences. Gentile et al. [15] extend this idea by developing a simplified version of the algorithm, that also allows merging clusters that were once erroneously separated and can handle situations where the set of items is unknown and potentially infinite.

SCLUB [21] is a further improvement that deals with users who are drawn from a non-uniform distribution and abandon the graph data structure used in previous methods in favor of a set-based clustering representation. Their method involves building clusters by splitting a user into a separate cluster if their estimated bandit weight or context frequency is significantly different from the rest of the cluster, and vice-versa for the merge operation. This allows the algorithm to adapt to changes in the distribution of users and improve its performance in online learning tasks.

3.3. Multi-task Bandits

Another adjacent area of research is the multi-task bandits literature, which aims to transfer knowledge across several similar bandit problems while managing the classical exploration-exploitation trade-off.

The multi-task linear bandit setting [30] works under the assumption that all the tasks are similar, where the similarity of two task is defined as having L2-norm of the difference of bandit parameters smaller than some threshold. This approach allows exploiting the fact that if two bandits have similar parameter vectors, then knowledge of one's interactions can be exploited to minimize regret of a learning algorithm while interacting with the other. Their algorithmic solution is a variant of LinUCB that constructs two confidence bounds, one task-specific and multi-task bounds, and at each round considers the tightest confidence bound, since it's the most certain, and then selects the arm with the largest retained confidence bound, in typical UCB fashion.

Xu and Bastani [36] consider the setting where the unknown parameter in each linear bandit instance can be decomposed into a global parameter plus a sparse instance-specific term. They propose propose an estimator that combines the trimmed mean from robust statistics to learn across similar instances and LASSO regression to debias the results by capturing the instance-specific information. This estimator, called Robust Multitask Estimator, is then used in N (number of users) linear bandits running at each problem instance.

Meta-Learning is a closely related area, where the goal is to select a learning algorithm which works well on average over a class of bandits tasks. For example, Cella et al. [10] present a regularized version of OFUL, where the regularization depends on the Euclidean distance to a bias vector, and propose several methods to estimate it under different assumptions.

Other works have proposed imposing and learning a shared Bayesian prior across bandit instances [6, 16]. A potential weakness is the assumption that bandit instances appear sequentially to learn the prior, which does not hold in many settings.

Note that in our work we consider solely the setting where there are multiple bandits that share information. However, in the literature, the terms bandit clustering and multi-task bandit also denote the problems of aggregating arms, sometimes called tasks as well.

4 | Problem Formulation

In this chapter we give both the intuitions and formal definitions of the setting that the proposed algorithm will solve, along with its assumptions and objective. Specifically, we introduce the Multi-Agent Stochastic Linear Bandit Setting, the general bandit problem that we tackle in this thesis.

4.1. Multi-Agent Stochastic Linear Bandit Setting

The multi-agent stochastic linear bandit setting is an extension of the stochastic linear bandit problem, where N linear bandits are trying to solve different yet related tasks. For a thorough explanation of the linear bandit setting please refer to Section 2.5. Each linear bandit i is characterized by a parameter vector $\boldsymbol{\theta}^i \in \mathbb{R}^{d_i}$. At each time step t, an external context index $i \in \{1, \dots, N\}$ arrives, determining which agent will be the actor of the current turn, out of the N available agents. Following the standard MAB interaction protocol (Algorithm 2.1), at each time step t, agent i chooses a vector arm $\boldsymbol{a}_{i,t} \in \mathcal{A}_i \subseteq \mathbb{R}^{d_i}$, where $\mathcal{A}_i$ is the set of available arms and d_i is the i-th vector dimension. The set of possible arms can be either discrete or continuous, and the dimensions of the arms of different bandits can be different, hence the subscript i for the dimension. The agent then receives a scalar reward $y_t \in \mathbb{R}$, which is generated by a linear function of the arm vector, and an additive stochastic noise, that characterizes the stochastic bandit setting. The function generating the reward is:

$$y_t = \boldsymbol{\theta}^{i\top} \boldsymbol{a}_{i,t} + \eta_t,$$

where $\boldsymbol{\theta}^i$ is the (unknown) bandit parameter, different for each agent i, and η is a zero-mean subgaussian random noise. It's worth underlining that all elements of the formulation above are specific to bandit i, and therefore any information on bandit i gives no advantage to solving on any other bandit instance $j \neq i$.

We make the following standard assumption on the noise η.

Assumption 4.1. *We assume the noise η is conditionally σ-subgaussian for some con-*

stant $\sigma > 0$. That is, $\forall \lambda \in \mathbb{R}$

$$\mathbb{E}\left[\exp(\lambda\eta) \mid \mathcal{F}_{t-1}\right] \leq \exp\left(\lambda^2\sigma^2/2\right).$$

Furthermore, we assume that arms and parameters are bounded, which means that the range of possible values for the arms and parameters is limited. This allows us to work with a finite set of values and avoid infinite or unbounded behavior. Additionally, the boundedness assumption makes the problem more tractable and easier to analyze mathematically.

Assumption 4.2. *There exist some $L, S > 0$ such that $\|\boldsymbol{\theta}_*\| \leq S$ and $\|\boldsymbol{a}\| \leq L$ for all $\boldsymbol{a} \in \mathcal{A}$. Let $\mathcal{F}_t$ be the σ-algebra generated by $(\boldsymbol{a}_1, y_1, \ldots, \boldsymbol{a}_t, y_t)$, then $\boldsymbol{a}_t$ is $\mathcal{F}_{t-1}$-measurable.*

For example, if we consider a linear bandit problem where the rewards are linear functions of the arms features, boundedness ensures that the range of possible reward values is finite. This means we can design algorithms that take advantage of this finite range to explore and exploit the arms efficiently without worrying about the algorithm diverging or behaving unexpectedly.

