POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

nAIk

Online Learning Applications project
Pricing and Advertising

Authors:
Valeria Amato - 10641790
Beatrice Insalata - 10708628
Emanuele Paci - 10681377
Matteo Pancini - 10656944
Andrea Riboni - 10699906

Professors:
Matteo Castiglioni
Nicola Gatti
Martino Bernasconi de Luca

Academic Year: **2022-23**

Project Overview

The Online Learning Applications 2022-2023 project explores the optimization of pricing and advertising strategies for an e-commerce website selling a product. The environment is constructed with three user classes that differ in terms of click behavior, conversion probability, and purchase rate. The objective is to maximize the reward, which is a function of clicks, conversion probability, margin, and advertising costs.
The project employs various algorithms, including UCB1, TS, GP-UCB, GP-TS, and EXP3, to learn the optimal behavior under different settings. The performance of the algorithms is evaluated through cumulative and instantaneous regret and reward plots.
In order to fulfill the previously defined requirements and obtain meaningful results, a structured methodological approach was followed. Each section was examined and discussed by group members as to achieve a complete understanding of the faced issues. Our assumptions were validated by extensive research about real-life market trends and customer behavior, while the construction of the algorithms uses as reference the information provided during the course.

Specifically, the project includes six steps:

- Step 0: Describes the parameters needed to build the simulator for the e-commerce environment.
- Step 1: Solves the pricing problem with UCB1 and TS algorithms when all users belong to a single class.
- Step 2: Solves the advertising problem with GP-UCB and GP-TS algorithms when the pricing curve is known.
- Step 3: Solves the joint pricing and advertising problem with GP-UCB and GP-TS algorithms when neither the pricing nor advertising curves are known.
- Step 4: Considers the case with three user classes and evaluates the performance of GP-UCB and GP-TS algorithms with and without context generation.
- Step 5: Deals with non-stationary pricing curves and applies the UCB1 algorithm

and two non-stationary versions of the UCB1 algorithm that exploit sliding windows or change detection tests.

- Step 6: Develops the EXP3 algorithm to handle non-stationary settings with high non-stationarity degrees and evaluates its performance on multiple phases.

Contents

Todo list

1 | Introduction

Consider a setting in which an e-commerce website sells a product and can control both the price and the advertising strategy.

1.1. Environment

1.1.1. Request

We assume that a round corresponds to one day. The users are characterized as follows.

- *Two binary features can be observed by the advertising platform, call them F1 and F2; users can be of three different classes according to these features, call them C1, C2, C3; these three classes differ in terms of:*
  - the function that expresses the number of daily clicks as the bid varies, and
  - the function that assigns the cumulative daily cost of the clicks as the bid varies.
- *The three classes (C1, C2, and C3) also distinguish in terms of purchase conversion rate. More precisely, they differ in terms of the function expressing how the conversion probability varies as the price varies.*

The construction of the environment can be done as follows.

- *For every user class, specify a concave curve expressing the average dependence between the number of clicks and the bid; then add Gaussian noise to the average curve that is used whenever a sample is drawn (that is, when the number of daily clicks is drawn given a bid).*
- *For every user class, specify a concave curve expressing the average cumulative daily click cost for the bid and add a Gaussian noise over the average to draw a sample (that is, when the cumulative daily click cost is drawn given a bid).*
- *For every user class, consider 5 different possible prices and use a Bernoulli distribution for every price. This specifies whether the user buys or not the item at that specific price. A sample of the Bernoulli must be independently drawn for every user*

who landed on the e-commerce website.

The time horizon to use in the experiments is 365 rounds long.

1.1.2. Solution

The project considers three distinct user classes, each characterized by the values of the two binary features used to differentiate them. For the sake of this implementation, they'll be referred to as **C1, C2** and **C3**. As stated in the requirements, each of these classes is associated with its own curves expressing a dependency with respect to the bid value, which is the amount placed by the advertiser considering the slot to acquire. The two requested curves show respectively the direction of the click amount as the bid grows and the cumulative costs of the associated bid.

Advertising Curves

These curves were defined by considering real-life scenarios and knowledge about the observed trends. They're expressed by mathematical formulas, which vary with respect to the user class and the bid. Our bid range was selected to span values from (0.01,3), which was seen as a coherent interval. Considering the Clicks per Bid curve, we described them as:

- **C1:** $(1.0 - e^{(-5.0*bid)}) * 200$
- **C2:** $(1.0 - e^{(-5.0*bid)}) * 100$
- **C3:** $(1.0 - e^{(-5.0*bid)}) * 50$

While for the Cost per Bid, a more elaborate definition was given, to better capture the possible trend of the function, multiplying a sublinear growth by the actual shape of the click function of each specific class. This choice was made to reflect the possible highest costs that come with respect to the value of the ad:

- **C:** $\ln(bid + 1) * Clicks$

Gaussian noise is added whenever the Environment executes a round over the advertising process, in order to better reflect realistic situations (respectively with $\sigma^2 = 50$ and $\sigma^2 = 10$ for clicks and costs). The following picture shows the classes' curves. As we can observe, the **C1** class obtains the best results in terms of clicks on the ad, considering the relevance of the offer for its users, while **C2, C3** are visibly less promising.

