University of Piraeus
Department of Digital Systems
MSc Digital Systems and Services
Specialization Big Data & Analytics

Online Learning Algorithms

with Application in Ranked Recommendations

Evgenia Panagiotopoulou *ME 1611*
Supervisor professor **Orestis Telelis**

TABLE OF CONTENTS

INDEX OF FIGURES AND TABLES

Figures

Tables

ACKNOWLEDGEMENT

I would like to thank my thesis advisor, Assistant Professor Orestis Telelis of the Department of Digital Systems at University of Piraeus. The door to his office was always open and there was always time in his schedule whenever I needed guidance about my research or writing. For all his support, patience and useful advice, I thank him.

Evgenia Panagiotopoulou

ABSTRACT

In this work we study the Online Learning and its application in ranked recommendations' systems that use context. Nowadays, modern platforms, websites and applications create an increased need for recommendations' systems that offer useful content suggestions. Online learning poses a great solution towards that purpose, as it can leave the customer – or user – satisfied, while requiring minimal computational resources, without demanding training or past data and with the ability to adapt quickly to new data. Furthermore, by introducing relevant context – side information – into an online learning recommendation system we can expect to produce content suggestions for the users that are appealing and tailored to their needs and interests.

Specifically, over the course of this study we explore bibliographically the Multiarmed Bandit problem (MAB), the Contextual Bandits, the Rankings of Recommendations and the corresponding algorithms. In order to delve deeper into the online learning recommendations, we design and conduct experiments with our own generated artificial datasets, using the algorithms that we found the most interesting. Our idea was to combine the recommendation meta-algorithms RBA and IBA with instances of the linear rewards contextual algorithm LinUCB. As it is, the two cases we are comparing are the RBA-LinUCB – a single-click, diverse-rankings algorithm that has been tested experimentally before – and the IBA-LinUCB, which is a multiple-clicks algorithm that is being tested for the first time in this work, to our knowledge.

In the results of our experiments it appears that the RBA-LinUCB has an increasingly better performance than the IBA-LinUCB, as an increase in the standard deviation of the arm rewards (SDR) of the MAB leads to a higher cumulative average regret by the IBA-LinUCB, while the RBA-LinUCB remains unaffected. Moving to another viewpoint, though, it appears that the IBA-LinUCB yields increasingly more clicks than RBA-LinUCB, as the average rate of rewards (ARR) of the arms increases. Finally, by monitoring the way the instances in the recommendation slots learn, it is revealed that the IBA-LinUCB slots learn much faster and more accurately than those of RBA-LinUCB. The above observations lead us to the fact that the IBA-LinUCB is expected to offer more substantial results and yield more clicks than the RBA-LinUCB, and thus constitutes a more effective solution when used in online contextual recommendation systems.

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία μελετάμε την περιοχή της «Άμεσης Εκμάθησης» και την εφαρμογή της σε συστήματα που παράγουν διατεταγμένες συστάσεις, κάνοντας χρήση επιπρόσθετης πληροφορίας. Σήμερα, οι σύγχρονες πλατφόρμες, ιστοσελίδες και εφαρμογές δημιουργούν την ανάγκη για συστήματα συστάσεων που προσφέρουν χρήσιμο περιεχόμενο για τον χρήστη. Η άμεση εκμάθηση προσφέρει μια ιδανική λύση προς αυτή την κατεύθυνση, καθώς μπορεί να ικανοποιήσει τον πελάτη – ή χρήστη – χωρίς να απαιτεί ακριβούς υπολογιστικούς πόρους, εκπαίδευση ή παρελθόντα δεδομένα και έχοντας τη δυνατότητα να προσαρμόζεται γρήγορα σε νέα δεδομένα. Επιπλέον, εισάγοντας παρακείμενη σχετική πληροφορία σε ένα σύστημα συστάσεων άμεσης εκμάθησης, μπορούμε να παραγάγουμε συστάσεις περιεχομένου, το οποίο είναι ελκυστικό και προσαρμοσμένο στις ανάγκες των χρηστών.

Συγκεκριμένα, κατά τη διάρκεια αυτής της μελέτης εξερευνούμε βιβλιογραφικά το «Πρόβλημα των Πολλαπλών Κουλοχέρηδων», τους «Κουλοχέρηδες Επιπρόσθετης Πληροφορίας», τις «Διατεταγμένες Συστάσεις» και τους αντίστοιχους αλγόριθμους. Με σκοπό να εμβαθύνουμε στις συστάσεις άμεσης εκμάθησης, σχεδιάζουμε και πραγματοποιούμε πειράματα με τεχνητά σύνολα δικιάς μας παραγωγής, χρησιμοποιώντας τους αλγορίθμους που μας φάνηκαν πιο ενδιαφέροντες. Η ιδέα μας ήταν να συνδυάσουμε τους μετα-αλγόριθμους συστάσεων RBA και IBA με στιγμιότυπα του LinUCB, ενός αλγορίθμου επιπρόσθετης πληροφορίας με γραμμικές ανταμοιβές. Συνεπώς, οι δύο περιπτώσεις που είχαμε να συγκρίνουμε είναι ο RBA-LinUCB – ένας αλγόριθμος μονού κλικ, διαφοροποιημένων συστάσεων που έχει δοκιμαστεί πειραματικά στο παρελθόν – και ο IBA-LinUCB, ο οποίος είναι ένας αλγόριθμος πολλαπλών κλικς που δοκιμάζεται για πρώτη φορά στην παρούσα εργασία, εξ ’όσων γνωρίζουμε.

Στα αποτελέσματα των πειραμάτων μας φαίνεται πως ο RBA-LinUCB έχει αυξανόμενα καλύτερη επίδοση από τον IBA-LinUCB, καθώς η αύξηση της τυπικής απόκλισης στις ανταμοιβές των χεριών οδηγεί σε αυξημένο σωρευτικό σφάλμα για τον IBA-LinUCB, ενώ ο RBA-LinUCB παραμένει ανεπηρέαστος. Από μια άλλη οπτική γωνία, όμως, φαίνεται πως ο IBA-LinUCB επιφέρει αυξανόμενα περισσότερα κλικς από ό,τι ο RBA-LinUCB, καθώς ο μέσος ρυθμός ανταμοιβών των χεριών αυξάνεται. Τέλος, παρακολουθώντας τον τρόπο με τον οποίο μαθαίνουν τα στιγμιότυπα των αλγορίθμων συστάσεων, αποκαλύπτεται πως τα στιγμιότυπα του IBA-LinUCB μαθαίνουν πολύ πιο γρήγορα και με μεγαλύτερη ακρίβεια από ό,τι αυτά του RBA-LinUCB. Οι παραπάνω παρατηρήσεις μας οδηγούν στο συμπέρασμα πως ο IBA-LinUCB αναμένεται να προσφέρει πιο ουσιαστικά αποτελέσματα και να επιφέρει περισσότερα κλικς από ό,τι ο RBA-LinUCB και άρα αποτελεί μια πιο αποτελεσματική λύση, όταν χρησιμοποιείται σε συστήματα συστάσεων άμεσης εκμάθησης με χρήση επιπρόσθετης πληροφορίας.

1 INTRODUCTION

In this work we discuss topics related to learning and we focus on the online contextual learning and its application in ranked recommendations. Specifically, we give an overview of an area in learning, which is called "Online Learning". We study two formalized problems of the kind and the most known algorithms that treat these problems. We examine each learning model and the way each algorithm learns, which depends on the various reward systems and the learning environments. In addition, we study the role of online learning in recommendations and rankings of recommendations, and evaluate its applicability and the value it can bring in real-world systems. Lastly, we design and conduct experiments on the contextual ranked recommendations with the algorithms that we found the most interesting. We compare their performance and their learning style and draw useful conclusions.

1.1 ONLINE LEARNING

In this section we are going to give the reader a brief introduction in the area of *Online Learning*. The Online Learning is a kind of reinforcement learning [1], where the learning occurs from interaction and the feedback of actions. In such a setting, the problems do not require any statistical and distributional assumptions; instead we are called to make a sequence of decisions and predictions. A setting like that has many applications in game theory and optimization [2].

Prediction of individual sequences has been a main topic of research in the theory of machine learning, specifically in the area of online learning. The need for individual sequence prediction came from various problems, such as games, compressing data, gambling and investment [3]. It is not easy to trace back the first appearance of such a study of this problem, but certainly, Blackwell, Hannan, Robbins and others in the 50's were pioneers in the field [3].

It is important to understand two things about the online learning setting; first of all, the requests for actions come in a stream – a sequence – and secondly, there is no training phase. In online learning the algorithms do not learn by training and testing, as it happens traditionally in supervised learning. Instead, in each step they receive an input, decide an action and obtain the payoff of this action. In a way, the learning happens sequentially, one bit at a time, as this process is repeated over and over again. For the above reasons, the algorithms of online learning are great in real-world large-scale applications, as they do not require any batch processing – they rather process one sample at a time.

In the online learning setting, the *learner* – otherwise called *agent* – is trying out different actions out of the available ones. In every *time step*, or *trial*, the learner selects an action and the *environment* – or *nature* – provides the learner with a *reward*. The reward for the action selected can be either positive reinforcement, or negative feedback, depending on the setting. The learner, thus, learns which choices yield better results.

Unlike static supervised learning, where the learner is in need for past data, in order to devise a *strategy*, the online learning algorithms devise their strategy one step at a time. They do not

need any past recollection, batch data, or any distributional assumptions, but they take into consideration the given reward and immediately re-assess the strategy, according to which they decide on the next action.

Online learning comes in handy in the following cases:

- If it is expensive to train the algorithm with past data
- If there are no available training data
- If the data get cold and outdated quickly (i.e. in a news website).

The problems in online learning usually attempt to achieve *dynamic allocation* of the available resources. The algorithms used for such problems seek to find an effective allocation strategy, or *policy*. The performance of the algorithms and the actions taken are measured through the *loss* suffered, or the *gain* obtained by the allocation policy used. The ultimate goal is the minimization of *regret*, which is the comparison between the cumulative loss/gain of the used strategy and that of the best strategy.

As it is, an online learning algorithm can work perfectly fine with streaming data, usually generated as a function over time. These kinds of algorithms are capable of handling data that change pattern often and can quickly re-adapt their strategy. This means that the points should not necessarily be *i.i.d.*[1], or following a fixed distribution, which makes online learning applicable in numerous real-world settings.

One popular application of the online learning is in the recommendations, or the rankings of recommendations, where the used algorithm is called to make an item suggestion, or a list of item suggestions to the user, or client. For example, the online learning algorithms could be used in a news website in order to produce a list of suggested news articles to the users. The decision for the suggestions could be made based on the popularity of the articles, so that the suggestions are appealing to the majority of the users. On the other hand, the decision can be made by taking into consideration the context – side information to be used along with the online feedback – such as the age of the user, gender, country of origin and more. This can contribute to a drastic change in the quality of our recommendations, as it can offer suggestions tailored to the interests of the users.

Apart from the use or no use of context, one other parameter of the recommendations can be the number of suggestions – or clicks in the case of a news website – that the customer, or user, is allowed to. There are recommendation algorithms that accept multiple clicks from the user, while others are considered single-click algorithms. In this work, we compare experimentally the efficiency and performance of algorithms from both categories.

In the next sections and chapters, we are going to introduce some online learning problems and algorithms and we are going to maintain our focus on the application of online learning in ranked recommendations, with the use of context.

[1] i.i.d.: independent and identically distributed

1.2 Online Learning Problems

In this section we are going to present two formalized problems in the area of Online Learning; first, the problem of "Prediction with Expert Advice" and, secondly, "The Mutliarmed Bandit" problem. These two problems have been studied both stochastically and non-stochastically.

To introduce ourselves into the "Prediction with Expert Advice" problem, let us contemplate the following scenario [4].

Each morning, a farmer needs to predict if it will rain, or not. For this task, he seeks the advice of some weather predicting websites and, based on them, he makes his own prediction. At the end of the day, he knows the real outcome of the daily weather. He then evaluates how truthful the predictions of the websites were, so that he knows which of them he can trust or not and he can make more accurate predictions in the future.

The problem described above is called *Prediction with Expert Advice* and it was introduced in Online Learning around 1990 in various works [3]. The algorithms that predict individual sequences with expert advice – as it is usual with the online learning algorithms – pose an efficient option for large-scale applications, because of their inexpensive processing needs and simplicity.

The problem of prediction with expert advice involves a *forecaster* – in the example above the farmer – who must predict a sequence of T elements, one-by-one. It also implicates N advisors, or *experts* – in the above example the weather predicting websites – who give their individual predictions to the forecaster about the next outcome of the sequence. Based on them, the forecaster makes his own prediction and receives the true result, so that he can evaluate his estimation and that of his experts.

This problem can be met in scenarios with both distinct and continuous sequence outcomes. Furthermore, it should be noted that we are in the adversarial scenario: this means that the elements of the sequence that we must predict are not independent and identically distributed, but instead they can be random draws from a bounded interval $[a, b]$ with $a < b$.

Further to the above setting, we can note that this problem corresponds to the *full information game*, according to the definition by Auer, Cesa-Bianchi, Freund, and Schapire, in 1998 [5]. And that is because in each time-step t the forecaster has access to the opinions of all the experts and, by receiving the outcome y_t, he gets to know the loss of every expert for round t.

Some of the most well-known algorithms that treat the Prediction with Expert advice problem are The Halving Algorithm (N. Littlestone, 1987) [6], The Weighted Majority Average (N. Littlestone and M. K. Warmuth, 1989) [7], The Randomized Weighted Majority Algorithm [7] and The Exponential Weighted Average Algorithm [7].

Now, in order to understand the Multiarmed Bandit problem, let us consider the following scenario.

A gambler in a casino has K available slot-machines in front of him to play. He does not know which machines give better payoffs than others, or which is the optimal machine profit-wise. In his possession he has T coins, which he will spend, one by one, in the slot-machines of his choice. For each coin:

- he chooses a machine out of the K available ones, enters the coin and pulls the lever
- he receives the payoff of the machine.

His objective is clear – he must cumulatively collect as big a profit as possible.

This kind of setting is a problem of *prediction with limited feedback*, or a *partial information game*, according to [8]. That is to say, the gambler in each round chooses an action and the only feedback he gets is the consequence of this action: the reward corresponding to arm i for time-step t. It is important to note that the gambler does not see the reward of any other arm, except for the one he chose to play in each time-step t. It is for this reason that the feedback is considered to be limited, in contrast to the "Prediction with expert advice" problem, discussed previously. This detail in the problem formulation, along with the lack of expert advice, are the two key factors that set the two problems apart.

The MAB problem can be met in settings where the actions of the player yield either losses [9], or gains [8]. This fact, along with its general and simplistic nature, makes the multiarmed bandit problem and its variations useful in numerous applications, such as auctions [10], pricing [11], recommendations [12], [13], [14], clinical trials [15], adaptive routing in networks [16], advertising [17] and more.

The following examples illustrate some possible applications of bandits:

- *Clinical trials*: A pharmaceutical company might use the multiarmed bandit model to test different drugs for the treatment of a disease. In this setting, the K drugs would correspond to the K arms and the payoffs would be the results of the drugs in the health of the patients.
- *Marketing*: A marketing company might apply a multiarmed bandit model to test possible ads, or possible ad placements, to test which ones are the most effective. In this setting, the K ad alternatives represent the K arms and the payoff might be the response of the customer, e.g. clicking on the online ad, or buying the product etc.
- *Pricing*: A retailer can test different price points for its new product with the help of the multiarmed bandit model. The K price points, which correspond to the K arms, can be put to test to check the reaction of the customers. In this case, the payoff of each arm could be the buying, or no buying of the product by the customer.

The algorithms UCB1 (P. Auer, N. Cesa-Bianchi and P. Fischer, 2002) [18], e-Greedy (R. S. Sutton and A. G. Barto, 1998) [1] and EXP3 (P. Auer, N. Cesa-Bianchi, Y. Freund and R. Shapire) [19] are some of the most known algorithms for the Multiarmed Bandit problem. We will present these algorithms in detail in Chapter 2. For more Reinforcement Learning problems and algorithms, we refer the reader to the work of R. S. Sutton, A. G. Barto [1].

1.3 OVERVIEW

In this section we give an overview of our contributions, that can serve as a map of our work. After this general introduction into the Online Learning subject and its two famous problems, the Prediction with Expert Advice problem and The Multiarmed Bandit problem, we proceed in Chapter 2, where we discuss in detail the Multiarmed Bandit problem – in its stochastic and non-stochastic version – and the respective algorithms in length. In Chapter 3, we study the Contextual Bandits, that is bandits that make decisions based on the available context, or side information. In Chapter 4 we engage ourselves with Ranked Recommendations settings and we examine algorithms that produce diverse rankings, or accept multiple clicks. Finally, in Chapter 5 we design some experiments regarding the area of ranked recommendations with the use of context, we generate artificial datasets and compare the performance and the learning style of some interesting algorithms.