4.1.1. Regret

Each individual bandit i aims to maximize the cumulative reward over time horizon T, which is equivalent to minimizing the regret, as defined by Equation 2.3. Since we take into account all agents together and not each individual separately, the overall objective function of the problem is to maximize the sum of the cumulative rewards throughout all bandits $\sum_{t=1}^{T} y_t$, where i denotes the agent i served at time t. Analogously, as in typical bandit fashion, we reformulate the objective function as the minimization of the cumulative regret, thus resulting in the following formulation:

$$R_T = \sum_{t=1}^{T} \max_{\boldsymbol{a}_{i_t,t} \in \mathcal{A}_{i_t}} \boldsymbol{\theta}_*^{i_t \top} \boldsymbol{a}_{i_t,t} - \mathbb{E}\left[y_t\right],$$

where $\boldsymbol{\theta}_*^i$ is the optimal parameter vector for agent i, and y_t is the actual reward collected. The expectation is with respect to the randomness in the environment and policy. The first term in this expression is the maximum expected reward using any policy since the parameter is known to be optimal, and the max operator selects the action that yields the highest reward. The second term is the expected reward collected by the learner.

4.2. Partitioned Linear Bandit Setting

The peculiarity of the partitioned setting consists in the decomposition of the linear bandit parameter, which has k of its components shared across all bandits, with $k < d$, and the remaining $d - k$ components specific to each bandit instance.

In the following sections, we often use the slicing notation $:k$ or $k:$ as a subscript to denote the first k or last k components of a vector. To simplify the notation here we will characterize them as global and local components, respectively, and use the subscript g for the subvector containing the first k components of a vector and l for the last $d - k$ components.

Therefore, we have a unique parameter vector $\boldsymbol{\theta}_g$ and a set of parameter vectors specific to each agent $\{\boldsymbol{\theta}_l^1, \dots, \boldsymbol{\theta}_l^N\}$. At each time step t the agent i chooses an arm $\boldsymbol{a}_{i,t}$ and observes the reward y_t generated as follows:

$$y_t = \boldsymbol{\theta}_g^\top \boldsymbol{a}_{g,t} + \boldsymbol{\theta}_l^{i\top} \boldsymbol{a}_{i,t} + \eta_t. \tag{4.1}$$

Note that the local parameter vector is specific to agent i, whereas the global one is common to all agents. The meaning of the equation is that we have two independent components that contribute to the reward of each agent i. There is a global component, shared by all bandits, that is a linear function of the first k components of the arm, and a local component specific to the agent to be served, which is also a different linear function of the remaining part of the arm. There is also the additive σ-subgaussian noise of the stochastic multi-armed bandit setting that follows the standard Assumption 4.1. Likewise, Assumption 4.2 on the boundedness of arms and parameters applies in this case as well.

5 | Proposed Algorithm

The goal of this algorithm is to provide a solution to the bandit problem laid out in the previous section, by minimizing the cumulative regret over a given time horizon. In order to minimize the regret, the algorithm will have to exploit all the available structure given by the problem setting, primarily consisting of the linearity assumption, i.e., the reward is a linear function of the parameters for each bandit, and the partitioning assumption which states that all bandits share a piece of the parameter vector.

5.1. Partitioned LinUCB

The most immediate approach to solve the setting is to utilize one single LinUCB instance for all tasks $i \in \{1 \dots N\}$. This algorithm correctly exploits the linearity assumption, and will learn the shared subvector $\boldsymbol{\theta}_g$ optimally, since the regression for $\boldsymbol{\theta}$ will always use data points from all tasks. However, this gain comes at the cost of completely disregarding any variance on the local components $\boldsymbol{\theta}_l$, which could potentially have a great impact on the overall performance.

Another solution would be to employ N independent LinUCB instances, which in the literature is named IND-LinUCB (which stands for INDependent LinUCB). This is a perfectly valid approach, that solves the problem setting as laid out in Chapter 4 in a straightforward way. The bandit learns a different $\boldsymbol{\theta}^i$ parameter vector for each instance i, which includes both the global and local components and achieves sublinear regret.

5.1.1. Algorithm

The proposed algorithm aims to find the sweet spot between the two naive solutions reported above. It does so by leveraging the assumption that the bandit parameter $\boldsymbol{\theta}^i$ consists in the concatenation of the shared and local parameters, denoted as $\boldsymbol{\theta}^i = [\boldsymbol{\theta}_g, \boldsymbol{\theta}_l^i]$. The intuition is that we should use this important piece of information to learn $\boldsymbol{\theta}_g$ using the *(action, reward)* pairs from all bandit instances, while learning $\boldsymbol{\theta}_l^i$ separately, since data coming from another instance $j \neq i$ would only bias the estimate of $\boldsymbol{\theta}_l^i$, with no added benefit. To achieve this result, we start with each independent bandit instance learning the whole θ_i on its own, until one of them achieves a low enough regression error, such that we can consider its estimate of the θ_g reliable. From that point on, we will only update the local components of the parameter, which will speed up the learning process.