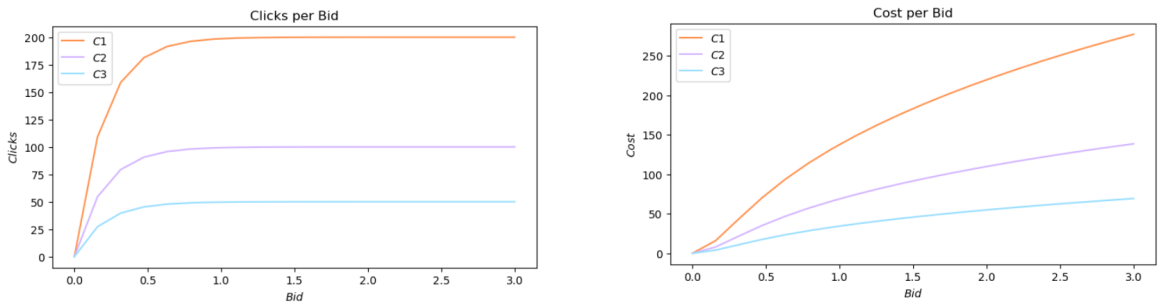

Figure 1.1: Clicks and Cost variation considering Bids

Pricing Curves

The pricing curves were defined considering important coherence aspects of the product. We chose to select 5 possible prices in the form of: [50, 100, 150, 200, 250]. In this way, we associated to each class with a conversion probability, which defines the probability that the user will actually purchase the product. To highlight the different characteristics, the conversion rates were selected as:

- **C1:** $[0.25, 0.35, 0.25, 0.15, 0.10]$
- **C2:** $[0.15, 0.25, 0.35, 0.30, 0.25]$
- **C3:** $[0.35, 0.30, 0.20, 0.15, 0.10]$

To visualize the curves, we plotted the results considering all the price values and the corresponding conversion rates, for each class.

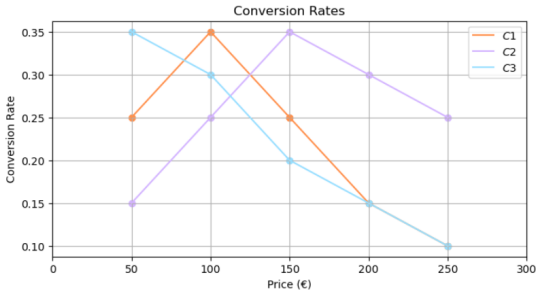

Figure 1.2: Conversion rates of users considering prices

1.2. Clairvoyant optimization algorithm

1.2.1. Request

The objective function to maximize is defined as the reward. For one class, the reward is defined as the number of daily clicks multiplied by the conversion probability multiplied by the margin minus the cumulative daily costs due to the advertising. With multiple classes of users, the reward is just the sum of the rewards provided by the single classes. The continuous set of possible bids can be approximated by a finite set of bids. In particular, the seller can choose among 100 possible bids. Given a fixed structure of contexts according to which the users are split, the optimization algorithm we suggest using is: for every single class find the best price, independently from the other classes; then optimize the bid for each class independently from the other classes. Such an algorithm requires an exhaustive search over the prices for every class and, subsequently, an exhaustive search over the bids for every class. Thus, the algorithm runs in linear time in the number of prices, bids, and contexts.

1.2.2. Solution

The Clairvoyant algorithm is a pivotal element for the functioning of the project and provides a measurement of the regret of the bandit algorithms used to solve the optimization problem. This means, in fact, finding how the reward can be maximized, and to do this, an objective function has to be defined. According to the project description we formulated it as:

$$n_{clicks}(b) * cr(p) * margin(p) - cost_c(b)$$

Where:

- $n_{clicks}(b)$ is the number of clicks obtained when considering a given bid, for a class.
- $cr(p)$ is the conversion rate for the price for a class.
- $margin(p)$ considers the fixed costs of production that define the real profitability of the product given a price, and is defined as $p - x$ where $x = (p/100) * 30$.
- $cost_c(b)$ is the previously defined cumulative daily cost when considering a bid for a class of users.

Following the suggested methodology, we conducted a search over the different possible values for the optimal price and bid considering each class, by conducting an analysis of the reward obtained by each combination. After performing the algorithm, we obtained the following results, which represented a key step in our process:

- Optimal Prices: [150, 250, 150]
- Optimal Bids: [1.1274747474747475, 1.187878787878788, 1.0670707070707073]
- Total Reward: 9502.35354550117

2 | Step 0: Motivations and Environment Design

2.1. Request

Imagine and motivate a realistic application fitting with the scenario above. Describe all the parameters needed to build the simulator.

2.2. Solution

2.2.1. The Product

In this solution, the product of interest was selected to be the **nAIk**, a brand of trekking shoes of medium quality. The choice was made considering some of the object's relevant properties when coming to specific project guidelines. Moreover, the shoe market is growing every year, with the online market being faster than the offline one.