2 THE MULTIARMED BANDIT PROBLEM

In the previous chapter we gave the reader an introduction into the Multiarmed Bandit problem, using the example of the gambler and the slot machines. This scenario describes perfectly the *Multiarmed Bandit problem (MAB)* and, therefore, lent its name to it. In the past, the slot-machines were often called "*one-armed bandits*", thus the setting with the K slot-machines was named "*K-armed bandit*", or "*Multiarmed Bandit*" (Figure 2.1). In general, the K arms or machines, represent K available actions, the reward generations of which are unknown and are assumed to be different to each other in general.

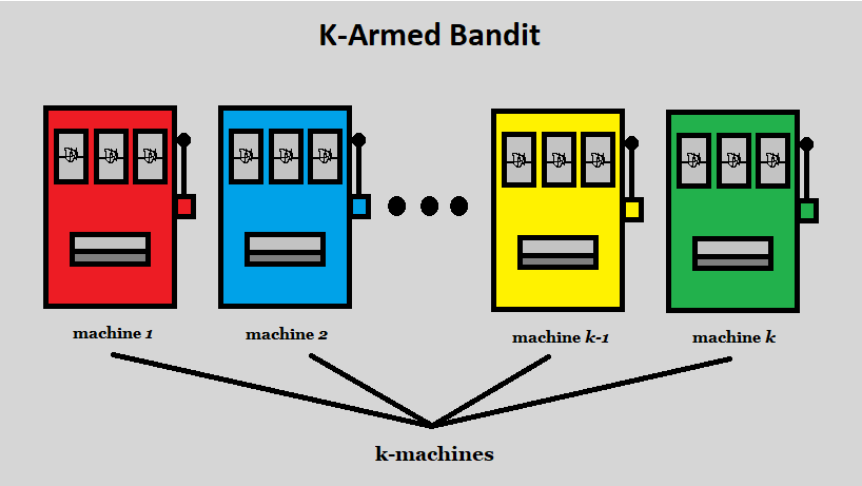

Figure 2.1. The Multiarmed Bandit model with K slot-machines

This sequential decision-making problem was originally formulated by Herbert Robbins in 1952 [20]. In this work, Robbins wrote for the problem: "[…] *Note that there is no terminal decision to make... The whole problem lies in deciding how to draw the sample* […]". This captures the core of the problem: we are not interested in estimating anything after we finish playing the machines, we merely intend to devise a rule – a strategy – according to which we should draw machines to play in each round.

In the heart of the multiarmed bandit problem lies the *exploration versus exploitation* dilemma. In the example of the gamble, we can imagine that the gambler would naturally want to keep playing a machine that has brought good results so far. This is the part of the exploitation and it is essential for the increase of the gambler's profits. On the other hand, if the gambler does not explore his options enough, he might not get a good sample of the machines' rewards and he might fail to discover a truly advantageous machine. The exploration and exploitation parts are not two separate phases to be implemented one after another in a gambler's strategy – although they could. Instead, they must be balanced effectively throughout the strategy. As it is expected, the more you exploit, the less you get to explore – and vice versa – and that is why this situation is often described as the *exploration-exploitation tradeoff.*

In the following paragraphs we will discuss the multiarmed bandit problem in more detail. Furthermore, there will be definitions of the terms and notations that will be used in the next sections. The presentation is based on [19], [18], [3], [21].

For the setting of this problem, we consider that the *forecaster*, otherwise called *learner*, *player*, or *gambler*, is repeatedly faced with K options: there are $K \in \mathbb{N}^*$ available *machines*, which are also called *arms*, or *actions*. The forecaster must explore and exploit those options, in order to get profitable results over time. Each *time-step*, or *trial*, or *round*, is indexed by t. The time horizon of this game will be noted as $T \in \mathbb{N}^*$, thus $t \in \{1, \dots, T\}$.

THE MULTIARMED BANDIT PROBLEM

For each trial t, for $t \in \{1, \dots, T\}$:

- the forecaster chooses an action $i \in \{1, \dots, K\}$
- the environment receives the choice of action and gives back to the forecaster the $r_i(t)$, which is the reward of the action i for trial t

Figure 2.2. Definition of the Multiarmed Bandit problem

The *environment*, or *nature* receives the choice of the forecaster and returns to him the *payoff*, or *reward* $r_i(t)$ that corresponds to machine $i \in \{1, \dots, K\}$ for trial t. The Figure 2.2 gives an illustration of the exact steps of the problem.

The MAB problem has been formulated and studied in both *stochastic* and *non-stochastic* ways. There has also been the *Markovian* approach, which we will not discuss in this work, however for an overview we refer the reader to Chapter 3 of [1]. The variations in the problem formulation originate from the mathematical assumptions that we make about the process of reward generation. The presentation of the stochastic and the non-stochastic problems are based on [19], [18], [21], [8].

However, before we can even talk about the stochastic and non-stochastic distinction, we have to establish some concepts and notions that will be shortly needed:

Definition 1. A *strategy*, or *policy,* usually noted as *A* or *B,* is a sequence of action choices noted as $(i_1, i_2, \dots, i_T)$, or $(j_1, j_2, \dots, j_T)$.

Definition 2. The *cumulative reward* of the forecaster, by following a strategy *A* of actions $(i_1, i_2, \dots, i_T)$, is:

$$G_A = \sum_{t=1}^{T} r_{i_t}(t)$$

Definition 3. The gain – or loss – that occurs when the forecaster follows a strategy *A*, instead of a strategy *B*, is measured through the *worst-case regret*:

$$G_A(T) - G_B(T)$$

In the bandit problem, we often use the notion of the *optimal machine*. This is the machine that has the *maximum expected reward*, which of course we can never know beforehand. The expected reward of a machine exists only as a theoretical concept, as the player could never accurately estimate it, unless he played only a certain machine for every $t \in \{1, \dots, T\}$.

Definition 4. In the case that one of the two strategies is the optimal, meaning that it is the consistent playing of the optimal machine:

$$G_A(T) = G_{max}(T) = max_{1 \le i \le K} \left(\sum_{t=1}^{T} r_i(t) \right),$$

then the difference $G_A(T) - G_B(T) = G_{max}(T) - G_B(T)$ is called *weak regret*.

2.1 THE STOCHASTIC BANDIT PROBLEM

The *stochastic bandit problem* was primarily analyzed in the work of Lai and Robbins in 1985 [22]. In this setting, the assumption is that the rewards of each machine are independent and identically distributed (*i.i.d.*). That is to say, the rewards of the K machines come from unknown distributions $P_1, \dots, P_K$, with means $\mu_1, \dots, \mu_K$, respectively.

In other words, in every time-step the environment receives the chosen action of the forecaster, i_t, and draws a reward to return, according to the distribution P_{i_t}. Note that every draw of reward by the environment is independent from the past actions of the forecaster. The steps of the problem are illustrated in Figure 2.3.

THE STOCHASTIC BANDIT PROBLEM

For each trial t, for $t \in \{1, \dots, T\}$:

- the forecaster chooses an action $i \in \{1, \dots, K\}$
- the environment draws independently and returns to the forecaster the $r_i(t) \sim P_i$, which is the reward of the action i for trial t, according to the distribution P_i

Figure 2.3. The stochastic bandit problem, as given in Bubeck and Cesa-Bianchi, 2012 [21]

In the stochastic scenario we use the notion of the *pseudo-regret*:

Definition 5. If i_t is the choice of the forecaster for time-step t, then we define as *pseudo-regret* the following:

$$\bar{R}_T = max_{i=\{1,\dots,K\}} E\left[\sum_{t=1}^{T} r_i(t) - \sum_{t=1}^{T} r_{i_t}(t)\right] = T\mu^* - E\left[\sum_{t=1}^{T} \mu_{i_t}\right],$$

as $\mu^* = max_{1\le i\le K}\, \mu_i$.

The measure of pseudo-regret is the most suitable for the stochastic bandit problem, as it can be bounded logarithmically. In addition, an algorithm that appears to work very well for the stochastic bandit model is the *Upper Confidence Bound* algorithm, introduced by Auer et al. in 2002 [18].

2.1.1 The UCB1 Algorithm

UCB1 is an algorithm fit for the stochastic bandit model and its name is an abbreviation for *"Upper Confidence Bound"*. In 2002, Auer, Cesa-Bianchi and Fischer [18] introduced some probabilistic policies, among which the deterministic UCB1.

Earlier in 1985, Lai and Robbins were the first ones to introduce the upper confidence index [22], a measure that served as an estimation of the expected reward for each machine. This index was very hard to calculate, as it relied on the complete sequence of rewards of each arm. In 1995 [23], R. Agrawal came up with a family of strategies that rely only on the sum of rewards of each machine for the computation of the index, which makes it a much easier computation than that of Lai and Robbins'. The UCB1, introduced in [18], is a variant of Agrawal's (1995) index-based policy and it appears to be the best fitting algorithm for the stochastic model.

In UCB1 the index for each machine i is the sum of two parts:

- the mean reward $\bar{r}_i$ of the machine i until time-step t and
- $\sqrt{\frac{2\ln(t-1)}{t_i}}$, which is the one-sided confidence interval for the average reward of i.

It should be noted that UCB1 works with the K machines having arbitrary distributions $P_1, P_2, \dots, P_K$, with support in [0,1]. The pseudocode of the algorithm is fully displayed in Figure 2.4 below, based on the presentation of [18].

ALGORITHM UCB1

Initialize the algorithm by playing each machine once.

Then for every time-step t:

- play the machine i that maximizes $\left(\bar{r}_i(t-1) + \sqrt{\frac{2\ln(t-1)}{t_i}}\right)$,
  where t_i is the number of times the machine i has been played and
  $\bar{r}_i(t-1)$ is the mean reward of machine i so far
- receive reward $r_i(t)$ and update t_i and $\bar{r}_i$(t)

Figure 2.4. Pseudocode for algorithm UCB1 [18]

As seen in the figure with the algorithm's pseudocode, for the initialization of UCB1, all K arms must be played once. After that, for each round t, we calculate the upper confidence bound for each machine i:

$$\left(\bar{r}_i(t-1) + \sqrt{\frac{2\ln(t-1)}{t_i}}\right)$$

and we play the machine with the highest UCB.

If one were to observe the quantity $\left(\bar{r}_i(t-1) + \sqrt{\frac{2\ln(t-1)}{t_i}}\right)$ and its behavior over time t, they would notice that for machine i:

- the confidence bound grows with the number of rounds,
- but it gets smaller as we play the machine.

On the one hand, this means that, as time t goes by, the algorithm does not set in its ways, but instead it invites exploration. On the other hand, it seems that we get more certain about machine i, as we play it.

In *Theorem 1* of [18], it is proved that UCB1 achieves logarithmic regret uniformly over time.

2.1.2 The *e*-Greedy Algorithm

e-Greedy is also an algorithm addressing the stochastic multiarmed bandit problem. It was introduced in 1998 by Sutton and Barto [1] and it is the most simplistic one out of the algorithms discussed in this chapter. This algorithm balances the exploration versus exploitation dilemma by applying a fixed ratio of exploration $e \in [0,1]$. One can easily implement the e-Greedy by following the pseudocode of the algorithm, as seen in Figure 2.5.

In each trial $t \in \{1, \dots, T\}$, the algorithm explores or exploits, according to the probabilities e and $1 - e$, respectively. To explore, the forecaster chooses uniformly at random an arm to play out of the K available ones. To exploit, the forecaster plays the arm that has the highest mean reward $\bar{r}_i$.

ALGORITHM *e*-GREEDY

Input parameter $e \in [0,1]$

For each time-step t:

- with probability e choose an arm $i \in \{1, \dots, K\}$ uniformly at random and play
- with probability $1 - e$ play the arm i that has the highest mean reward so far, $\bar{r}_i(t-1)$

Figure 2.5. Pseudocode for algorithm e-Greedy

In their work in 2002, Auer et al. [18] gave a variant of the e-Greedy, the e_n-Greedy, where the parameter of exploration e converges to zero as $t \to T$.

2.2 THE ADVERSARIAL BANDIT PROBLEM

The formulation of the *non-stochastic*, or *adversarial bandit problem*, has originated from the *Game Theory* and it was not considered an instance of the multiarmed bandit problem [21], until the work of Auer et al.,2002 [19] linked it to the stochastic bandit problem. In this game, the player is playing the K machines against an *opponent*, or an *adversary*. In every round, while the player chooses his next action, the adversary decides on the rewards of the K machines. The steps of the problem are illustrated in Figure 2.6.

In this version of the problem, there are no statistical assumptions on the generation of rewards. Instead, we consider the generation of rewards to be an arbitrary selection of payoffs in a real interval $[a, b]$, with $a < b$. In that context, the sequence of rewards can even be dependent on time.

In the adversarial setting, there is room for some more considerations, such as that of an *oblivious*, or *non-oblivious* adversary. A non-oblivious adversary, as opposed to an oblivious, is one that has the potential to adapt its strategy, according to the actions of the player. In this case, the adversary can devise a strategy, which depends on the player's past actions, and even maliciously set the rewards to be disadvantageous for the player.

With the scenario of a non-oblivious adversary in mind, two points are becoming apparent:

- The concept of the optimal machine could get invalidated. This means that if the adversary saw that the player has discovered and plays the optimal machine, he could adapt his strategy and worsen the rewards of this machine, possibly turning it into non-optimal anymore.
- The necessity of a randomized player strategy. A deterministic player would have no chance against a malicious non-oblivious adversary.

THE ADVERSARIAL BANDIT PROBLEM

For each round t, for $t \in \{1, \dots, T\}$:

- the player chooses an action $i \in \{1, \dots, K\}$
- the adversary decides on the rewards of the machines for round t
- the adversary receives the choice i of the player and returns to him the $r_i(t)$, which is the reward of the action i for trial t

Figure 2.6. The adversarial bandit problem, as given in Bubeck and Cesa-Bianchi, 2012 [21]

In the theorem 5.1 of [19], it is proved that there exists a set of K reward distributions with expected weak regret $\Omega(\sqrt{KT})$ for any time horizon T.

2.2.1 The EXP3 Algorithm

EXP3 is an algorithm that solves the adversarial bandit problem, introduced in 2002 by Auer, Cesa-Bianchi, Freund and Shapire [19]. The name EXP3 is an abbreviation for the full name of the algorithm: *"Exponential-weight algorithm for Exploration and Exploitation"*. As the name suggests, the EXP3 algorithm uses exponential weighting for the available actions and attempts to strategically balance the exploration and the exploitation.

The algorithm is a variant of the algorithm *Hedge* [9], which solves the full information game. The Hedge algorithm was a generalization of the *weighted majority algorithm*, originally proposed by Littlestone and Warmuth in 1989 [7].

EXP3 works with arbitrary rewards in the interval $[0,1]$ and in the following paragraphs we will assume that this is the case, but the rewards and theoretical guarantees can be expanded to any interval $[a, b]$, where $a < b$. The trick is to rescale the rewards into the $[0,1]$ interval. This trick can also be applicable to loss scenarios with negative rewards.

Note that in the adversarial scenario the algorithm does not need fixed distributions of rewards in order to work. In their theoretical analysis, Auer et al., [19] prove bounds for the regret that hold for any worst-case sequence of rewards. This means that the rewards of the K arms could even be dependent on the time variable t.

The above elements - the generalization in any interval $[a, b]$ and the lack of obligation for fixed distributions of rewards - make the algorithm EXP3 very convenient and applicable in numerous problems. In fact, the algorithm can be used in real-world problems, with scenarios of either gains or losses, and with rewards' distributions that change over time.

ALGORITHM EXP3

To initialize:

- Input parameter $\gamma \in (0,1]$
- Set initial weights $w_i(0)$ to 1, for $i = \{1, \dots, K\}$

Then for each time-step t:

- Calculate $p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}$, for $i = \{1, \dots, K\}$
- Choose an action a_t, according to the probabilities $p_1(t), \dots, p_K(t)$ and receive reward $r_{a_t}(t)$
- Calculate $\begin{cases} \hat{r}_i(t) = r_i(t)/p_i(t), \ for\ i = a_t \\ \hat{r}_i(t) = 0, for\ i \neq a_t \end{cases}$
- Set the weight $w_i(t+1) = w_i(t) \exp\left(\gamma \frac{\hat{r}_i(t)}{K}\right)$

Figure 2.7. Pseudocode for algorithm EXP3 [19]

A pseudocode description of the algorithm EXP3 appears in Figure 2.7, based on the presentation of the algorithm in [19]. What the algorithm basically does, is that it takes as input $\gamma \in (0,1]$ and sets the weights of the K arms to 1. After that – for each round t – an action is chosen, according to the probabilities $p_i(t)$ that depend on the parameter γ and the weights w_i. For that action we receive the payoff and adjust the weights for the next round $w_i(t-1)$.