Algorithm 5.1 PARTITIONED LINUCB

```
 1: Input: $N, T, \{\mathcal{A}_i\}_{i=1}^N, k, \varepsilon, w, \lambda$
 2: $is\_split$ = false
 3: Instantiate $\text{LINUCB}_i(\mathcal{A}_i, \lambda) \quad \forall i = 1, \dots, N$
 4: for $t = 1, 2, \dots, T$ do
 5:     Receive index $i$
 6:     if $is\_split$ then
 7:         $\boldsymbol{a}_l = \arg\max_{\boldsymbol{a} \in \mathcal{A}_{i,l}} \text{UCB}_i(\boldsymbol{a})$
 8:         Pull arm $\boldsymbol{a}_{i,t} = [\boldsymbol{a}_g, \boldsymbol{a}_l]$
 9:         Receive reward $y_t$
10:         $y_{l,t} = y_t - \boldsymbol{\theta}_{\boldsymbol{g}}^\top \boldsymbol{a}_g$
11:         $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}_i, y_{l,t})$
12:     else
13:         Pull $\boldsymbol{a}_{i,t} = \arg\max_{\boldsymbol{a} \in \mathcal{A}_i} \text{UCB}_i(\boldsymbol{a})$
14:         Receive reward $y_t$
15:         $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}_{i,t}, y_t)$
16:         if $\text{AGGREGATIONCRITERION}(\boldsymbol{y}_{i,hist}, \hat{\boldsymbol{y}}_{i,hist}, w, \varepsilon)$ then
17:             $\text{LINUCB}_j = \text{SPLIT}(\text{LINUCB}_j, k) \quad \forall j \neq i$
18:             $\boldsymbol{a}_g = \boldsymbol{a}_{i,t,:k}$
19:             $is\_split$ = true
20:         end if
21:     end if
22: end for
```

5.1.2. Set-up

Algorithm 5.1 reports the pseudocode of this approach. The algorithm takes as inputs the number of agents N, time horizon T, the set of arms of each agent $\{\mathcal{A}_i\}_{i=1}^N$, the partitioning parameter k that splits the arms dimension in global vs local, the sliding window dimension w and the regularization hyper-parameter λ. The instantiation of the N LinUCBs happens according to Line 2 of the standard LinUCB algorithm (Algorithm 3.1).

5.1.3. Update

The update of the parameters of bandit i is denoted with $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}, r)$, which happens according to Lines 10-11 of the same algorithm. At each time step t a bandit index i is sampled from the unknown context distribution, and a different action and update are performed based on whether we are in the first few rounds, or we have already performed the aggregation. If the aggregation has not happened yet, we follow the standard bandit protocol (as per Algorithm 2.1), and then we check the condition for the aggregation (Line 16), following a defined criterion.

5.1.4. Aggregation Criterion

The split is performed only the first time the aggregation condition is met. This condition is captured in the AggregationCriterion subroutine, which defines the specific metric and condition to use to evaluate the goodness of the estimate of $\hat{\boldsymbol{\theta}}$.

To assess the quality of the estimation of $\hat{\boldsymbol{\theta}}$ we evaluate the regression error on the prediction of the reward. Among the plethora of available regression metrics, we chose the Mean Absolute Percentage Error (MAPE), defined as:

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|,$$

where y_t is the actual value at time t, $\hat{y}_t$ is the forecast value at time t, and n is the total number of observations. Bear in mind that in our setting the forecast value $\hat{y}_t$ represents the predicted reward, which is computed as $\hat{y}_t = \boldsymbol{\theta}^{i\top}\boldsymbol{a}_{i,t}$ and stored in the vector denoted as $\hat{\boldsymbol{y}}_{i,hist}$, and the true value y_t is the noisy reward generated by the environment, stored in the vector $\boldsymbol{y}_{i,hist}$.

Our choice for the regression error metric fell on the MAPE because it provides a relative scale, i.e., a score from 0 to 1, which makes choosing a threshold value somewhat more generalizable, and therefore allows for a more intuitive interpretation compared to other metrics such as the Mean Average Error or the Mean Squared Error.

Since the regression model improves over time, we do not use all past samples to compute the MAPE, but rather perform a moving average over the last w samples, which would correspond to the n of the above equation. This allows for a robust evaluation of the regression performance, for a big enough value of w, while also allowing to forget past inaccurate predictions, a value of w, while also allowing to forget the past incorrect prediction, characteristic of the online learning setting.

In summary, when the moving average over the last w samples of the residual of the regression for $\boldsymbol{\theta}^i$ is smaller than some threshold $\varepsilon > 0$, we know that we have a good enough estimate of the true parameter of bandit i, and each agent i may focus solely on learning its local parameters.

It is important to note that this criterion was derived from experimental evaluations, as well as the threshold value. Although this approach proves to be experimentally robust and works well in practice. we suppose that a theoretical analysis could provide a more sound criterion, which would only need to be plugged in as the new AggregationCriterion subroutine.

Algorithm 5.2 AggregationCriterion

```
1: Input: $\boldsymbol{y}_{hist}, \hat{\boldsymbol{y}}_{hist}, w, \varepsilon$
2: $e_t := \left|\frac{y_t - \hat{y}_t}{y_t}\right|$
3: if $\frac{e_t + \ldots + e_{t-w}}{w} < \varepsilon$ then
4:     return true
5: else
6:     return false
7: end if
```

5.1.5. Split

If the above aggregation requirement is satisfied, we save the estimated optimal subarm and run the Split subroutine, which partitions all the parameters of the given agent. Namely, it splits the local subarms, which will be the only ones to be pulled from now on, since the global optimal subarm has been found, and recomputes the local $\hat{\theta}_{l,t}$, using the closed form solution, as in Equation 3.1, using the last k components of the arm history matrix H_t and reward history vector y_t.

It's worth noting that the condition is met under the assumption that there is one of the N products whose reward depends almost exclusively on the global components, such that the local ones give a negligible contribution to the expected reward. From the following time step, the algorithm will execute the first branch of the if statement, where the bandit only chooses the best local subarm, i.e., some $\boldsymbol{a}_l \in \mathcal{A}_{i,l}$. Note that this set may completely differ from all other agents both in terms of content and dimensionality. The agent then pulls the arm obtained as the concatenation of the global and local components. The reward needed for the update is corrected by removing the contribution of the global components since we only want to update the local ones.

5.1.6. Regret

We would like to give an intuition on why we expect the algorithm to perform better on the proposed setting, based on a qualitative analysis of the regret. In Section 2.5 we thoroughly present and break down the components of the regret of the LinUCB

Algorithm 5.3 Split

```
1: Input: LinUCB, $k$
2: for $\boldsymbol{a} \in \mathcal{A}$ do
3:     $\boldsymbol{a} = \boldsymbol{a}_{k:}$
4: end for
5: $\hat{\boldsymbol{\theta}}_{l,t} = (H_{k:,t}^\top H_{k:,t} + \lambda I)^{-1} H_{k:,t}^\top \boldsymbol{y}_{k:,t}$
6: return LinUCB
```

algorithm, upon which our approach is based. Equation 3.2 reports the complete equation of a single LinUCB instance. Clearly, this bound does not apply to the multi-agent setting, in which we instantiate N, different bandits. Indeed, the regret of the aforementioned baseline solution that allocates one independent bandit per context, named IND-LinUCB, has a regret bound given by the sum of the regrets of each bandit:

$$\hat{R}_T \leq \sum_{i=1}^{N} \sqrt{8T\beta_T^i \log\left(\frac{\det V_T^i}{\det V_0^i}\right)} \leq \sum_{i=1}^{N} \sqrt{8d^iT\beta_T^i \log\left(\frac{d^i\lambda + TL^{i^2}}{d^i\lambda}\right)}, \tag{5.1}$$

where the superscript i indicates that the Gram matrix V, dimension d, arm bound L or β_T may be different for each bandit. The main point of the above equation is that since each agent is independent, the regret is the sum of the regrets generated by N independent instances. It's worth noting that the bound mentioned earlier is a pessimistic estimate because not all N bandits will have their arms pulled exactly T times. The actual number of pulls of each bandit is determined by the random distribution of contexts.

The regret analysis changes substantially in case there is information sharing among the bandits. Our algorithm will go through an *exploration phase*, up to a time T_{expl}, where the regret will be the same as above, and then it will have a *partial exploitation phase*, from $T_{expl} + 1$ up to T, where the global parameters will be fixed and shared by all bandit instances.

In fact, in the latter phase, since we won't have to learn the global components of θ, we expect that the regret will be inferior. After T_{expl} the algorithm will effectively run as N new independent bandits with reduced dimensions of the arms and parameters. Indeed, we expect the gain to come primarily from the reduction of the dimension d, the arm bound L and θ bound m_2 (from Equation 3.3).

6 | Experimental Evaluation

This chapter presents an experimental evaluation of our proposed algorithm, with the primary focus on analyzing their performance using the random regret metric. As formally defined by Equation 2.2, the random regret measures the difference in reward obtained by our algorithms compared to that of an ideal agent, referred to as the *Clairvoyant*. This agent possesses complete knowledge of the environment's parameters and selects actions that yield maximum rewards. In essence, we compute the random regret of our algorithms by comparing the loss in reward incurred at each round against that of the Clairvoyant.

In the partitioned arms setting, the environment generates rewards based on the decomposition of the linear bandit parameter, which comprises k shared components across all bandits and $d-k$ components specific to each bandit instance, where d is the dimensionality of the entire arm and $\boldsymbol{\theta}$ vectors. Therefore the global parameters are responsible for a piece of the reward, and the local parameters are responsible for another additive piece of the reward. Bare in mind that as in the classic Stochastic Multi-Armed Bandit, there is also a stochastic component, due to the σ-Subgaussian noise.

6.1. Baseline

Before presenting the experimental evaluation of our algorithm, we analyze similarities and differences of our algorithm with respect to a well-known adaptation of the classical LinUCB algorithm, which was thoroughly discussed in Sections 2.5 and 3.1.

The LinUCB algorithm exploits the linearity assumption, and therefore only needs to learn one parameter θ shared by all arms that generate the reward, making the problem easier to learn, as arms are not independent anymore, but tied together by this parameter. We will now discuss how we exploit the additional structure given by our specific problem setting laid out in Section 4.2, and how an existing variation of the LinUCB algorithm fails to exploit all the assumptions of the setting.

6.1.1. IND-LinUCB

An improvement on the basic LinUCB approach to solving the setting would be to employ N independent LinUCB instances, which in the literature is named IND-LinUCB (which stands for INDependent LinUCB). This is a perfectly valid approach since the bandit learns for each instance the whole $\boldsymbol{\theta}_i$ parameter vector, which includes both the global and local components, and achieves sublinear regret.

However, it also has a weak spot: it is unable to leverage the underlying structure of the problem. The algorithm lets each bandit instance learn the global components independently, which especially hurts performance when data is scarce since it cannot use the feedback received by other agents. Of course, the magnitude of the loss depends on the contribution to the reward of the local components as opposed to the global ones.

Ultimately, we chose IND-LinUCB as a baseline throughout the experiments, since it fits best the problem setting. Namely, it captures the linear structure of generating the reward as the scalar product between an unknown parameter vector and the arms feature vector, and it is also able to learn a local parameter vector for each context. Furthermore, it is the standard baseline used in the literature on the aggregation of linear bandits. On the other hand, using a single bandit for all contexts performs quite poorly on this problem, as it lacks the locality property and therefore fails to capture most of the structure of the problem.

6.2. Experiment parameters

There are several parameters and hyperparameters, that influence the results. Most parameters are kept constant across experiments, while other are changed from the standard default to show their effect on the outcome and derive some conclusions. Below we present a detailed description of the main parameters:

- **Dimensionality**
  We fixed the arms dimensionality $d = 5$, since the number of arms scales exponentially with it, as explained below, and therefore larger values of d would pose computational challenges. Bear in mind that both the theoretical framework and implementation allow to choose of a different arm dimension for each context's agent. For the sake of simplicity, this property is not used in the following experiments, therefore we assume $d_i = d \quad \forall i$.