If we were to look more closely at the algorithm, we would notice that the parameter γ regulates the randomness in the probability $p_i(t)$ formula and, thus, modifies the exploration-exploitation ratio. Specifically:

$$\gamma \to 0 \quad \Rightarrow \quad p_i(t) \to \frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)}\ , \quad for\ i = \{1, \dots, K\}$$

and

$$\gamma \to 1 \quad \Rightarrow \quad p_i(t) \to \frac{1}{K}\ , \quad for\ i = \{1, \dots, K\}.$$

The uniform part, γ/K, of the probabilities $p_i(t)$ promotes the exploration, which in turn increases the chances that the best action will be spotted and used for exploitation. On the other hand, the part

$$(1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)}$$

of the probabilities $p_i(t)$ is the basis of the exploitation of the best actions discovered so far.

In addition, the inverse-propensity weighting estimator

$$\hat{r}_i(t) = r_i(t)/p_i(t),$$

gives an extra boost to actions that have lower probability $p_i(t)$, meaning that they are more unlikely to be drawn and played. Giving an advantage to arms that were picked, despite their low odds, and gave an unexpectedly high reward, contributes to the exploration and might lead to the discovery of a hidden gem. The score of the inverse-propensity weighting estimator is used afterwards to adjust the exponential weight:

$$w_i(t+1) = w_i(t) \exp\left(\gamma \frac{\hat{r}_i(t)}{K}\right), \ \text{for}\ i = \{1, \dots, K\}.$$

The choice of the value of the parameter γ is a matter that needs special attention, as it defines how randomized the algorithm gets. In [19] there is a way to calculate the optimal value of the parameter γ. If the horizon $\mathrm{T} > 0$ is given, we can use the corollary 3.2. from [19] and apply the formula:

$$\gamma = \min\left\{1, \sqrt{\frac{KlnK}{(e-1)g}}\right\}, \qquad \text{where } g \geq G_{max}\ .$$

Calculating g is usually impossible, however, there is a trick that can be used, in the case that the horizon T is known and the distributions are in $[0,1]$. Instead of g, we can use the value of T as following:

$$g = T \cdot (reward_{max}) = T \cdot 1 = T.$$

The above formula gives us the optimal value of the parameter γ, which in turn makes:

$$G_{max} - E[G_{EXP3}] \leq 2.63\sqrt{gKlnK} \leq 2.63\sqrt{TKlnK}.$$

Algorithm	**Bound**	**Proposed use**
EXP3	Expected Cumulative Regret $O(\sqrt{TKlnK})$	For adversarial multiarmed bandits
e-greedy	Regret with high probability $O(\sqrt{TKlnN})$	Not computationally efficient for large N
UCB1	Expected Cumulative Regret $O(\sqrt{KTlnT})$	For stochastic multiarmed bandits

Table 2.1. Regrets bounds and proposed use of MAB algortihms

3 CONTEXTUAL BANDITS

One very interesting extension of the multiarmed bandits model is the *contextual bandits* model. The contextual model is very similar in structure to that of the multiarmed bandit, with the extra addition of some *side information*, or *context*. In numerous problems and applications, there is information available that can be used to enhance the decision-making of the bandits. This side information is exactly what the contextual bandits are targeting to utilize in order to make more informed decisions.

Contextual bandits go by many other names, such as "multiarmed bandits with expert advice" [24] [25], "associative reinforcement learning" [26], "multiarmed bandits with side information" [27] and more. In the 1990's the contextual bandit settings were studied in various works as "associative reinforcement learning", under assumptions about linearity in the reward functions.

The context may be conveyed in the form of features, which bear information about the environment at time t and/or the arms of the bandit. As an example setting, we can consider a recommender system that displays ads to users on a website and records their clicks. In this system, the knowledge of user information like the gender, the location, or the age could be indicative of the user's taste. This information, paired with the ad's specific category, or type, can boost the suggestions of a recommender system. To take advantage of the side information, we need to map the context to certain actions, meaning we need to map the context to the bandit arms.

The above mapping is modeled through the notion of *experts*. The forecaster has N experts, who provide him with advice about which arm he should play in every round t. The forecaster no longer decides himself about the arm pulls, but instead at each round t he consults an expert about which arm to pull. He has to explore and exploit his way to the discovery of the cumulatively most reward-generating expert. What the expert, essentially, represents is a *strategy*, or a *policy* – a mapping of contexts to actions.

The recommendation of each expert is otherwise called the *distribution* of the expert i. This is because the expert gives a distribution over the arms: for the sake of generality, we consider that he does not only predict the single most promising arm, but instead he gives a vector of probabilities, one for each arm. As expected, these probabilities should add up to 1. The recommendation for the K arms of each expert i, at time t, is:

$$e_i(t) = \left(e_{i,1}(t), \dots, e_{i,K}(t)\right)$$

While it is possible that the estimation of an expert is deterministic, this is considered as a special case of this problem. That is to say, if the expert were to become the decision maker for one round, in the general case he would choose an action at random, according to the probabilities he gave in his distribution, whereas in the case of the deterministic estimation he would recommend and play only one specific arm.

The expert advice in a system could either be external advice, or calculations of the forecaster. In general, there are two contextual bandit models that have been studied: the non-stochastic and the stochastic contextual bandit model, which will be analyzed in the following

paragraphs. These two models correspond to different expert models as well: in the case of the non-stochastic scenario, the logic behind the mapping of an expert is not necessarily known – it can be any arbitrary strategy. On the other hand, in the stochastic scenario, the values of the features are continuous and the reward of each arm is assumed to be linear – or approximately linear – to the features of the arm [28]. In such a case, the experts represent a linear function: each expert i, for each round t corresponds to a vector of coefficients for the features in the linear function.

Overall, in the contextual model we suppose that the experts' opinions are fixed and they do not change according to the rewards. The experts react only to the context of the round. It should be noted that in the contextual bandit setting we only get to observe the reward of the arm that was chosen to be played. This means that we have no idea about the rewards of the other arms. Usually, in a web setting the measure we use in order to track the rewards is the click, or no click of the user. This way we can quantify that a link, an article, or an item has a certain *Click-Through-Rate* (CTR).

While collaborative filtering is a traditional solution in recommendations, the contextual bandits pose a great alternative: they can take advantage of context, they learn fast and they do not require past data about the users, or the arms. In their majority, they are also easy to implement. Indeed, the contextual bandits have been proposed as a suitable solution in many recommendation problems in websites and applications, such as advertisements' placement, display of content to the user and more. In [14], Li, Chu, Langford and Schapire use a real-world dataset from the Yahoo! Front Page Today module in order to recommend news articles to users, according to the user and article features. They compare contextual bandits with context-free multiarmed bandits and they claim a 12,5% increase in clicks. The same dataset has been used by Beygelzimer, Langford and Li in [29] for experimentation with various contextual bandit algorithms, towards a more personalized user experience. In [24], McMahan and Streeter use contextual bandits to achieve effective ad placement in search queries.

We hereby give the notations and definitions that will be used in the next paragraphs. In the following paragraphs, we will index the experts by $i \in \{1, \dots, N\}$ and the arms by $a \in \{1, \dots, K\}$. As per usual, the time variable will be indexed by $t \in \{1, \dots, T\}$.

Given the difference between the stochastic and non-stochastic model, we are going to be giving some model-specific definitions. Firstly, for the non-stochastic model we can define the following:

> **Definition 1.** The recommendation of expert i at round t is a distribution over the K arms, specified by a vector $e_i(t)$ of K probability masses:

$$e_i(t) = \left(e_{i,1}(t), \dots, e_{i,K}(t)\right)$$

As we can imagine, the sum of the probabilities that each expert i gives in each round t will correspond to 1, thus:

$$\sum_{a=1}^{K} e_{i,a}\,(t) = 1$$

As in the previous sections, the cumulative regret is the difference between the cumulative reward of the best expert and that of the forecaster.

Definition 2. If $r(t) = (r_1(t), \dots, r_K(t))$ is the vector with the rewards of the arms for the round t, then the expected cumulative reward of an expert i up to time T is:

$$y_i(T) = \sum_{t=1}^{T} e_i(t) \cdot r(t)^T$$

Definition 3. This means that the expected cumulative regret of the forecaster up to time T, if he follows the advice of the expert A, is:

$$max_{1 \le i \le N}\big(y_i(T)\big) - \ y_A(T)$$

In the stochastic scenario where the rewards are continuous and we can assume that the context is approximately linear to the rewards of the arms, we use "linear" experts. A *linear expert* corresponds to a *vector of coefficients*: each expert consults on one arm a only, for each round t, and gives its estimation of coefficients for the linear equation of the arm. For the stochastic case we define the following:

Definition 4. The values of the context features for arm a, at time t, will be noted in the form of a M_a-dimensional vector:

$$f_a(t) = \Big(f_{a,1}(t), \dots, f_{a,M_a}(t)\Big)^T$$

Definition 5. If $r_a(t)$ is the reward of the arm a for the round t, then there is a vector of coefficients

$$c_a^* = \big(c_{a,1}^*, \dots, c_{a,M_a}^*\big)^T,$$

for which:

$$r_a(t) = {c_a^*}^T \cdot f_a(t).$$

Definition 6. At time t and for the arm a, the respective linear expert gives an estimation for the coefficients c_a^*. The estimated vector will be noted as:

$$c_a(t) = \Big(c_{a,1}(t), \dots, c_{a,M_a}(t)\Big)^T.$$

Lastly, it should be noted that in the non-stochastic case it is usual for the algorithms to assume the existence of the uniform expert in the expert set, in order to prove the bounds for their algorithms. This would be an expert that always gives the same probabilities to each arm: $1/K$. With this technique the convexity of the rewards' space is ensured.

In the next paragraphs we are going to be looking into some of the most important contextual bandit algorithms. Our presentation of the algorithms is organized by the type or rewards of the arms. The two categories to be discussed are the stochastic category and the non-stochastic one, or worst-case scenario.

3.1 NON-STOCHASTIC CONTEXTUAL BANDIT ALGORITHMS

Our first category is the non-stochastic – or adversarial – contextual bandit setting. In such a scenario, the rewards of the K arms can be arbitrary and there are no assumptions about their generation. In adversarial bandits we use the experts as mere policies, mappings from the context to the arms. There are no assumptions here about the form of the context and the context itself is only used by the experts.

3.1.1 EXP4

The first algorithm that was presented for the non-stochastic contextual bandit problem is the *EXP4*, which was introduced by Auer, Cesa-Bianchi, Freund and Schapire in 2002 [19]. It was presented as part of a family of policies, that can be implemented in the non-stochastic, or adversarial, bandit scenarios. The EXP4 – *Exponential-weight algorithm for Exploration and Exploitation using Expert advice* – is a variant of the EXP3 algorithm, which was discussed previously in the section 2.2.1. The idea is the same as EXP3, but here instead of exponentially weighing the arms, we are weighing exponentially the experts, as we are in search for the best strategy.

ALGORITHM EXP4

To initialize:

- Input parameter $\gamma \in (0,1]$
- Set initial weights $w_i(0)$ to 1, for $i \in \{1, \dots, N\}$

Then for each time-step t:

- For each expert $i \in \{1, \dots, N\}$ get the advice vector $e_i(t)$
- Calculate $p_a(t) = (1-\gamma)\sum_{i=1}^{N} \frac{w_i(t)e_{i,a}(t)}{W(t)} + \frac{\gamma}{K}$, for $a = \{1, \dots, K\}$ and $W(t) = \sum_{i=1}^{N} w_i(t)$
- Choose an action a_t, according to the probabilities $p_1(t), \dots, p_K(t)$ and receive reward $r_{a_t}(t)$
- Calculate $\hat{r}(t) = \left(\hat{r}_1(t), \dots, \hat{r}_K(t)\right)$ as: $\begin{cases} \hat{r}_a(t) = r_a(t)/p_a(t), & for\ a = a_t \\ 0, & for\ a \neq a_t \end{cases}$
- For each expert $i \in \{1, \dots, N\}$ calculate $\theta_i(t) = e_i(t) \cdot \hat{r}(t)$
- Set the weights for $i \in \{1, \dots, N\}$: $w_i(t+1) = w_i(t)\exp(\gamma \frac{\hat{r}_i(t)}{K})$

Figure 3.1. Pseudocode for the algorithm EXP4, as presented in Auer et al. [19]

As presented in Figure 3.1, the algorithm needs as input the parameter $\gamma \in (0,1]$, which is related to the exploration. The closer the parameter is to 1, the more uniform and at random is the choice of arms, meaning that the algorithm is more explorative. Whereas, in the case that the parameter is closer to 0 the algorithm gives priority to exploitation. In every round t

of the algorithm, the experts give their advice vectors $e_i(t)$. After that, for each arm a the probabilities $p_a(t)$ are calculated, which are mathematical expressions involving the parameter γ and the weighted advice of each expert i for a specific arm. They are a tool for deciding which arm to pull next: at every round t an action is chosen at random, according to the $p_a(t)$ probabilities. The reward of the drawn arm is received and the weights of the experts are adjusted for the next round.

To prove a bound of regret for the EXP4, the authors assume that there is a uniform expert, who is always included in the set of experts and who assigns uniform weights to all K actions. Under this assumption, it is proved in Theorem 7 of [19] that the expected regret of EXP4 is $O(\sqrt{gKlnN})$, where $g \leq G_{max}$, K is the number of actions and N is the number of experts. As with the proof of EXP3's regret bound, we can use the Corollary 3.2 of [19] if the horizon T is known and rewards are in $[0,1]$ and turn the regret bound into $O(\sqrt{TKlnN})$.

3.1.2 EXP4.P

The algorithm *EXP4.P* – a variant algorithm of EXP4 – was presented by Beygelzimer, Langford, Li, Reyzin and Shapire [29] in 2011. Unlike EXP4, for which we have a proven regret bound in expectation [19], it is proved that for this algorithm we can get a regret bound of $O(\sqrt{KTlnN})$ with high probability. This bound can be very useful in practice and may be preferable to that of EXP4 for applications of contextual bandits.

The pseudocode of EXP4.P has little difference with that of EXP4. The only distinctions lie in the $p_a(t)$ probabilities' and the weights' computation of the experts. That is because for both those computations, the EXP4.P uses a new parameter instead of γ:

$$p_{min} = \sqrt{\frac{lnN}{KT}}.$$

One possible drawback of EXP4.P is that it can favor some badly-performing arms by giving them higher probabilities, because it makes use of the uniform distribution to set up the probabilities of the K arms. EXP4.P, just as EXP4, can also become computationally inefficient, if the number of experts N becomes very high, because the algorithm maintains weights on all the experts.

3.1.3 NEXP

Another algorithm in this category is the *NEXP* – N stands for Nonuniform exploration. The algorithm was introduced as a solution for adversarial contextual bandit problems by McMahan and Streeter, in 2009 [24]. This approach is proposed as fitting in settings where the number of arms K is quite larger than the number of experts N. This is also the case where EXP4 might perform poorly.

A distinction about NEXP is its exploration phase. McMahan and Streeter propose in [24] 3 different exploration policies, which are based on linear programing. These policies are to be

used as subroutines in the code of the algorithm. With the above features, the NEXP algorithm has been proved to achieve a bound for the expected cumulative regret in $O(\sqrt{TSlnN})$, where the S is a measure of how much the experts agree on their recommendations.

The NEXP algorithm, with a specific exploration subroutine, was used in the same work for experimentation in a real-world problem, with real-world dataset of a 12-month period. The training data were Google queries containing a certain keyword and the clicks of the ads that were displayed in it. In the experiments that took place, the number of arms K was approximately 40 times larger than the number of experts N. As the authors promised, NEXP outperformed EXP4, who was expected not to do well under these parameters.

	Algorithm	Regret Bound
Non-stochastic	EXP4 [19]	$O(\sqrt{TKlnN})$ in expectation
	EXP4.P [29]	$O(\sqrt{TKlnN})$ with high probability
	NEXP [24]	$O(\sqrt{TSlnN})$ in expectation
Stochastic	Epoch-greedy [27]	$O(T^{2/3})$ in expectation
	SupLinREL (LinREL) [30]	$O(\sqrt{Td}lnT)$ with high probability
	LinUCB [14]	No theoretical guarantees
	SupLinUCB [25]	$O\left(\sqrt{Tdln^3(KTln(T)/d)}\right)$. with high probability

Table 3.1. Representative contextual bandit algorithms and their regret bounds.

3.2 Stochastic Contextual Bandit Algorithms

In this paragraph we are going to be mentioning some of the most representative algorithms of the stochastic contextual bandit scenario, where the rewards are assumed to be independently and identically distributed.

3.2.1 Epoch-Greedy

A simple algorithm addressing the stochastic contextual bandit problem is the *Epoch-Greedy*. The algorithm was presented by Langford and Zhang, in 2007 [27] and, as its name suggests, it is a greedy algorithm that implements the exploration and the exploitation phases in epochs. The Epoch-Greedy bears great similarity and was inspired by the e-Greedy of the context-free bandit problem, now translated into the contextual bandit problem.

Each new epoch of the algorithm starts with exploration: the algorithm explores the arms and the context with a certain fraction of exploration e. During this phase, the payoffs are observed, so that the most promising results are exploited during the exploitation phase that comes next. After a certain epoch ends, a new one begins with a different value of e. The parameter e gradually decreases to zero as the epochs go by – this means the exploration decreases and the exploitation increases, in the attempt to gain cumulatively more rewards. In [27] it is proven that the algorithm has a regret bound in $O(T^{2/3})$.

3.2.2 Algorithms for context inducing linear rewards

As mentioned before, it is common for contextual bandit settings to be accompanied by assumptions about linear functions of rewards. Therefore, for such settings there have been developed various algorithms, which have been analyzed, or tested experimentally. In 1999, Abe and Long [26] presented such an algorithm – algorithm *A* – for which was proved an upper bound for the expected regret in $O(T^{3/4})$. Later in 2002, Peter Auer [30] improved on that bound with his algorithm *LinREL* and its modified alternative *SupLinREL*, achieving an upper bound for the regret in $O(\sqrt{Td}lnT)$ with high probability.

Another such algorithm for contextual bandits is the *LinUCB*. The algorithm was originally proposed by Li, Chu, Langford and Schapire in 2010 [14]. In this work, the authors discuss the contextual bandit problem through a popular real-world application – personalized news article recommendations in the web. This is only one of the many scenarios where the contextual bandits are applicable and there are other similar web problems that can be modeled this way.

ALGORITHM LinUCB

To initialize:

- Input parameter $\lambda > 0$

Then for each time-step $t \in \{1, \dots, T\}$:

- Observe the K features $f_1(t), \dots, f_K(t)$
- For each arm $a \in \{1, \dots, K\}$:
  - If a is new, then set $A_a = I_d$ and $b_a = 0_{d\times 1}$
  - Set $c_a(t) = {A_a}^{-1} b_a$
  - Calculate upper confidence bound:

$$p_a(t) = c_a(t)^T \cdot f_a(t) + \lambda\sqrt{f_a(t)^T {A_a}^{-1} f_a(t)}$$

- Choose the action a_t, for which $p_a(t)$ is the highest, and receive reward $r_{a_t}(t)$
- Set $A_{a_t} = A_{a_t} + f_{a_t}(t) f_{a_t}(t)^T$
- Set $b_{a_t} = b_{a_t} + r_{a_t}(t) f_{a_t}(t)$

Figure 3.2. Pseudocode of LinUCB, as given in [14]

The LinUCB algorithm is an Upper-Confidence-Bound-type of algorithm and it is applicable to many scenarios beyond the news article recommendations. The "Lin" prefix hints the assumption that the payoffs of the arms are linear to the features of the arms. This linearity assumption allows the algorithm to use ridge regression for the computation of the upper confidence bounds of the arms. As per usual, the algorithm chooses to play the arm with the highest upper confidence bound. LinUCB's logic is very similar to that of its ancestor LinREL, as they both try to predict the arm with the highest expected reward by linearly fitting the past observations.

Specifically, the algorithm applies ridge regression for each arm using training data of the past rounds. This calculation enables the algorithm to define an estimated upper bound for the reward of each arm and, subsequently, to pick the arm with the highest upper bound. The algorithm presents the advantage that arms' pool can be changed – items can be removed or added – and that is very useful for many real-world applications.

The authors consider two models for this setting and present two versions of the LinUCB algorithm: the disjoint and the hybrid one. What makes the distinction between the two of them is the features' type. There are contextual bandit settings, where the context features are exclusively arm-specific, while in others there are common – commonly shared – features between the arms. In the first case we can use the disjoint LinUCB, while in the second case we can use the hybrid LinUCB algorithm.

Although there is no theoretical analysis for LinUCB and no proven upper bound for regret, the algorithm holds interest, because it is simpler in its implementation and more efficient computationally than LinREL. Furthermore, its experimental performance is very promising and, in fact, outperforms other known contextual bandit algorithms.

Because the LinUCB poses a challenge in its theoretical analysis, Chu, Li, Reyzin and Shapire (2011) in [25] are proposing a modification of LinUCB – the *SupLinUCB* – that has an expected regret of $O\left(\sqrt{Tdln^3(KTln(T)/d)}\right)$.

4 RANKED RECOMMENDATIONS

In the problems we discussed in the previous sections, we tried to produce recommendations for just one spot – a single recommendation in each round. In the problem we are going to be discussing next we are required to provide P ranked recommendations. Our interest, again, is to examine the problem through solutions in the spectrum of online learning and bandits.

The traditional recommendation systems, like collaborative filtering, work well in cases when offline computation and evaluation is needed. But in situations where data are produced in fast rates, or in systems that go under frequent changes, there is need for fast online methods.

This is where the online learning and specifically bandits can provide sustainable solutions. In fact, the multiarmed bandits have been proposed as fitting for a number of real-world contemporary problems in the ranked recommendations area. For example:

- In [12] Radllinski, Kleinberg and Joachims are using multiarmed bandit algorithms – adapted for ranking settings – for ranked recommendations of documents to users.
- In [13] Kohli, Salek and Stoddard are using their newly-presented algorithm for rankings and they experiment with two real-life problems: movies' recommendations and jokes' recommendations to users.
- In [31] Katariya, Kveton, Szepesvari and Wen face the problem of ranked web page recommendations in response to search queries.
- In [32] Combes, Magureanu and Proutiere also experiment on the problems of movies' and search queries' recommendations.
- In [33] Yue and Guestrin are using their algorithm in personalized news articles recommendations.

In modern web problems the biggest challenge is to identify the tastes of the users and use that knowledge to captivate their interest. A user's interest – or lack of it – in such settings is revealed through the clicking, or no clicking, of items, links, documents etc. Again, we can use the measure of the *Click Through Rate*. The situation where a user does not click on any of the recommended items is called *abandonment* – also called *%no*. It means that the user found no interest in any of the displayed items. As expected, the challenge in the ranked recommendations problems is minimizing the abandonment.

It is defined that in a ranked recommendations problem there are K items, as a pool of arms. In every round $t \in \{1, \dots, T\}$, the system has to create a list of ranked suggestions of dimension P. The ordered list is displayed and the reward $r(t)$ is recorded. The positions of the ordered list are going to be indexed by $p \in \{1, \dots, P\}$.

4.1 DIVERSE RANKINGS

There are problems where the audience taste might be steady and universal. In that case the best set of items to recommend is deterministic. However, in most problems the taste of different users is rather diversified. In this case, there is need for ranked recommendations that incorporate diversity, so that the engagement of users is maximized.

Older approaches did not consider the relevance of the documents to each other when creating the results for a query. They rather tried scoring the documents in relevance with the query and displayed as recommendations the top P of them. This approach is based on the *Probability Ranking Principle (PRP)* [34], which suggests that the items should be ranked by descending order of probability of being relevant to the query.

In reality, the score, or relevance of the documents with queries should not be deterministic, as two users might mean something different by the same search query. For example, a query with the keyword "bear" in a search engine could refer to the animal bear, or the verb bear. Thus, the display of a document with information about bears would not be appreciated by all users.

The above reasons motivate for more modern approaches that target to produce diverse rankings. A list of suggestions that incorporates diversity is one that lists items that are relevant to the query, but they are as diversified between them as possible. This practice, as opposed to displaying similar versions of the same topics, has the advantage that it increases the likelihood to appeal to the user's taste. There are more options for the user and, therefore, it is more likely that the user will find what he is looking for, and in the long run, more users will be satisfied. Often, a needed step in diverse rankings is establishing a way of calculation, or a measure of how similar two documents are and how much of the same audience are targeting.

Some of the most known works on diverse rankings are described in the following paragraphs.

4.1.1 REC and RBA

In 2008, Radlinski, Kleinberg and Joachims [12] studied the problem of diverse rankings and presented some results, that have been extensively studied ever since. They came up with two algorithms that address the problem of recommending P items out of K, with the intention of maximizing the reward, which usually means maximizing the clicks of the users.

In their approach they claim to produce optimally diverse rankings by learning through the clicking behavior of the users. The ranked recommendations that they produce are not user specific and they do not attempt personalization in any way. They are rather geared towards offering a solution that will be interesting to as many users as possible. Therefore, they seek to find the best set of P objects that will bring the highest possible cumulative reward. For this purpose, they proposed the following two methods.

▪ *Ranked Explore and Commit (REC)*

The REC algorithm, as its name suggests, explores and then commits to the results of the exploration. It is a greedy approach that begins with a massive exploration phase: for each rank P, it tries out all the available documents x times and records the clicks. The documents that received the best score at a certain rank are committed to the respective rank. The authors [12] prove for the algorithm a regret bound in $O(K^3 P/\varepsilon^2 \ln(K/\delta))$.

ALGORITHM REC

Input parameters: ε, δ, P

Set $x = 2P^2/\varepsilon^2 \log(2P/\delta)$
Make a list of P arbitrarily chosen documents
For each rank in $\{1, \dots, p\}$:
    Set scores of documents to zero
    Iterate through $\{1, \dots, x\}$:
        For each document in $\{1, \dots, k\}$:
            - display the current document in current rank
            - if it is clicked, then increase its score by 1
    Choose document with highest score and commit to the current rank

Figure 4.1. Pseudocode for algorithm REC [12]

This algorithm provides a basic solution to a rankings problem: very straight-forward, it tests out the items at different ranks and uses the ones that did best. However, it also carries the limitation that it is not very flexible, as it assumes that the pool of items and the interests of the users remain stationary. The pseudocode for algorithm is given in Figure 4.1 – based on its presentation in [12].

▪ *Ranked Bandits Algorithm (RBA)*

The second algorithm, presented by Radlinski et al. in [12], is the *Ranked Bandits Algorithm* (RBA). The RBA, unlike REC, is an algorithm that learns throughout t. It does not assume that the user interests and pool of items remain the same and it is capable of detecting changes and adjust to them, through constant alternations of exploration and exploitation. The algorithm has an upper bound for the expected regret[2] in $O\left(P\sqrt{TKlogK}\right)$.

The RBA learns and exploits by using P independent versions of the same multiarmed bandit algorithm (MAB), one for each rank of the recommendation list. The MAB algorithm can either be a stochastic, or a non-stochastic one. Each copy, or instance, of the MAB algorithm learns what is the most profitable option for the specific rank and takes advantage of it.

Specifically, for each time-step and each rank, the algorithm asks for the suggestion of the corresponding MAB instance. If the arm that the MAB algorithm suggests has already been selected in a higher rank, then another arm is selected arbitrarily instead. After all ranks have been filled, the selected items are displayed to the user. The user may click up to one item in the list. Then the rewards of the items get determined and subsequently the MAB algorithms

[2] The upper bound in the expected regret is calculated by Radlinski et al. [12], using P instances of the EXP3 as multiarmed bandits. Since EXP3 has the optimal regret bound, the bound of RBA with EXP3 is also optimal. In general, the upper bound of the RBA depends on the multiarmed bandit algorithm that is used with it.

get updated with the respective rewards. The exact steps of the algorithm can be seen in the Figure 4.2. Pseudocode for algorithm RBA .

ALGORITHM RBA

Initialize the P MAB instances

For each time-step $t \in \{1, \dots, T\}$:
  For each rank $p \in \{1, \dots, P\}$:
    - select item with MAB_p
    - if the selected item is displayed in one of the previous ranks $\{1, \dots, p-1\}$, then select arbitrarily another unselected item
  Display the P selected items to user
  For each rank $p \in \{1, \dots, P\}$:
    - Set reward for the item in the position p to:
      $\begin{cases} 1, \text{if the item was clicked and it was selected by the } MAB_p \\ 0, \text{in all other cases} \end{cases}$
    - Update the MAB_p with the reward

Figure 4.2. Pseudocode for algorithm RBA [12]

The work of Radlinski et al. [12] and, specifically, the RBA algorithm is one that has been often studied and built upon [13] [33]. The reason being that it provides a simple, flexible and easy to implement solution. The fact that it can work with both stochastic and non-stochastic multiarmed bandits' algorithms is a big advantage and it makes the algorithm applicable in many different settings.

What is also interesting in this algorithm is its rewards' system, as it has some particularities. In Table 4.1 we can see the four possible cases of rewards for one item, depending on the user clicks and the way the item is selected. If we take a moment to look at the table, we could observe that:

- Only items that were selected by the respective MAB can get a reward of 1, in case they get clicked by the user.
- In case an item has been selected arbitrarily for some rank p, it is predefined that it will receive a reward of 0.

This rewarding system is in line with the greediness of the algorithm, emphasizing on finding the locally optimal choices. However, it should be noted that the lower ranks of the recommendation list might have a learning disadvantage. This is due to the following two facts:

- The lower ranks generally do not get as much attention from the users as the higher ones.
- The lower ranks are more likely to select an item that has been suggested in a higher rank, resulting thus in a zero reward.

As an effect, the last ranks have a more "pessimistic" way of learning, as they are rewarded with zero more frequently.

Definition of reward for item p in RBA		USER CLICKED ON ITEM	
		YES	NO
ITEM WAS SELECTED BY MAB_p	YES	1	0
	NO	0	0

Table 4.1. Reward system of items in RBA

4.1.2 PIE and PIE-C

In 2015, Combes, Magureanu, Proutiere and Laroche [32] presented the results of their studying on the "structured" stochastic multiarmed bandit problems. As they explain, by structure they mean that the reward functions of the arms of the multiarmed bandit have certain properties.

In their approach, they consider a system where the user can input queries. Upon request, the system must return an ordered list of P items out of the K available ones. The target is the maximization of the cumulative reward. To achieve this target, they propose categorization of items, according to the topic and clustering of the users, according to their interests. The number of item categories – let us say C – is equal to the number of user clusters, creating thus a 1-1 mapping of the topics and the users. That is to say, a user of a certain cluster would like the items of a certain items' category.

Of course, in this approach there is the difficulty of not always knowing the user cluster, or the topic of the query. For this reason, the authors [32] examine separately the two cases. In the first case, where the topic of the query is known, they introduce a new algorithm, the *Parsimonious Item Exploration* (PIE), which has a regret in $O(|c|\log(T))$, with $|c|$ being the number of items in the item category c. They prove that this lower bound stands for any algorithm in this specific problem. In the second case, they consider the scenario, where the user cluster is known, but the mapping between user clusters and item categories is unknown. For this case, they introduce the *Parsimonious Item Exploration-Clustered* (PIE-C), which has the same regret as PIE, since the learning of the mapping only inflicts a constant regret, as they prove in section 6 [32].

Both algorithms attempt to bring diversity in the recommended lists, by exploring several topics, in case that the topic of the query is unknown. At the same time, the algorithms try to incorporate a system, where the items are presented in a decreasing order of relevance. With the assumption that the user examines the items of the list from the top to the bottom and clicks if he finds something relevant, the reward decreases with the position of the item

clicked. Note that both algorithms take for granted that the system has some side information on the user and, therefore, the user cluster is always known.

In order to confirm the optimal performance of their results, the authors conducted experiments in the same work [32], where they used both artificial data and the real-world dataset Movielens.

4.1.3 LSBGreedy

In 2011, Yisong Yue and Carlos Guestrin [33] work on the combination of the contextual bandit problem with the ranked recommendations, in order to produce diverse rankings. Their study is a logical next step for the contextual bandits and their application in more than just single recommendations, as well as for the rankings, which now can be more flexible and personalized through the use of context. For this purpose, they define the *linear submodular bandits' problem*, where the target is to optimize a submodular utility function and yield diverse rankings. For the defined problem they introduce a new algorithm, the *LSBGreedy*, which has a regret bound[3] in $O\left(d\sqrt{PT}\right)$.

This approach has multiple advantages, the biggest of which is the flexibility. In such a setting it is possible to change the pool of arms, which is great for systems that undergo frequent changes, or updates. With an algorithm, such as the LSBGreedy, the system can generalize from past knowledge and recommend to new users, or find out how to use newly entered items in recommendations. In addition, the system is able to produce not just a single best set for all users, like other approaches do [12] [32], but it is able to adapt to each user's interests, achieving greater personalization. Lastly, by applying the principal of diversity we can avoid redundant recommendations and appeal to a wider range of users.

The authors also put their ideas into practice: they conduct experiments with a synthetic and a real-world dataset with news articles recommendations from a blog [35]. In the experiments they compare the LSBGreedy, with the RBA using LinUCB, Multiplicative Weighting [35] and e-Greedy. The results reveal that LSBGreedy and RBA-LinUCB do much better that the other two algorithms, with LSBGreedy having the best overall performance between the four of them.