- **Arm set**
  To choose the arms for our multi-armed bandit, we use the vertices of a hypercube

in d dimensions, where each vertex has a size of `max_a`. In other words, the arm set consists of all possible permutations of the d components that can be either +`max_a` or −`max_a`, resulting in a total of 2^d arms. This choice is motivated by the fact that the vertices of the hypercube are the only possible candidates for the optimal solution to the bandit optimization problem in continuous arm spaces, making them a natural choice for our arm set.

- **Arm norm**
  The arm norm is solely determined by the number of arms and dimensions. From the definition of norm and the fact that all components are equal in absolute value, it follows that the arm norm is calculated as `max_a`$\cdot\sqrt{d}$. Note that after the aggregation the both the arm and theta norms are reduced, since the dimensionality of the arm and thetas shrinks.

- **Thetas**
  These are the main parameters used by the environment to generate the rewards, and are obviously unknown to the learner, which is trying to estimate them. The global parameter $\boldsymbol{\theta}_g$ is uniformly sampled from an interval ranging from +`max_theta_g` to −`max_theta_g`. Likewise the local parameter $\boldsymbol{\theta}_l$ is sampled from +`max_theta_l` to −`max_theta_l`.

- **Theta norm**
  The same as for the arm norm applies here.

- **Number of contexts**
  We use $N = 11$ contexts in all experiments.

- **Context distribution**
  This parameter regulates the distribution from which contexts are sampled. The default is a deterministic round-robin policy, and the alternative is a long tail discrete distribution, where there is one popular context extracted with probability p, while all the others share the remaining probability uniformly, i.e., each has probability $p/(N-1)$ of being sampled.

- **Threshold**
  From experimental evaluations, we choose $\varepsilon = 0.1$ as the default threshold, except in the threshold experiments, where we study the regret for different values of ε.

- **Split dimension**
  We set $k = 2$. Therefore after the aggregation the global arms will have size $k = 2$ and local arms $d - k = 3$.

- **Noise**
  For σ-subgaussian noise we use a zero-mean Gaussian with $\sigma = 1$.
- **Regularization**
  The λ regularization parameter of the ridge regression is set to 1, as it's not relevant to the setting.
- **Epochs**
  We run each experiment 10 times and then plot the mean of the monitored metric, i.e.,cumulative regret, as a solid line, and a colored area surrounding it, that visualizes the standard deviation.

Table 6.1 below gives an overview of the relevant parameters that we tune in the experiments to follow in order to demonstrate some properties of our algorithm. The keyword `exp` is used to denote the parameter that is the focus of the experiment, and for which we try different values to assess its effect on the regret. All the unmentioned parameters are kept constant with the values reported in the above section.

Experiment	$\|\boldsymbol{a}\|$	$\|\boldsymbol{\theta}\|$	$\|\boldsymbol{\theta}_l\|$	**Sampling**	ε
Threshold	2.2	2.2	1.73	round robin	`exp`
Long Tail	2.2	2.2	1.73	`exp`	0.1

Table 6.1: Experiments' parameters.

6.3. Experiments

6.3.1. Threshold

Goal

We start by comparing our algorithm with IND-LinUCB in a standard setting. In this experiment, we aim to evaluate the performance of our algorithm in a setting as plain as possible, which does not take advantage of the strengths of our algorithm. In fact, In fact, here we seek to empirically demonstrate that for a reasonable choice of the threshold ε, our algorithm's regret is upper bounded by IND-LinUCB, meaning that for a small enough value of ε, P-LinUCB has the same performance as IND-LinUCB in the worst case. This is the case in which the algorithm does not find a good enough $\boldsymbol{\theta}^i$ to aggregate the bandit instances and therefore performs the same exact actions of IND-LINUCB. Instead, in most cases, our algorithm has a gain in performance, whose magnitude depends on some specific environment properties explored in the following experiments. This experiment shows that there is virtually no downside in using P-LinUCB as opposed to IND-LinUCB, with at least a moderate gain in performance in most cases and a significant gain in a few quite common scenarios.

Setting

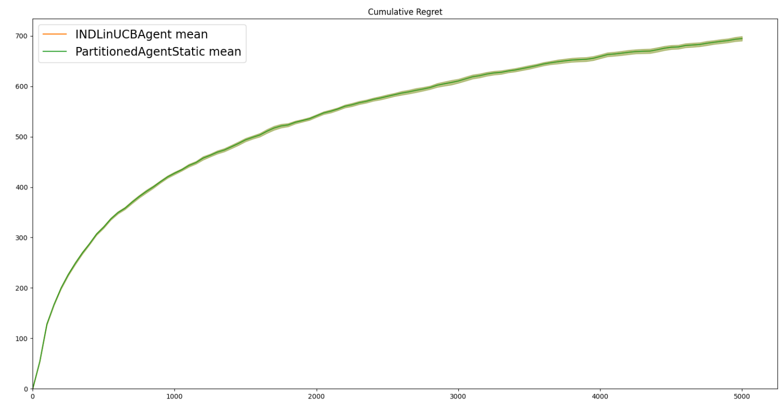
This experiment uses the standard parameters reported in the beginning of the chapter and vary the values of the hyper-parameter ε, for which we test three configurations, two extremes to show the edge cases, and a reasonable choice.

Results

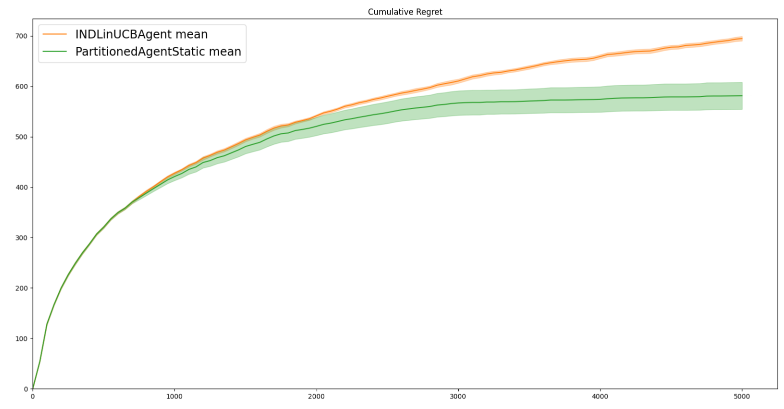
As expected, we can observe that the extreme cases perform poorly, while for an appropriate choice of the threshold, P-LinUCB yields a good result when compared to the baseline algorithm. Figure 6.1a shows P-LinUCB falling back to IND-LinUCB when a good estimate of $\boldsymbol{\theta}$ is not found, where the definition of good is solely dictated by the threshold ε on the moving MAPE. We artificially achieve to avoid the aggregation by setting an aggressively low allowed error. In Figure 6.1c we show the opposite extreme, where the threshold is too large, which inevitably leads to very bad results, since the global bandit parameter $\boldsymbol{\theta}_g$ is fixed before giving it enough time to be accurately learned. In fact this is the case we want to absolutely avoid, since it underperforms compared to IND-LinUCB. The large variance is explained by the fact that $\boldsymbol{\theta}_g$ is chosen with very low confidence, and is almost random, therefore in some epochs it might be much better or much worse than in others. The reasonable-valued scenario in Figure 6.1b performs quite well, even though this very plain setting that does not take advantage of the strengths of the algorithm. This experiment is an empirical demonstration that for a reasonable choice of the threshold P-LinUCB will perform at least as well as IND-LinUCB. Table 6.2 breaks down the cumulative regrets for the two algorithms, with their own standard deviations in parenthesis, and the percentage difference in regret, for the three values of ε.

ε	**IND-LinUCB**	**P-LinUCB**	**Δ-regret**
0.01	694 (14)	694 (14)	0%
0.1	694 (14)	**581** (84)	-16.28%
100000	**694** (14)	1254 (2871)	80.69%

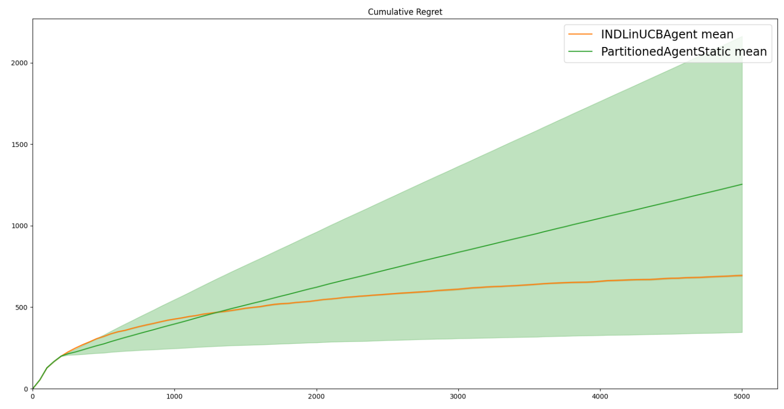
Table 6.2: Regret for different values of ε.



(a) $\varepsilon = 0.01$



(b) $\varepsilon = 0.1$



(c) $\varepsilon = 100000$

Figure 6.1: Regret for different values of the threshold ε.

6.3.2. Long Tail Context Distribution

Goal

One of the primary benefits of our algorithm is its ability to learn the global parameter $\boldsymbol{\theta}_g$ at the same rate as the top-performing agent, which we will call *Leader*. The stronger is the *Leader* compared to other agents the bigger are the gains in performance with respect to to a strategy that does not leverage cross-agent structure, such as our baseline. A strong *Leader* is one that learns its parameter vector much faster than other agents, since it will satisfy the splitting criterion sooner, and perform the split, which will benefit all agents, by reducing the size of the remaining piece of the parameters to learn $\boldsymbol{\theta}_l^i$.

One scenario that induces the creation of a strong *Leader* is the long-tail economic model [3], which refers to the phenomenon where the vast majority of product sell in relatively small quantities, making up the tail-end of a distribution curve, shown in yellow in Figure 6.2, in contrast to a few popular products that sell in large quantities and dominate the head of the curve, corresponding to the green area. It is evident that in this setting the most popular product will collect much more data, since it will be sampled much more often than the others and will be able to explore more, thus reducing its confidence interval and improving the estimate of its parameter much faster.

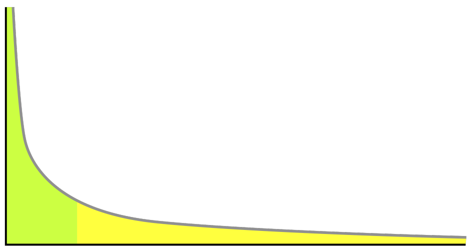


Figure 6.2: Long tail distribution.

Setting

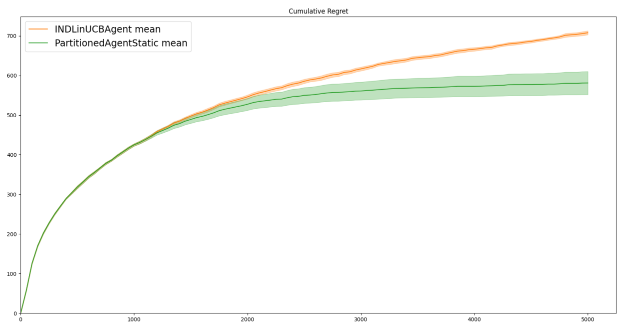
We perform four runs using a different context sampling distribution for each. In one run we use the uniform distribution to serve as comparison, and in the remaining three we simulate a long-tail distribution by sampling one context p times, while the rest of the context share the remaining probability uniformly, as explained in depth in Section 6.2. The three long-tail experiments are run with $p = 0.2$, $p = 0.5$, $p = 0.9$. Note that the uniform distribution is equivalent to $p = \frac{1}{N} = \frac{1}{11} = 0.\overline{09}$.