4.2 Multiple Clicks

Some recommendation systems, like the ones we are discussing, are able to support the feature of accepting *multiple clicks*. This means that such a recommendation system can record the reaction of a user in more than one items of the recommended list. Unlike all the settings that we discussed previously, in such a system the multiple clicks of the user are accepted, tracked and rewarded befittingly.

[3] Ignoring log factors. d is the number of topics that we wish to cover.

This is a very advantageous feature for two main reasons. The first one is that in many real-world scenarios the multiple clicks are more realistic, as opposed to limiting the user to just one click. The second is that the multiple clicks permit the algorithm to learn faster, as there is more feedback per round usually.

Some examples of bandit recommendation works that support multiple clicks are the following.

4.2.1 IBA

An interesting work that is similar to that of Radlinski et al. [12] – transferred in the multiple clicks setting – is that of Kohli, Salek and Stoddard, in 2013 [13]. In this work, they present the *Independent Bandit Algorithm* (IBA), which promises great performance, fast learning from past user behavior and minimizing abandonment. While the RBA – discussed in the previous section (4.1) – is based on the diversity principle, the IBA is based on the PRP principle. Kohli et al. make a comparison of the two principles by comparing the two algorithms, RBA and IBA.

Just like RBA, the IBA uses P instances of the same multiarmed bandit algorithm, one for each rank of the recommendation list. These instances work independently to form collectively the best set of items to recommend. The *optimal set* is considered the one that appeals to the most possible users. This means that, in the optimal set, for the most possible users there is at least one item out of the recommended ones, which they find relevant.

The writers assume that, for every pair of user and item, there is a mapping of 0 or 1, which represents the relevance of the item for the user. Therefore, they use a *vector of relevance* for every user j, $X^j = \{0,1\}^K$, where K is the number of all the items [13]. If there is an item i in the recommended set, which the user finds relevant – meaning that $X_i^j = 1 -$ then the payoff for this item is 1, otherwise 0. The vector of relevance for every user is, of course, unknown, until it is partially revealed (P elements out of K) when the user reacts to the recommended set.

ALGORITHM IBA

Initialize the P MAB instances

For each time-step $t \in \{1, \dots, T\}$:
  For each rank $p \in \{1, \dots, P\}$:
    - select an item with MAB_p that has not been selected in highest ranks
  Display the P selected items to user
  For each rank $p \in \{1, \dots, P\}$:
    - For the item in the position p set reward to:
      $\begin{cases} 1, \text{if the item was clicked on} \\ 0, \text{otherwise} \end{cases}$
    - Update the MAB_p with the reward

Figure 4.3. Pseudocode of the algorithm IBA [13]

Since the target here is to maximize the payoff of the recommended set in expectation and the user payoffs are 0 or 1, then the expected regret is the expected difference between the portion of satisfied users by the optimal set and the portion of satisfied users by the algorithm.

It should be noted that the IBA uses stochastic multiarmed bandit instances and not adversarial ones, as they make an assumption of independence in their theoretical analysis.

The independence of the ranks in IBA is essentially owed to the independence of the payoffs between the ranks. Unlike RBA, where the payoff of one rank depends on the payoff of the previous ranks, the IBA rewards every item that was clicked by the user with 1. In other words – even though it is not specifically advertised in the work of Kohli et al. [13] – the IBA algorithm supports the very interesting feature of multiple clicking by the user. Because of multiple clicks, the different instances of MABs in IBA learn at the same time, which makes the algorithm achieve good results quicker.

This is apparent also in the experiments, where the authors [13] compare the RBA and IBA algorithms, using four datasets. In these experiments they use the stochastic algorithms UCB1 and e-Greedy as MAB instances in both RBA and IBA. The results reveal that the IBA algorithm performs better than RBA in most cases. According to the authors, this might indicate that diversity is not that much needed for maximizing rewards in ranked recommendations.

4.2.2 dcmKL-UCB

In 2016, Katariya, Kveton, Szepesvari and Wen [31] proposed the *DCM bandits*, a learning variant of the *dependent click model* (DCM), which is a generalization of the cascade model.

In this model, the user's query is answered with an ordered list of items, as per usual. The user then can view the list and click on as many items as he wants to examine. The authors assume that the user views the items from top to bottom and clicks on the items he wants to examine one by one. After the clicking of one item:

- if the user leaves, then the system considers that the user has been satisfied
- if the user keeps examining items lower in the list, then the user is yet considered to be unsatisfied.

At the end, the reward is set to one, if the user leaves satisfied, or zero if the user remains unsatisfied. Note that the reward is not specific for every item, but it is intended that the list of suggested items is the one to be rewarded, or not rewarded.

For this problem, the authors proposed the algorithm *dcmKL-UCB*:

At each round t, the algorithm starts by calculating the attraction probabilities for every item – the upper confidence bounds on them, to be exact – which depend on the popularity of the item in the previous rounds. The P items with the highest UCBs are selected to be displayed to the user at the round t. The order of the items in the displayed list depends on the termination probabilities, which are computed according to the frequency of the termination

positions[4] in the past rounds. After the user clicks and examines the items he is attracted to, the reward is defined and the statistics are updated.

This model is quite interesting, as it supports multiple clicking by the user. The question remains, if the assumptions made by the authors make the scenario hard to be applied in real-life problems. In their work, Katariya et al. [31] test their algorithm, dcmKL-UCB, in experiments with a real-world dataset. Even in these experiments, the applicability of the algorithm is dubious, as the assumptions they have made in their analysis are in many cases violated. Nevertheless, the algorithm exhibits good performance even in those cases.

4.3 OTHER WORKS

There are more interesting results and approaches than the ones we have described so far. Therefore, we reference the reader towards:

- Slivkins, Radlinski, Gollapudi (2010) [36], for diverse learning over large documents collection
- Slivkins (2014) [37], for contextual bandits with similarity information
- Bubeck and Cesa-Bianchi (2012) [21] and Busa-Fekete and Hullermeier (2014) [38], for surveys on online learning and bandit studies.

Lastly, we will talk about the work of Yue and Joachims (2009) [39].

4.3.1 DBGD

A different approach from the ones we have seen can be read in the publication of Yue and Joachims, in 2009 [39]. In this work, the authors study the *dueling bandit problem*, which they categorize as “an online optimization problem”.

Yue et al. propose a new algorithm for the above problem, which achieves sublinear regret: the *Dueling Bandit Gradient Descent* (DBGD). As its name suggests, the algorithm uses duels – comparisons – in order to approximate the best retrieval function. That is to say, they choose a relative measures’ method, rather than a most commonly used absolute measures’ method.

Their approach is based on the comparison of neighboring points in a space of defined retrieval functions. In order to do so, they use the notion of distinguishability between functions, upon which they also build the definition of the regret for this problem. The comparison between two functions, the distinguishability, is a reference to the portion of users that prefer the recommendations of the one function over the other.

The authors use a real-world web search dataset for experimentation with the algorithm. By consecutive function comparisons they are performing gradient descent, to produce results with the most rewarding function.

[4] In [31] the termination position is defined as the position reviewed right before the user terminated the examination of the presented list.

5 EXPERIMENTS OF RANKINGS WITH CONTEXT

In the previous chapters we talked about some algorithms of various categories. From the algorithms described thus far we were the most interested in LinUCB, as the context seems to be very powerful and has the potential of offering tailored, personalized recommendations to users. Especially, we were intrigued by the idea of using LinUCB instances into the meta-algorithms RBA and in IBA and experimenting with them.

The setting of our experiments is similar to one where a website presents its users with a number of options – recommendations – and the users can click on some of them, all of them, or none. For this setting we determine that the users come in an orderly manner, meaning that they visit the website one after another and they never coincide. The choices in the recommendation list will be made by either the RBA-LinUCB or by the IBA-LinUCB. Although this setting refers to web recommendations, the experiments can be translated and adapted to fit numerous other scenarios involving recommendation, perhaps with different reward system or rules.

In the following sections we will make a comparison of the performance of RBA-LinUCB and IBA-LinUCB and examine the differences in their performance. Although the RBA and IBA algorithms are very similar, there is one key difference that the reader should keep in mind; the RBA allows only one click from the user, while the IBA allows multiple clicks. Our target is to experiment with diversified datasets and see how the above algorithms perform in each case and which are their strengths.

In order to be consistent with the previous chapters, we will use the following notation:

- K, as the number of arms
- T, as the time horizon
- d, as the number of the arm features.

We also define in the table below some terms and metrics that will be used in the following sections, along with some indications about their use (Table 5.1).

Term	Definition	Use cases
Average Rate of Rewards (ARR)	It is defined as the mean of the arms reward rates.	▪ As a metric of diversity for comparisons between the generated datasets ▪ As a predictive variable for the performance of the algorithms
Standard Deviation of Rewards (SDR)	It is the calculated standard deviation between the arm reward rates.	▪ As a metric of diversity for comparisons between the generated datasets ▪ As a predictive variable for the performance of the algorithms
Sum of Smoothed and Averaged Average Regret Sequence (SSAARS)	The sequence of average regret of each repetition is smoothed by 10 and then all the sequences are averaged for each smoothed time-step t. The sum of the averaged and smoothed average regret sequence is the SSAARS.	▪ As an indication of good tuning of LinUCB over a dataset ▪ As a performance metric of RBA-LinUCB and IBA-LinUCB
Accumulated Clicks (AC)	It is the cumulative clicks that the algorithm yields over the horizon T.	▪ As a metric of comparison between the algortihms RBA and IBA with LinUCB.
Learning Distance (LD)	It is the Euclidean distance between the θ vector that the algorithm estimates and the actual θ, according to which the rewards dataset was created.	▪ As a metric of performance and comparison between the algortihms RBA and IBA with LinUCB.

Table 5.1. Terms used in our experimental analysis and their definitions.

5.1 DATASETS CREATION

In this section, we describe the process of creating datasets for experimentation with rankings and context. For this purpose, we created datasets of features and of rewards. These two types of datasets are created in sets, so that the rewards of the rewards' dataset correspond to the feature values of the features' dataset. The logic behind the algorithms creating the datasets is that we first choose random feature values for the arms and a random theta and then we calculate the linear payoffs that develop after the multiplication of the features and the theta. The specific steps of the algorithms are presented in the following section, so that the reader can recreate our experiments.

5.1.1 Artificial data creation

For the creation of artificial features' datasets of K arms, d features and T rounds, we used the algorithm as presented in the Table 5.2. The steps that are followed are:

- Initially, we choose the mean value of every feature of every arm from the uniform distribution $U(0,1)$, having the $l_2 - norm$ of the arm vector being less than or equal to 1

- Then for every round in the horizon T, we choose a feature value from the uniform distribution, which has as mean the mean value of every feature
- Finally, we assemble all the feature values into a vector of vectors, forming thus the dataset of features.

ALGORITHM FOR FEATURES' DATASET CREATION

For each arm $a \in \{1, \dots, K\}$:
Do until $\|m^a\|_2 \leq 1$:
For each feature $f \in \{1, \dots, d\}$:
- draw the mean value $m_{a,f}$ of the feature from the $U(0,1)$

$m^a = (m_{a,1}, \dots, m_{a,d})$

For each time-step $t \in \{1, \dots, T\}$:
For each arm $a \in \{1, \dots, K\}$:
For each feature $f \in \{1, \dots, d\}$:
- draw the feature value $x_{a,f}^t$, for time t, from the uniform distribution that has as mean the $m_{a,f}$

Set the features' dataset:

$$F = \left(\left(\begin{pmatrix} x_{1,1}^1 \\ \vdots \\ x_{1,d}^1 \end{pmatrix}, \dots, \begin{pmatrix} x_{K,1}^1 \\ \vdots \\ x_{K,d}^1 \end{pmatrix} \right), \dots, \left(\begin{pmatrix} x_{1,1}^T \\ \vdots \\ x_{1,d}^T \end{pmatrix}, \dots, \begin{pmatrix} x_{K,1}^T \\ \vdots \\ x_{K,d}^T \end{pmatrix} \right) \right)$$

Table 5.2. The steps of the algorithm used for creating the datasets of the arm features

ALGORITHM FOR REWARDS' DATASET CREATION

Do until $\|\theta\|_2 \leq 1$:
For each feature $f \in \{1, \dots, d\}$:
- draw θ_f from the $U(0,1)$

$\theta = (\theta_1, \dots, \theta_d)$

For each time-step $t \in \{1, \dots, T\}$:
For each arm $a \in \{1, \dots, K\}$:
- set $p = \theta \cdot x_a^t$, where $x_a^t = \begin{pmatrix} x_{a,1}^t \\ \vdots \\ x_{a,d}^t \end{pmatrix}$
- draw the reward $r_a(t)$ from the $Ber(p)$

Set the rewards' dataset:

$$R = \Big(\big(r_1(1), \dots, r_K(1) \big), \dots, \big(r_1(T), \dots, r_K(T) \big) \Big)$$

Table 5.3. The steps of the algorithm used for creating the datasets of the contextual arm rewards

Using the same parameters – K arms, d features, T rounds – we proceed to create the dataset of rewards. The steps of the algorithm for this dataset generation are presented in Table 5.3 and they can be described as following:

- We begin by choosing the theta – θ – of length d from the uniform distribution $U(0,1)$
- Then for every round in the horizon T, we multiply the chosen theta with the features chosen for the round for every arm
- We use the result of every such multiplication as the parameter of a Bernoulli distribution, according to which we draw the payoff for a specific arm, for a specific round
- Finally, we create a vector of vectors out of the arms' payoffs, which is the required dataset.

5.1.2 Dataset types

In order to experiment with LinUCB, RBA-LinUCB and IBA-LinUCB, we created datasets of the six following types: 1F, 1R, 2F, 2R, 3F, 3R. With "F" we notate a dataset of feature values, whereas with "R" we notate a dataset of arm rewards. In front of F, or R, there is a number which indicates the number of features in the dataset. The features' dataset and the rewards' datasets are produced in bundles – sets – where the rewards of the "R" dataset are dependent to the values of the features in the "F" dataset and the chosen theta.

Full details on the different datasets used in our experiments can be found in the appendix of datasets, at the end of our work.

Set	Category	K	d	T	Description	Brief Type Name
Set 1	F	10	1	10000	Features' dataset	1F
	R	10	1	10000	Rewards' dataset	1R
Set 2	F	10	2	10000	Features' dataset	2F
	R	10	2	10000	Rewards' dataset	2R
Set 3	F	10	3	10000	Features' dataset	3F
	R	10	3	10000	Rewards' dataset	3R

Table 5.4. The types of the datasets that were used in our experiments on rankings with context.

5.2 METHODOLOGY

For the experiments, we selected and used 10 datasets for each of the categories in Table 5.4. The selection of the 10 datasets of each category was made between many generated datasets, with the following criterion in mind; the datasets in the selection should be as diversified as possible. The metric used to measure the diversity between the datasets is the standard deviation between the arm rewards (SDR), which is a metric defined and explained in the Table 5.1.

5.2.1 Selection of the datasets

The steps in the selection process of the datasets for experimentation with the algorithms are:

- Generating sets of datasets – features' datasets and rewards' datasets – using the algorithms described in Table 5.2 and Table 5.3.
- Measuring for each rewards' dataset of each set the standard deviation between the reward rates of the arms (SDR).
- Selecting 10 rewards' datasets – and the 10 corresponding features' datasets – that are diversified in SDR, according to the above criterion.

5.2.2 Experimental phase

In this section we present the specifics of the experimental phase. All the experiments, as well as the datasets' generation, were implemented in Python. For each selected set of datasets, the experimental phase proceeds as following:

Step 1. The features' and the rewards' datasets of the set are used for tuning the algorithm LinUCB. Various values of the parameter a are tested, each one of them over 100 repetitions of the algorithm. The performances[5] of the algorithm for the various values of a are compared and the parameter that yielded the best performance of the algorithm is chosen for the next experiments.

Step 2. The set of the two datasets are used for implementing the RBA-LinUCB algorithm. The RBA runs for 3 slots, using 3 instances of the LinUCB algorithm with the parameter a that was chosen in Step 1. This experiment is repeated 100 times and some performance[6] metrics are collected and used for the analysis.

Step 3. The set of the two datasets are used for implementing the IBA-LinUCB algorithm. The IBA runs for 3 slots, using 3 instances of the LinUCB algorithm with the parameter a that was chosen in Step 1. Again, this experiment is repeated 100 times and some performance[7] metrics are collected and used for the analysis.