Results

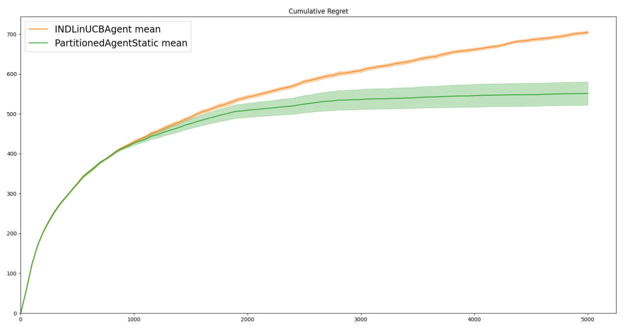
We can clearly see that the uniform distribution in Figure 6.3a is at a great disadvantage compared to the rest, and that the more the distribution is unbalanced, the greater the gain in performance. In fact we observe in Figures 6.3b, 6.3c, 6.3d that the regret gets smaller for larger values of p. This follows from the fact that the *Leader* will be sampled increasingly frequently, and therefore achieves a good estimate of his parameter sooner, leading to an overall smaller cumulative regret. IND-LinUCB instead is not able to share the learned parameter across bandits, and only the *Leader* will benefit from the additional data, thus leading to a more modest improvement w.r.t. the uniform case. Table 6.3 reports the cumulative regrets in the four scenarios, and the associated standard deviations between parenthesis.

Distribution	**IND-LinUCB**	**P-LinUCB**	**Δ-regret**
Uniform	707 (15)	**581** (92)	-17.82%
$p = 0.2$	704 (12)	**551** (91)	-21.73%
$p = 0.5$	650 (11)	**405** (87)	-37.69%
$p = 0.9$	432 (9)	**231** (51)	-46.53%

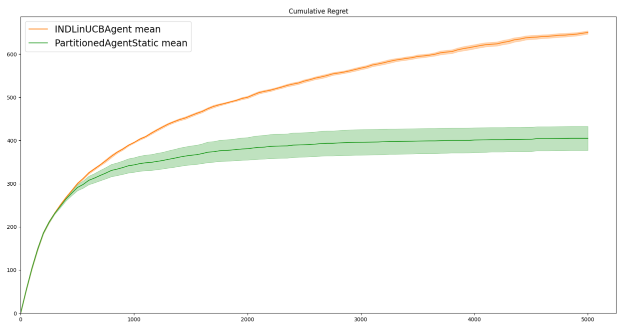
Table 6.3: Context distribution study.



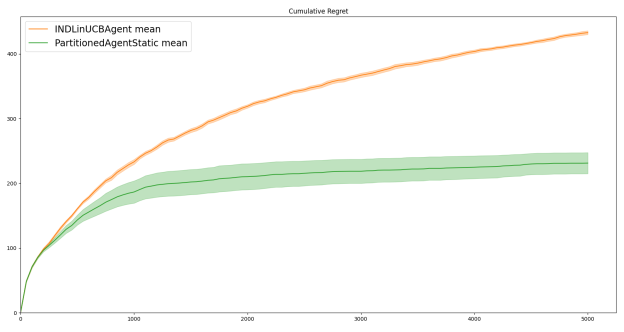
(a) Uniform distribution



(b) $p = 0.2$



(c) $p = 0.5$



(d) $p = 0.9$

Figure 6.3: Regret for different distributions of contexts.

7 | Conclusions and Future Works

7.1. Conclusions

This work addresses the Multi-Agent Stochastic Linear Bandit setting by introducing a structure to the interactions between agents. The study focuses on a new problem setting called the Partitioned Linear Setting, in which multiple agents make decisions, each with its own set of actions, and receive rewards based on their actions and a parameter vector. The parameter vector has two components: a global component shared by all agents and a local component unique to each agent. The global component depends only on the first k components of the parameter vector, while the remaining $d - k$ components constitute the local component. The goal of the agents is to maximize their collective reward over time by balancing the exploration-exploitation trade-off and collaborating effectively.

We develop a novel algorithm, called Partitioned LinUCB, designed to solve the Partitioned Linear Setting by taking advantage of the structure of the problem. The algorithm makes use of the assumption that the bandit parameter consists of shared and local components, and exploits this information to learn the global component from the most accurate bandit instance, and then shares it across all bandits when the estimate is accurate, while it learns the local components separately for each instance. Experimental results show that Partitioned LinUCB outperforms the naive solution of running an independent linear bandit per context, and performs at least as well as the naive solution, with a moderate gain in most cases and a significant gain in common scenarios such as the long-tail distribution of contexts. The main contribution of this work is the idea of using reliable estimates of the global parameter to speed up the learning process, which offers a new perspective on solving the partitioned linear bandit problem and contributes to the development of more efficient and effective algorithms for similar problems.

7.2. Future Works

We propose potential improvements to the Partitioned LinUCB algorithm to enhance its performance and robustness. While we have provided a qualitative analysis of the algorithm's regret, it would be beneficial to conduct a theoretical analysis to establish theoretical guarantees for the algorithm and derive an aggregation criterion and expression for the threshold based on problem-specific variables such as noise variance and arm norm.

Another potential development would be to update the shared component of the bandits. However, this would require allowing the *Leader* to change, as the best-performing bandit instance may vary over time.