5.3 Analysis of the Experiments

By analyzing the experiments, we get to observe the behavior of the algorithms, hopefully, discovering patterns and gaining some intuition over them. For the purpose of the analysis of

[5] The performance of the algorithm is evaluated by the examination of the plot of its average regret and the SSAARS metric, which is defined in Table 5.1.
[6] The performance of the algorithm is the evaluated by the SSAARS metric (Table 5.1), the accumulated clicks and the learning distance.
[7] The performance of the algorithm is the evaluated by the SSAARS metric (Table 5.1), the accumulated clicks and the learning distance.

the experimental results, we plot, measure and observe the performance of each algorithm, over the diverse datasets.

Specifically, in the following sections we are going to be evaluating the ARR and SDR metrics as predictive variables of the performance of RBA-LinUCB and IBA-LinUCB. We will compare the performance of the two algorithms through the SSAARS metric. We will also compare the two algorithms through the cumulative clicks they yield and through their ability to learn quickly and accurately.

5.3.1 Average Rate of Rewards (ARR) and Standard Deviation of Rewards (SDR)

In this phase of the analysis we observe the *average rate of rewards* (ARR) and the *standard deviation of rewards* (SDR) of the selected rewards' datasets. Specifically, we examine the relationship between ARR and SDR in the rewards' datasets we used in our experiments. The plots for the 3 different types of sets can be seen below.

This paragraph and the examination of the relationship between the ARR and SDR is, hopefully, going to add some extra insight for our readers for the paragraphs to come. As we can see in the Figure 5.1, the average reward rate and the standard deviation of the rewards seem to be correlated in all cases with a positive linear relationship. Naturally, this observation does not stand against instinct. Furthermore, it seems that the higher the number of arm features, the fuzzier the correlation line gets. For d=3 the positive correlation is still apparent, but it is quite noisy.

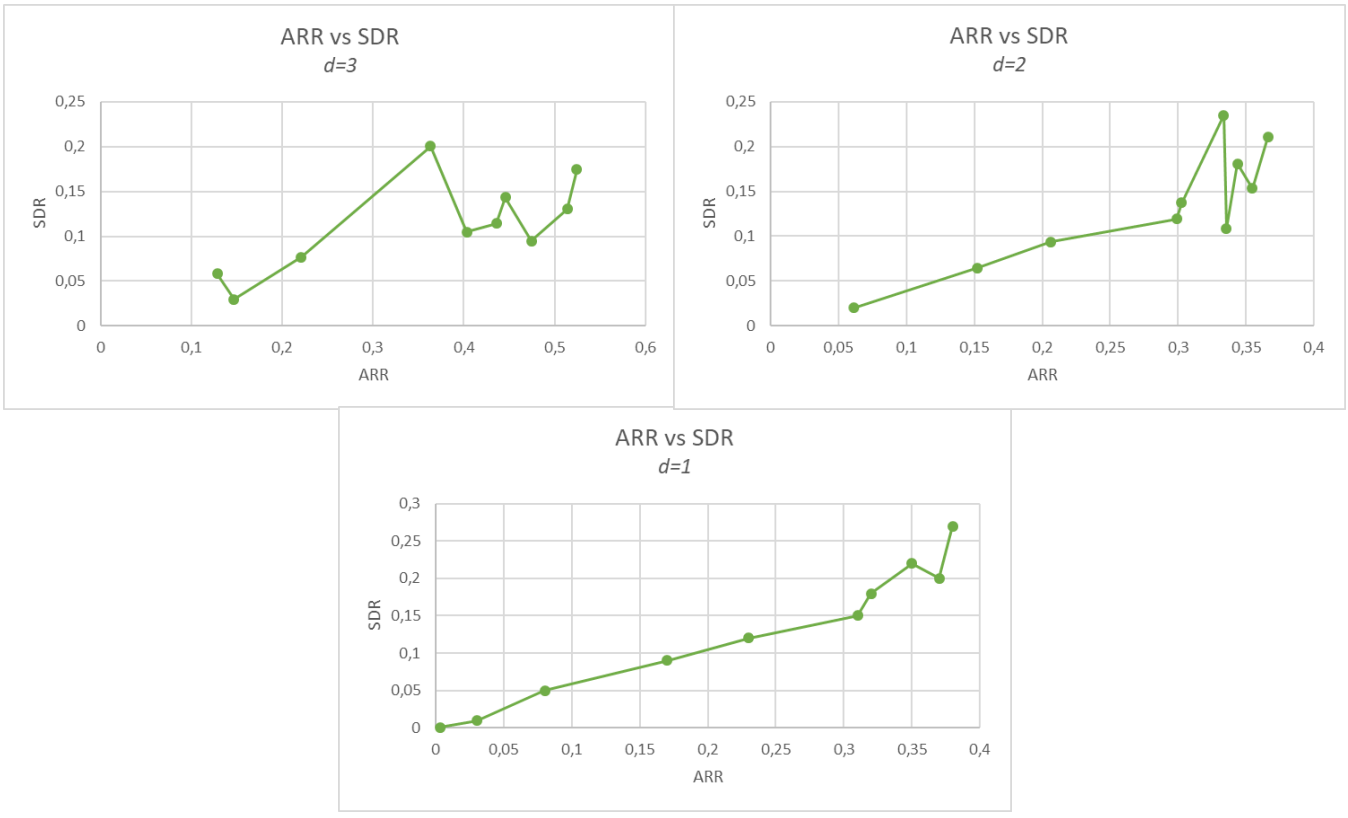

Figure 5.1. The relationship between ARR and SDR for number of features $d \in \{1,2,3\}$.

5.3.2 RBA-LinUCB vs IBA-LinUCB via ARR and SDR

In this section we plot and analyze the relationship of ARR and SDR against the SSAARS metric, which corresponds to the cumulative average regret. One of our goals is to observe if the ARR or the SDR are related to the SSAARS result of RBA-LinUCB and IBA-LinUCB. Our second concern is to compare the performance of the two algorithms through the SSAARS. The plots are depicting the experiments with 10 datasets with variations in ARR and SDR, repeated 100 times.

Figure 5.2. Cumulative average regret and its relationship with ARR and SDR for number of features $d \in \{1,2,3\}$.

In the Figure 5.2 there is an apparent differentiation between the lines that correspond to the two algorithms. For the IBA-LinUCB there is a clear positive, almost linear, correlation between the standard deviation of the rewards (SDR) and the SSAARS. On the other hand, the

RBA-LinUCB seems to be generally unaffected by the standard deviation. However, in all three cases it seems that the relationship between the two algorithms is similar; the lower values of SDR make the RBA-LinUCB have a slightly higher SSAARS than the IBA-LinUCB. Nevertheless, the two lines have a meeting point around 0.1-0.15 and then the difference between them increases again with the IBA-LinUCB rising steadily upwards.

The above observations are reasonable, if we consider the way the average regret is calculated for the RBA and the IBA; in each time step t the RBA can have a regret of 0 or 1, because it only allows one click from the user, whereas the IBA can have a regret of 0 to 3 for a 3-slot recommendation, as it allows multiple clicks (Figure 5.3). This means that, as the standard deviation increases, the probability of having an arm paying higher also increases, which in turn favors the RBA, who finds the best arm and exploits it, thus achieving low regret. However, the IBA has a higher regret by definition, since in its regret calculation all the unclicked slots are considered, but this does not necessarily mean that the performance is worse than that of RBA's. In order to measure better the performance of the RBA-LinUCB and IBA-LinUCB and compare the two algorithms closely, we will use some more metrics and plots for comparison in the next sections.

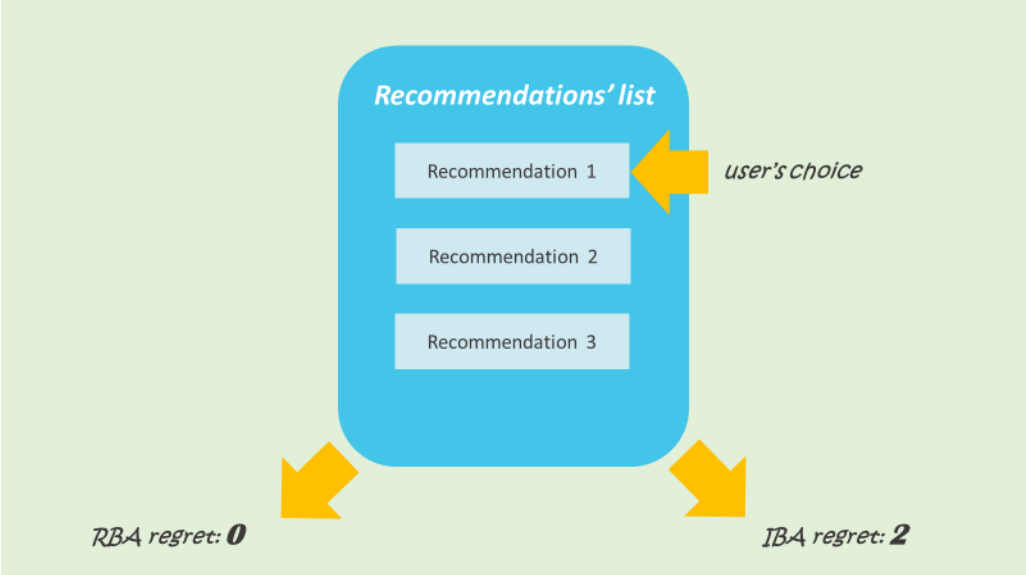

Figure 5.3. Regret calculation for RBA and IBA in the same scenario.

The average rate of rewards (ARR) on the other hand, seems to be a rather unfit predictive variable for the behavior of the algorithms, as the lines are far noisier. Again, we can see similar patterns between the three plots, with the RBA line being higher than that of IBA for lower values of ARR and the lines reversing around 0.25-0.3 of ARR.

Taking a look at the next plots (Figure 5.4), where the metric of difference in SSAARS between RBA-LinUCB and IBA-LinUCB is plotted against the SDR, will reveal a clearer picture. The SDR appears to be a negative linear predictive variable for the difference in SSAARS between the two algorithms. This means that, for the lower values of SDR in the datasets, the RBA has a higher regret, while its performance gets significantly better than that of IBA's for the higher SDR values. As before, this observation is in line with the regret calculation philosophy of the two algorithms.

Figure 5.4. Difference of RBA-LinUCB minus IBA-LinUCB in SSAARS against SDR for feature number $d \in \{1,2,3\}$.

5.3.3 RBA-LinUCB vs IBA-LinUCB via clicks

Since – as we mentioned above – the average regret and the SSAARS metric are affected by the way each algorithm calculates regret, we believed that they might not be suitable indicators of the true performance of an algorithm. Towards that purpose, we found useful to approach the comparison of the algorithms from a different viewpoint. Instead of looking at vague metrics, we decided to take a look at the actual clicks each algorithm had achieved.

Indeed, we monitored the clicks each algorithm yielded over the horizon $T = 10000$. The datasets that were used were the same as before; $K = 10, d \in \{1,2,3\}$ and varying average rewards (ARR).

Our first and most obvious conclusion is that in all cases the IBA-LinUCB yielded more clicks, cumulatively. For all values of ARR, and all feature values, the IBA always accomplished to do better than RBA. The second – and very interesting conclusion – is that there seems to be relationship between the ARR value of the dataset and the increase in IBA clicks, over RBA clicks. In fact, it appears that the higher the ARR is in the dataset, the higher is the difference between IBA and RBA clicks, with IBA being always on top. In the set of figures (Figure 5.5) one can see the cumulative clicks each algorithm achieved for the different values of ARR, as well as the increase in IBA clicks, as a percentage of the RBA clicks. In one specific case – for ARR=0.52 and d=3 – the IBA-LinUCB managed to get 90% increased clicks, which means it got almost double clicks, compared to RBA-LinUCB.

One could argue that it is the capability of IBA to accept multiple clicks that makes the difference and accounts for IBA-LinUCB always being a winner at the click battle. In any case, the click comparison indicates that the use of IBA-LinUCB in a recommendation system of rankings with context would be more lucrative than the use RBA-LinUCB.

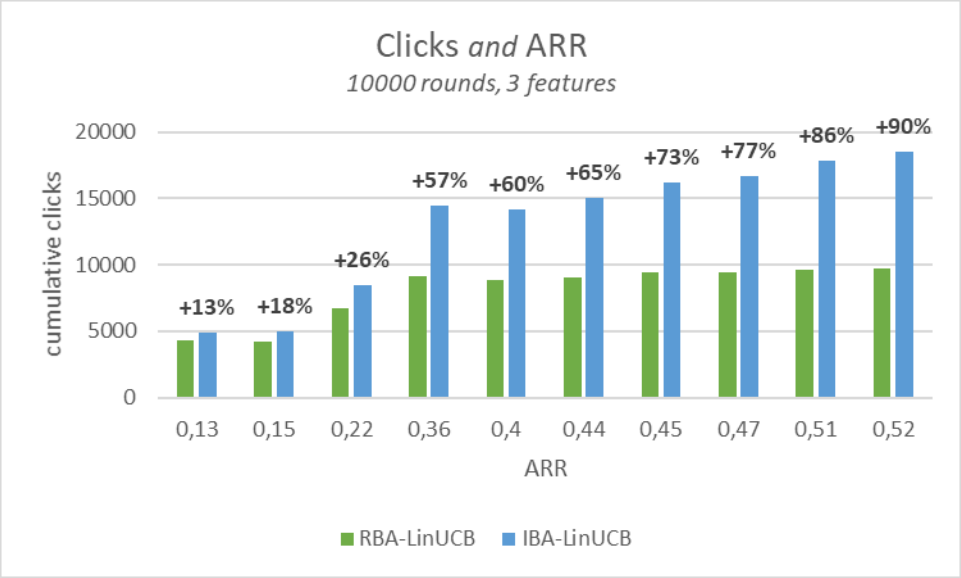

Figure 5.5. The accumulated clicks of RBA-LinUCB and IBA-LinUCB for $d \in \{1, 2, 3\}$, T=10000, K=10. The charts display the increase in IBA clicks as a percentage of the RBA clicks. The results were averaged over the 100 experimental repetitions for each dataset.

5.3.4 RBA-LinUCB vs IBA-LinUCB via learning

As a further step, in the spirit of comparing the algorithms in ways that are more intuitive than dry metrics, we decided to monitor and analyze the learning style of each algorithm. While thinking of possible ways in which we could measure the learning, it occurred to us that the following *learning distance* (LD) could be quite intuitive and insightful; the Euclidean distance between the theta that the algorithm estimates in round t and the actual chosen theta of the dataset. This metric could show us how far is the algorithm from predicting the real theta at every moment.

For each slot, the learning distance was monitored individually, since each slot runs a different algorithm instance. For each feature parameter, $d \in \{1,2,3\}$, two datasets were chosen; one with a larger standard deviation between the arm rewards and one with a smaller one.

The results of the experiments are presented in the figures (Figure 5.6, Figure 5.7, Figure 5.8). As one might have expected, the first slots for the two algorithms learn in exactly the same way, while the differences lie in the second and third slots, correspondingly. It appears that the second and third slots of IBA-LinUCB learn in a much faster and more effective way than the respective slots of RBA-LinUCB. In fact, the IBA slots number 2 and 3 appear to learn so well, that in the cases of two and three features they learn even better than the first slot.

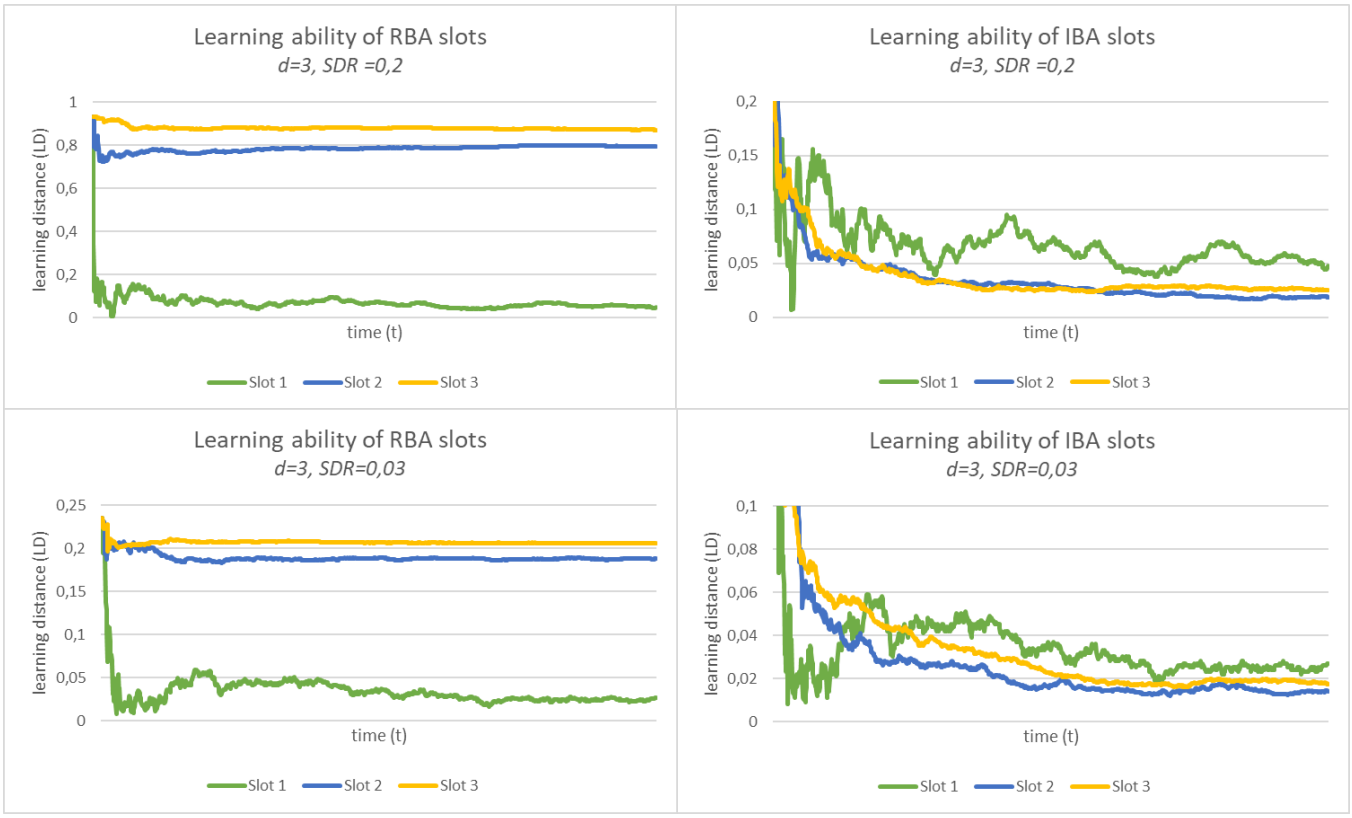

Figure 5.6. The ability to learn for the RBA and IBA slots, tested for 3 features and with two datasets with $SDR \in \{0.03, 0.2\}$.

As for the effect of SDR in learning, it seems that a higher SDR causes a higher gap between the estimated theta and the real theta. It is important to note that a small learning distance is bound to bring better results, as the closer it is the estimation of theta, the higher would be the reward, as an effect of the linear rewards of LinUCB.

It is our estimation that two factors contribute towards the advantageous situation for the IBA-LinUCB, as described above. First of all, the fact that the IBA allows for multiple clicks and, thus, for more feedback. And secondly, the stricter feedback system of the clicks for RBA – as discussed in paragraph 4.1 – leads to erroneous learning of all the slots, apart from the first one. As a consequence, the IBA slots make more informed recommendations, that is recommendations that have a higher possibility of being clicked.

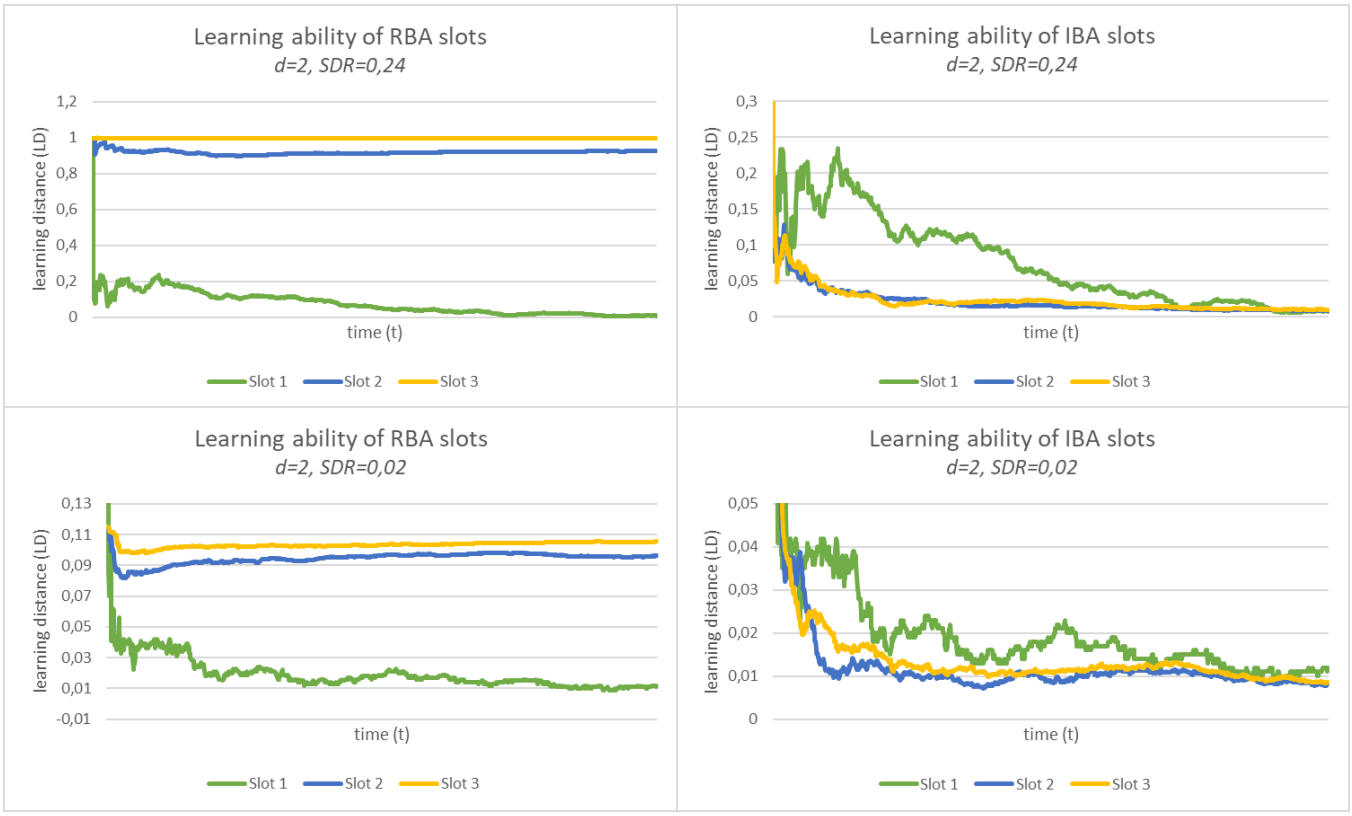

Figure 5.7. The ability to learn for the RBA and IBA slots, tested for 2 features and with two datasets with $SDR \in \{0.02, 0.24\}$.

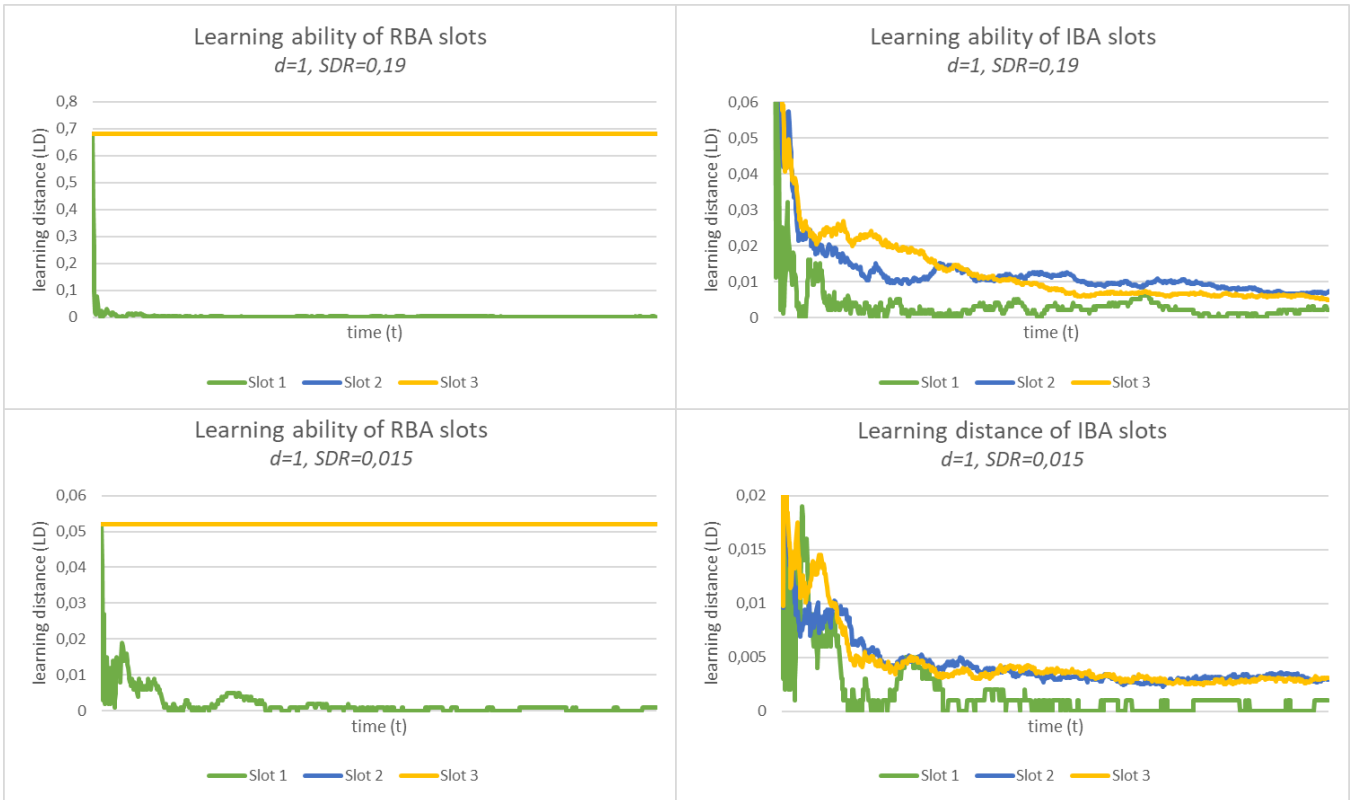

Figure 5.8. The ability to learn for the RBA and IBA slots, tested for 1 feature and with two datasets with $SDR \in \{0.015, 0.19\}$.

5.4 CONCLUSIONS

To sum up, this work is a study on the online learning area with a special focus in its application in the ranked recommendations with use of context. We examined the Multiarmed Bandit problem and we studied bibliographically its origins and the algorithms that treat the problem. We evaluated the advantages and disadvantages of such algorithms and their applicability in real-world scenarios. We also studied the Contextual Bandits, that involve the side-information in their decisions, and the problem of Ranked Recommendations. We revised the algorithms that fit in such recommendations' scenarios and decided to contribute by combining the use of context and the ranked recommendations. We selected the meta-algorithms RBA and IBA with LinUCB instances, thus using the context with linear rewards in rankings of recommendations.

Finally, we generated artificial data so that we can experiment with the algorithms RBA-LinUCB and IBA-LinUCB, which make use of the context in order to produce ranked recommendations. We compared the two of them to understand how they learn, how they perform and what is the relationship between their performance and the metrics ARR and SDR. Our results showed that the SDR has a positive linear relationship with the cumulative average regret (SSAARS) of IBA-LinUCB, while the regret of RBA-LinUCB remains unaffected by SDR, as the standard deviation of rewards increases. It was also clear that there is a negative linear relationship between the SDR and the difference in SSAARS of RBA-LinUCB minus IBA-LinUCB. This revealed that as the SDR increases, the RBA has an increasingly better performance than the IBA.

On the other hand, the monitoring of the clicks yielded by each algorithm exposed the fact that as the ARR increases in the dataset, the IBA-LinUCB brings increasingly more clicks than the RBA-LinUCB. Lastly, we observed the way the slots – instances – of each algorithm learn and we noticed that the IBA-linUCB slots learn much faster and more accurately than those of RBA-LinUCB. This means that the IBA is bound to yield more clicks than the RBA and, thus, leave the user – or client – more satisfied.

6 REFERENCES

[1] R. S. SUTTON and A. G. BARTO, Reinforcement Learning: An Introduction, Bradford/MIT Press, Cambridge, MA; second edition, 2018.

[2] M. MOHRI, A. ROSTAMIZADEH and A. TALWALKAR, Foundations of Machine Learning, Adaptive computation and machine learning, MIT Press 2012, ISBN 978-0-262-01825-8, pp. I-XII, 1-412.

[3] N. CESA-BIANCHI and G. LUGOSI, Prediction, Learning, and Games, Cambridge University Press 2006, ISBN 978-0-521-84108-5, pp. I-XII, 1-394.

[4] N. CESA-BIANCHI, Y. FREUND, D. HAUSSLER, D. P. HELMBOLD, R. E. SCHAPIRE and A. K. WARMUTH, "How to use expert advice.," *Journal of Association for Computing Machinery 44(3): 427-485 (1997).*

[5] P. AUER, N. CESA-BIANCHI, Y. FREUND and R. E. SCHAPIRE, "Gambling in a Rigged Casino: The Adversarial Multi-Arm Bandit Problem," in *FOCS 1995: 322-331*.

[6] N. LITTLESTONE, "Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm".*Machine Learning 2(4): 285-318 (1987).*

[7] N. LITTLESTONE and M. K. WARMUTH, "The Weighted Majority Algorithm," in *Foundations of Computer Science (FOCS) 1989: 256-261*.

[8] P. AUER, N. CESA-BIANCHI, Y. FREUND and R. E. SCHAPIRE, "Gambling in a Rigged Casino: The Adversarial Multi-Arm Bandit Problem," in *Foundations of Computer Science (FOCS) 1995: 322-331*.

[9] Y. FREUND and R. E. SCHAPIRE, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting"*.J. Comput. Syst. Sci. 55(1): 119-139 (1997).*

[10] A. BLUM, V. KUMAR, A. RUDRA and F. WU, "Online Learning in Online Auctions".*Theor. Comput. Sci. 324(2-3): 137-146 (2004).*

[11] K. MISRA, E. M. SCHWARTZ and J. ABERNETHY, "Dynamic Online Pricing with Incomplete Information Using Multi-Armed Bandit Experiments (February 13, 2018). Available at SSRN: https://ssrn.com/abstract=2981814 or http://dx.doi.org/10.2139/ssrn," [Online].

[12] F. RADLINSKI, R. KLEINBERG and T. JOACHIMS, "Learning diverse rankings with multi-armed bandits.," in *International Conference on Machine Learning (ICML) 2008: 784-791*, Helsinki, Finland.

[13] P. KOHLI, M. SALEK and G. STODDARD, "A Fast Bandit Algorithm for Recommendation to Users with Heterogenous Tastes," in *Association for the Advancement of Artificial Intelligence (AAAI) 2013: 1135-1141*.

[14] L. LI, W. CHU, J. LANGFORD and R. E. SHAPIRE, "A Contextual-Bandit Approach to Personalized News Article Recommendation," in *The International World Wide Web Conference (WWW) 2010: 661-670*, Raleigh, North Carolina, USA.

[15] S. S. VILLAR, J. BOWDEN and J. WASON, "Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges".*Statistical Science 30(2): 199–215, 2015.*

[16] B. AWERBUCH and R. KLEINBERG, "Online Linear Optimization and Adaptive Routing". *J. Comput. Syst. Sci. 74(1): 97-114 (2008).*

[17] E. M. SCHWARTZ, E. T. BRADLOW and P. S. FADER, "Customer Acquisition via Display Advertising Using Multi-Armed Bandit Experiments".*Marketing Science 36(4): 500 - 522 (2017).*

[18] P. AUER, N. CESA-BIANCHI and P. FISCHER, "Finite-time Analysis of the Multiarmed Bandit Problem".*Machine Learning 47(2-3): 235-256 (2002).*

[19] P. AUER, N. CESA-BIANCHI, Y. FREUND and R. E. SCHAPIRE, "The Nonstochastic Multiarmed Bandit Problem".*SIAM J. Comput. 32(1): 48-77 (2002).*

[20] H. ROBBINS, "Some Aspects of the Sequential Design of Experiments".*Bulletin of the American Mathematical Society 58(5): 527-535 (September, 1952).*

[21] S. BUBECK and N. CESA-BIANCHI, "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems".*Foundations and Trends in Machine Learning 5(1): 1-122 (2012).*

[22] T. L. LAI and H. ROBBINS, "Asymptotically Efficient Adaptive Allocation Rules," in *Advances in Applied Mathematics and Mechanics (AAMM) 6: 4-22 (1985)*.

[23] R. AGRAWAL, "Sample Mean Based Index Policies with O(log n) Regret for the Multi-Armed Bandit Problem".*Advances in Applied Probability 27(4): 1054-1078, (December 1995)* .

[24] H. B. MCMAHAN and M. J. STREETER, "Tighter Bounds for Multi-Armed Bandits with Expert Advice," in *Conference on Learning Theory (COLT) 2009*.