The current algorithm formulation could accommodate the first extension by simply replacing the current aggregation criterion subroutine with a newly designed condition, whereas the second improvement would be more complex, particularly in providing its theoretical analysis.

Bibliography

[1] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011.

[2] R. Agrawal. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

[3] C. Anderson. *The long tail: Why the future of business is selling less of more*. Hachette UK, 2006.

[4] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[6] H. Bastani, D. Simchi-Levi, and R. Zhu. Meta dynamic pricing: Transfer learning across experiments. *Management Science*, 68(3):1865–1881, 2022.

[7] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[8] D. Bouneffouf, I. Rish, and C. Aggarwal. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[9] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear algorithms for online multitask classification. *The Journal of Machine Learning Research*, 11:2901–2934, 2010.

[10] L. Cella, A. Lazaric, and M. Pontil. Meta-learning with stochastic linear bandits. In *International Conference on Machine Learning*, pages 1360–1370. PMLR, 2020.

[11] N. Cesa-Bianchi, C. Gentile, and G. Zappella. A gang of bandits. *Advances in neural information processing systems*, 26, 2013.

[12] W. Chu, L. Li, L. Reyzin, and R. Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial*

Intelligence and Statistics, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

[13] V. Dani, T. P. Hayes, and S. M. Kakade. Stochastic linear optimization under bandit feedback. In *Annual Conference Computational Learning Theory*, 2008.

[14] C. Gentile, S. Li, and G. Zappella. Online clustering of bandits. In *International Conference on Machine Learning*, pages 757–765. PMLR, 2014.

[15] C. Gentile, S. Li, P. Kar, A. Karatzoglou, G. Zappella, and E. Etrue. On context-dependent clustering of bandits. In *International Conference on Machine Learning*, pages 1253–1262. PMLR, 2017.

[16] B. Kveton, M. Konobeev, M. Zaheer, C.-w. Hsu, M. Mladenov, C. Boutilier, and C. Szepesvari. Meta-thompson sampling. In *International Conference on Machine Learning*, pages 5884–5893. PMLR, 2021.

[17] T. L. Lai. Adaptive treatment allocation and the multi-armed bandit problem. *The annals of statistics*, pages 1091–1114, 1987.

[18] T. Lattimore and C. Szepesvári. *Bandit algorithms.* Cambridge University Press, 2020.

[19] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[20] S. Li, A. Karatzoglou, and C. Gentile. Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 539–548, 2016.

[21] S. Li, W. Chen, S. Li, and K.-S. Leung. Improved algorithm on online clustering of bandits. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2923–2929, 2019.

[22] M. Mussi, G. Genalti, A. Nuara, F. Trovò, M. Restelli, and N. Gatti. Dynamic pricing with volume discounts in online settings. *arXiv preprint arXiv:2211.09612*, 2022.

[23] M. Mussi, G. Genalti, F. Trovò, A. Nuara, N. Gatti, and M. Restelli. Pricing the long tail by explainable product aggregation and monotonic bandits. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3623–3633, 2022.

[24] A. Nuara, F. Trovò, N. Gatti, and M. Restelli. Online joint bid/daily budget optimization of internet advertising campaigns. *Artificial Intelligence*, 305:103663, 2022.

[25] S. Paladino, F. Trovo, M. Restelli, and N. Gatti. Unimodal thompson sampling for graph-structured arms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[26] A. Rangi and M. Franceschetti. Multi-armed bandit algorithms for crowdsourcing systems with online estimation of workers' ability. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1345–1352, 2018.

[27] H. E. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

[28] G. Romano, A. Agostini, F. Trovo, N. Gatti, and M. Restelli. Multi-armed bandit problem with temporally-partitioned rewards: When partial feedback counts. In *Proceedings of the International Joint Conference on Artiricial Intelligence*, 2022.

[29] A. Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[30] M. Soare, O. Alsharif, A. Lazaric, and J. Pineau. Multi-task linear bandits. In *NIPS2014 workshop on transfer and multi-task learning: theory meets practice*, 2014.

[31] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

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

[33] F. Trovò, S. Paladino, M. Restelli, and N. Gatti. Improving multi-armed bandit algorithms in online pricing settings. *International Journal of Approximate Reasoning*, 98:196–235, 2018.

[34] S. Vaswani, M. Schmidt, and L. Lakshmanan. Horde of bandits using gaussian markov random fields. In *Artificial Intelligence and Statistics*, pages 690–699. PMLR, 2017.

[35] J. White. *Bandit algorithms for website optimization.* " O'Reilly Media, Inc.", 2013.

[36] K. Xu and H. Bastani. Learning across bandits in high dimension via robust statistics. *arXiv preprint arXiv:2112.14233*, 2021.

[37] L. Zhou. A survey on contextual multi-armed bandits. *arXiv preprint arXiv:1508.03326*, 2015.

List of Figures

List of Tables

List of Symbols

Symbol	Description	Meaning
u	lowercase	scalar
$\boldsymbol{u}$	bold lowercase	column vector
V	uppercase	matrix
$\boldsymbol{u}^\top \boldsymbol{v}$		dot product between $\boldsymbol{u}$ and $\boldsymbol{v}$
$\|\|\boldsymbol{u}\|\|$	L_2 norm	$\sqrt{\boldsymbol{u}^\top \boldsymbol{u}}$
$\|\|\boldsymbol{u}\|\|_V$	norm w.r.t. matrix	$\sqrt{\boldsymbol{u}^\top V \boldsymbol{u}}$
I_d		identity matrix of size d

Ringraziamenti

A mio padre Gioacchino, a mia madre Marina e a mio fratello Ludovico, che mi hanno sostenuto in questo percorso, e mi hanno aiutato a diventare la persona che sono oggi.

Ai miei amici e compagni di studio, con cui ho condiviso i momenti di gioia e di tristezza in questi anni intensi e gratificanti.