[25] W. CHU, L. LI, L. REYZIN and R. E. SCHAPIRE, "Contextual Bandits with Linear Payoff Functions," in *Artificial Intelligence and Statistics (AISTATS) 2011: 208-214*.

[26] N. ABE and P. M. LONG, "Associative Reinforcement Learning using Linear Probabilistic Concepts," in *International Conference on Machine Learning (ICML) 1999: 3-11*.

[27] J. LANGFORD and T. ZHANG, "The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information," in *Neural Information Processing Systems (NIPS) 2007: 817-824*.

[28] N. ABE, A. W. BIERMANN and P. M. LONG, "Reinforcement Learning with Immediate Rewards and Linear Hypotheses".*Algorithmica 37(4): 263-293 (2003).*

[29] A. BEYGELZIMER, J. LANGFORD, L. LI, L. REYZIN and R. E. SCHAPIRE, "Contextual Bandit Algorithms with Supervised Learning Guarantees," in *Artificial Intelligence and Statistics (AISTATS) 2011: 19-26*.

[30] P. AUER, "Using Confidence Bounds for Exploitation-Exploration Trade-offs".*Journal of Machine Learning Research 3: 397-422 (2002).*

[31] S. KATARIYA, B. KVETON, C. SZEPESVARI and Z. WEN, "DCM Bandits: Learning to Rank with Multiple Clicks.," in *International Conference on Machine Learning (ICML) 2016: 1215-1224*.

[32] R. COMBES, S. MAGUREANU, A. PROUTIERE and C. LAROCHE, "Learning to Rank: Regret Lower Bounds and Efficient Algorithms," in *Association for Computing Machinery's Special Interest Group on Measurement and Evaluation (SIGMETRICS) 2015: 231-244*.

[33] Y. YUE and C. GUESTRIN, "Linear Submodular Bandits and their Application to Diversified Retrieval," in *Neural Information Processing Systems (NIPS) 2011: 2483-2491*.

[34] S. E. ROBERTSON, "The Probability Ranking Principle in IR".*Journal of Documentation 33: 294-304 (1977).*

[35] K. EL-ARINI, G. VEDA, D. SHAHAF and C. GUESTRIN, "Turning down the noise in the blogosphere," in *Conference on Knowledge Discovery and Data Mining (KDD) 2009: 289-298*.

[36] A. SLIVKINS, F. RADLINSKI and S. GOLLAPUDI, "Learning optimally diverse rankings over large document collections," in *International Conference on Machine Learning (ICML) 2010: 983-990*.

[37] A. SLIVKINS, "Contextual Bandits with Similarity Information".*Journal of Machine Learning Research 15(1): 2533-2568 (2014).*

[38] R. BUSA-FEKETE and E. HULLERMEIER, "A Survey of Preference-based Online Learning with Bandit Algorithms," in *International Conference on Advanced Laser Technologies (ALT) 2014: 18-39*.

[39] Y. YUE and T. JOACHIMS, "Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem.," in *International Conference on Machine Learning (ICML) 2009: 1201-1208*, Montreal, Canada.

[40] A. KALAI and S. VEMPALA, "Efficient Algorithms for Online Decision Problems".*J. Comput. Syst. Sci. 71(3): 291-307 (2005).*

[41] A. AGARWAL, D. J. HSU, S. KALE, J. LANGFORD, L. LI and R. E. SCHAPIRE, "Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits," in *International Conference on Machine Learning (ICML) 2014: 1638-1646*.

APPENDIX OF DATASETS

	Mean rewards of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**a**
3R-1	0,4307	0,3582	0,3056	0,4569	0,2841	0,6875	0,7043	0,3164	0,3873	0,5266	1,8
3R-2	0,5621	0,3041	0,5807	0,5114	0,5799	0,4348	0,3614	0,5497	0,3777	0,4811	0,8
3R-3	0,4121	0,6194	0,5934	0,3611	0,4972	0,3712	0,2648	0,2915	0,5245	0,4249	0,5
3R-4	0,333	0,7305	0,6408	0,5641	0,4938	0,4581	0,4724	0,686	0,3519	0,4108	1
3R-5	0,4361	0,3892	0,2379	0,574	0,4821	0,332	0,2641	0,4206	0,5406	0,3557	1,1
3R-6	0,2215	0,0883	0,1082	0,0379	0,0593	0,1087	0,1075	0,1986	0,1748	0,1795	1,1
3R-7	0,7753	0,5284	0,3902	0,5536	0,3933	0,6844	0,5872	0,7443	0,1928	0,3936	0,8
3R-8	0,6794	0,1426	0,2654	0,2789	0,7391	0,1375	0,3522	0,1931	0,3609	0,4829	0,6
3R-9	0,142	0,1617	0,0686	0,1714	0,1662	0,1311	0,1408	0,1708	0,1697	0,1468	0,7
3R-10	0,107	0,2041	0,1495	0,2674	0,1305	0,1799	0,3135	0,2223	0,2847	0,3479	0,07

	Mean feature values of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**θ**
3F-1	(0.34, 0.91, 0.16)	(0.23, 0.33, 0.20)	(0.013, 0.57, 0.29)	(0.43, 0.52, 0.15)	(0.23, 0.37, 0.07)	(0.58, 0.22, 0.62)	(0.82, 0.54, 0.16)	(0.05, 0.76, 0.21)	(0.19, 0.47, 0.28)	(0.55, 0.37, 0.19)	(0.70448539, 0.21952181, 0.60131692)
3F-2	(0.54, 0.047, 0.36)	(0.047, 0.23, 0.64)	(0.65, 0.39, 0.20)	(0.36, 0.33, 0.65)	(0.7, 0.49, 0.06)	(0.15, 0.17, 0.84)	(0.20, 0.18, 0.46)	(0.58, 0.35, 0.27)	(0.49, 0.65, 0.09)	(0.26, 0.1, 0.69)	(0.84162602, 0.03489823, 0.40449049)
3F-3	(0.69, 0.3, 0.58)	(0.49, 0.87, 0.06)	(0.43, 0.83, 0.047)	(0.41, 0.48, 0.7)	(0.71, 0.6, 0.36)	(0.36, 0.37, 0.23)	(0.3, 0.23, 0.24)	(0.73, 0.048, 0.64)	(0.56, 0.66, 0.03)	(0.38, 0.6, 0.49)	(0.32428396, 0.62908945, 0.08559457)
3F-4	(0.42, 0.14, 0.51)	(0.51, 0.76, 0.18)	(0.8, 0.31, 0.015)	(0.88, 0.077, 0.25)	(0.56, 0.31, 0.46)	(0.079, 0.55, 0.29)	(0.68, 0.11, 0.06)	(0.26, 0.75, 0.29)	(0.1, 0.39, 0.44)	(0.63, 0.1, 0.46)	(0.56715651, 0.77019236, 0.02840739)
3F-5	(0.44, 0.37, 0.61)	(0.43, 0.36, 0.17)	(0.28, 0.082, 0.15)	(0.73, 0.47, 0.28)	(0.6, 0.77, 0.23)	(0.23, 0.75, 0.17)	(0.13, 0.43, 0.37)	(0.52, 0.5, 0.15)	(0.7, 0.58, 0.2)	(0.33, 0.87, 0.05)	(0.66932208, 0.19589478, 0.2624173)
3F-6	(0.36, 0.77, 0.34)	(0.31, 0.24, 0.81)	(0.85, 0.35, 0.12)	(0.31, 0.03, 0.78)	(0.52, 0.1, 0.8)	(0.51, 0.32, 0.49)	(0.44, 0.31, 0.57)	(0.64, 0.7, 0.14)	(0.09, 0.62, 0.46)	(0.46, 0.63, 0.2)	(0.02836751, 0.29475412, 0.02568552)
3F-7	(0.83, 0.52, 0.17)	(0.14, 0.22, 0.80)	(0.24, 0.66, 0.12)	(0.46, 0.17, 0.39)	(0.086, 0.36, 0.57)	(0.66, 0.41, 0.36)	(0.37, 0.16, 0.63)	(0.73, 0.29, 0.37)	(0.01, 0.12, 0.33)	(0.38, 0.43, 0.02)	(0.79752291, 0.22365097, 0.48562551)
3F-8	(0.26, 0.71, 0.28)	(0.19, 0.074, 0.37)	(0.25, 0.18, 0.65)	(0.53, 0.18, 0.12)	(0.23, 0.79, 0.46)	(0.4, 0.016, 0.51)	(0.94, 0.17, 0.22)	(0.31, 0.1, 0.2)	(0.25, 0.42, 0.84)	(0.36, 0.51, 0.42)	(0.22624748, 0.90132846, 0.08542429)
3F-9	(0.0003, 0.85, 0.40)	(0.19, 0.72, 0.28)	(0.017, 0.41, 0.088)	(0.64, 0.25, 0.63)	(0.071, 0.79, 0.59)	(0.05, 0.53, 0.61)	(0.14, 0.21, 0.76)	(0.76, 0.44, 0.18)	(0.57, 0.54, 0.54)	(0.16, 0.39, 0.68)	(0.1451488, 0.13533871, 0.12596317)
3F-10	(0.17, 0.14, 0.02)	(0.25, 0.45, 0.22)	(0.11, 0.46, 0.81)	(0.51, 0.34, 0.75)	(0.16, 0.07, 0.42)	(0.1, 0.95, 0.26)	(0.56, 0.18, 0.45)	(0.34, 0.05, 0.39)	(0.44, 0.65, 0.24)	(0.59, 0.43, 0.24)	(0.55194864, 0.11658752, 0.07924622)

	Mean rewards of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**a**
2R-1	0,2216	0,0785	0,0215	0,2046	0,182	0,1705	0,1442	0,1904	0,2227	0,0869	0,6
2R-2	0,1208	0,6194	0,849	0,1538	0,1924	0,1567	0,2747	0,2282	0,1994	0,5425	0,4
2R-3	0,2049	0,1677	0,2225	0,4062	0,079	0,2353	0,1231	0,1597	0,3293	0,136	0,2
2R-4	0,5197	0,2994	0,326	0,3998	0,2284	0,4321	0,3265	0,3354	0,1366	0,0206	0,07
2R-5	0,0567	0,0816	0,0849	0,0209	0,0507	0,0542	0,0802	0,0591	0,0395	0,0826	0,9
2R-6	0,0703	0,3662	0,3382	0,1487	0,2458	0,3477	0,2187	0,4346	0,4698	0,348	1,1
2R-7	0,2433	0,1001	0,4546	0,4921	0,1796	0,2044	0,5449	0,4659	0,5169	0,3445	0,7
2R-8	0,3019	0,1352	0,4173	0,4023	0,4172	0,3888	0,5054	0,2859	0,1797	0,3216	0,6
2R-9	0,5547	0,6997	0,0524	0,1953	0,1792	0,6216	0,5392	0,3567	0,248	0,2133	0,3
2R-10	0,6288	0,0322	0,3434	0,3828	0,3123	0,1737	0,4965	0,3465	0,5762	0,1451	0,4

	Mean feature values of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**θ**
2F-1	(0.43, 0.66)	(0.2, 0.14)	(0.06, 0.03)	(0.42, 0.57)	(0.11, 0.78)	(0.6, 0.07)	(0.07, 0.66)	(0.35, 0.55)	(0.14, 0.95)	(0.29, 0.05)	(0.26772485, 0.2002624)
2F-2	(0.11, 0.6)	(0.64, 0.22)	(0.87, 0.35)	(0.13, 0.81)	(0.19, 0.23)	(0.14, 0.66)	(0.27, 0.17)	(0.22, 0.32)	(0.19, 0.19)	(0.55, 0.17)	(0.99722452, 0.02916942)
2F-3	(0.39, 0.58)	(0.29, 0.53)	(0.52, 0.22)	(0.99, 0.08)	(0.18, 0.11)	(0.56, 0.57)	(0.23, 0.27)	(0.26, 0.55)	(0.78, 0.31)	(0.26, 0.3)	(0.39157734, 0.11512913)
2F-4	(0.8, 0.49)	(0.43, 0.3)	(0.49, 0.43)	(0.62, 0.58)	(0.35, 0.01)	(0.68, 0.64)	(0.51, 0.45)	(0.59, 0.78)	(0.21, 0.02)	(0.03, 0.01)	(0.63825172, 0.10432262)
2F-5	(0.26, 0.5)	(0.15, 0.88)	(0.93, 0.26)	(0.02, 0.23)	(0.33, 0.3)	(0.5, 0.27)	(0.5, 0.57)	(0.11, 0.63)	(0.01, 0.47)	(0.89, 0.21)	(0.08082718, 0.08082319)
2F-6	(0.04, 0.12)	(0.36, 0.58)	(0.98, 0.01)	(0.27, 0.12)	(0.02, 0.5)	(0.62, 0.33)	(0.41, 0.16)	(0.51, 0.72)	(0.23, 0.86)	(0.66, 0.3)	(0.34616488, 0.46392721)
2F-7	(0.9, 0.28)	(0.39, 0.09)	(0.21, 0.73)	(0.26, 0.81)	(0.74, 0.14)	(0.51, 0.25)	(0.01, 0.93)	(0.27, 0.76)	(0.09, 0.86)	(0.45, 0.56)	(0.1235766, 0.59022211)
2F-8	(0.4, 0.25)	(0.17, 0.12)	(0.65, 0.22)	(0.39, 0.81)	(0.44, 0.79)	(0.4, 0.7)	(0.78, 0.32)	(0.03, 0.89)	(0.2, 0.19)	(0.3, 0.52)	(0.55971034, 0.30549954)
2F-9	(0.72, 0.31)	(0.9, 0.23)	(0.04, 0.18)	(0.17, 0.58)	(0.2, 0.23)	(0.79, 0.25)	(0.69, 0.4)	(0.42, 0.63)	(0.2, 0.91)	(0.14, 0.89)	(0.76649147, 0.12060185)
2F-10	(0.02, 0.96)	(0.15, 0.03)	(0.22, 0.5)	(0.43, 0.58)	(0.42, 0.44)	(0.74, 0.14)	(0.2, 0.73)	(0.13, 0.5)	(0.27, 0.86)	(0.65, 0.12)	(0.10808829, 0.65071762)

	Mean rewards of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**a**
1R-1	0,5697	0,4291	0,3087	0,5106	0,0242	0,2435	0,099	0,5921	0,086	0,926	1
1R-2	0,3624	0,2785	0,1893	0,2046	0,1638	0,3771	0,0376	0,4884	0,516	0,4442	1
1R-3	0,3423	0,6104	0,1099	0,5582	0,3536	0,735	0,331	0,3542	0,0371	0,2808	1
1R-4	0,4023	0,2846	0,4642	0,6442	0,5116	0,0906	0,1393	0,3621	0,069	0,2601	1
1R-5	0,0894	0,2543	0,1732	0,3366	0,2381	0,174	0,0157	0,1263	0,2215	0,1032	1
1R-6	0,0777	0,2321	0,3996	0,2969	0,2413	0,0102	0,3391	0,3463	0,1292	0,2203	1
1R-7	0,0663	0,1222	0,129	0,0688	0,0413	0,1625	0,0217	0,0006	0,1291	0,0702	1
1R-8	0,0168	0,049	0,0296	0,0099	0,0336	0,0476	0,025	0,0496	0,0083	0,023	1
1R-9	0,2419	0,365	0,5018	0,285	0,6969	0,0465	0,7293	0,1402	0,204	0,2944	1
1R-10	0,0009	0,0029	0,0002	0,0007	0,0033	0,0027	0,0036	0,0043	0,0031	0,0038	1

	Mean feature values of the arms										
Dataset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**θ**
1F-1	0.6	0.44	0.32	0.53	0.03	0.25	0.1	0.61	0.09	0.96	0.96575059
1F-2	0.7	0.54	0.36	0.39	0.33	0.73	0.07	0.94	0.99	0.85	0.51699138
1F-3	0.44	0.77	0.14	0.7	0.44	0.93	0.42	0.45	0.05	0.36	0.79399211
1F-4	0.61	0.43	0.7	0.98	0.77	0.14	0.21	0.56	0.11	0.4	0.65547171
1F-5	0.23	0.65	0.44	0.85	0.61	0.46	0.04	0.33	0.56	0.26	0.39025382
1F-6	0.16	0.49	0.84	0.64	0.49	0.03	0.7	0.72	0.27	0.46	0.47775207
1F-7	0.35	0.63	0.65	0.35	0.22	0.78	0.12	0.004	0.67	0.34	0.20034661
1F-8	0.31	0.96	0.56	0.19	0.67	0.97	0.49	0.95	0.2	0.45	0.05172766
1F-9	0.3	0.45	0.63	0.36	0.88	0.06	0.91	0.18	0.25	0.37	0.79630038
1F-10	0.17	0.43	0.03	0.11	0.44	0.32	0.57	0.88	0.47	0.49	0.0062966