The product has the possibility of well adapting to changes in the market from a trend,

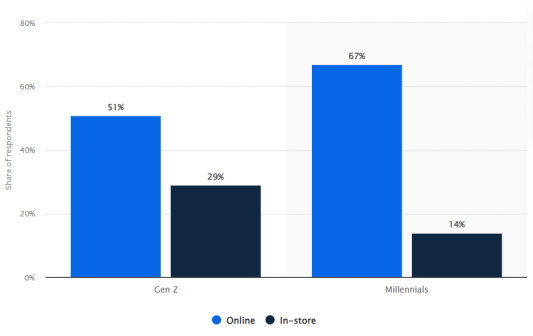

Figure 2.1: Share of Gen Z and Millennials who shop online vs offline

seasonality, and influencing factors point of view, causing smooth or abrupt variations in the demand. Also, the potential attractiveness to diverse and broad targets is another pivotal factor when validating this choice.

2.2.2. Customers

The definition of the three classes of users was made by identifying relevant and coherent binary features:

- The age range of the user, which is a factor that influences the pertinence of the selling.
- The personal interest in trekking hobbies and excursions, a key point when marketing the shoe.

Given that, we identified three specific classes of users:

1. Young users, interested in trekking, which are seen as the most appealing target, given their personal inclination and passion for outdoor workouts.
2. Middle-aged users, interested in trekking, with limitations over the potential product attractiveness due to their lower level of activity.
3. Young users, not interested in trekking, but who still retain potential value considering the exploitation of fashion and occasional use for excursions or other similar activities.

Figure 2.2: Infographic of target classes

3 | Step 1: Learning for Pricing

3.1. Request

Consider the case in which all the users belong to class C1. Assume that the curves related to the advertising part of the problem are known, while the curve related to the pricing problem is not. Apply the UCB1 and TS algorithms, reporting the plots of the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward.

3.2. Solution

The goal of the advertising problem is to maximize the number of clicks and minimize the cumulative cost. The solution to this problem is known, and it involves choosing a bid that maximizes the product between the estimated conversion rate and the margin associated to that price.

The two learners, Thompson Sampling and UCB, use this solution to the advertising problem. However, they differ in how they estimate the conversion rate. Thompson Sampling samples the conversion rate from a Beta distribution, while UCB adds a confidence bound to the empirical mean.

In each round, the learners choose the bid that maximizes the product between the estimated conversion rate and the margin, for this class of users. The margin is the difference between the price and the production cost:

$$p^* \in argmax CR_p * (p - cost)$$

where CR_p is the estimated conversion rate for the arm associated with the price p. By applying the clairvoyant algorithm, we have observed that the optimal bid value is 1.127.

The assignment requires implementing the UCB1 and TS algorithms and reporting the plots of the average value and standard deviation of the cumulative regret, cumulative

reward, instantaneous regret, and instantaneous reward.

- The Advertising_Environment class is used to model the advertising environment. The class includes the generate_observations and get_total_cost methods that add Gaussian noise to the bid-click function and cost function, respectively.
- The Pricing_Environment class is responsible for modeling the pricing environment. This class includes the get_conversion_price_probability and round methods, which are used to calculate the probability of conversion and the reward for each pricing round.
- The TS_Learner class includes the pull_arm and update methods, which are used to sample a value for each arm and update the beta parameters based on the pulled arm and the reward from the environment.

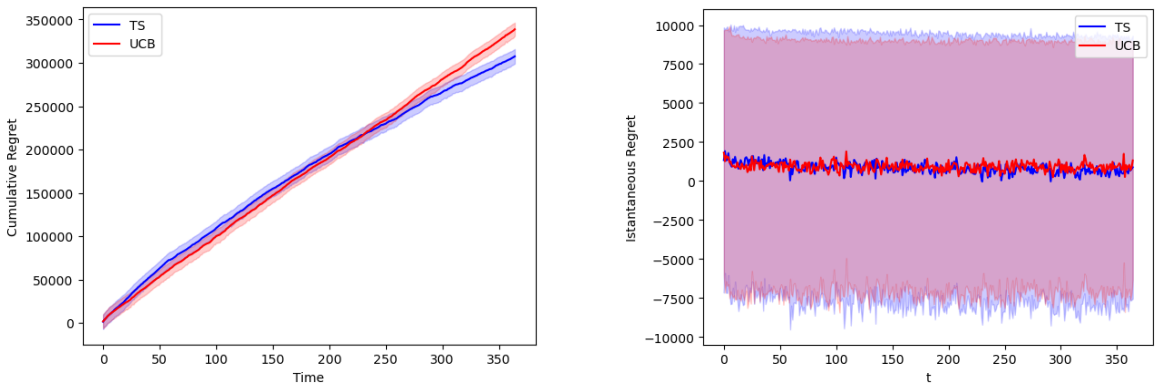

Figure 3.1: Cumulative and Instantaneous Regret

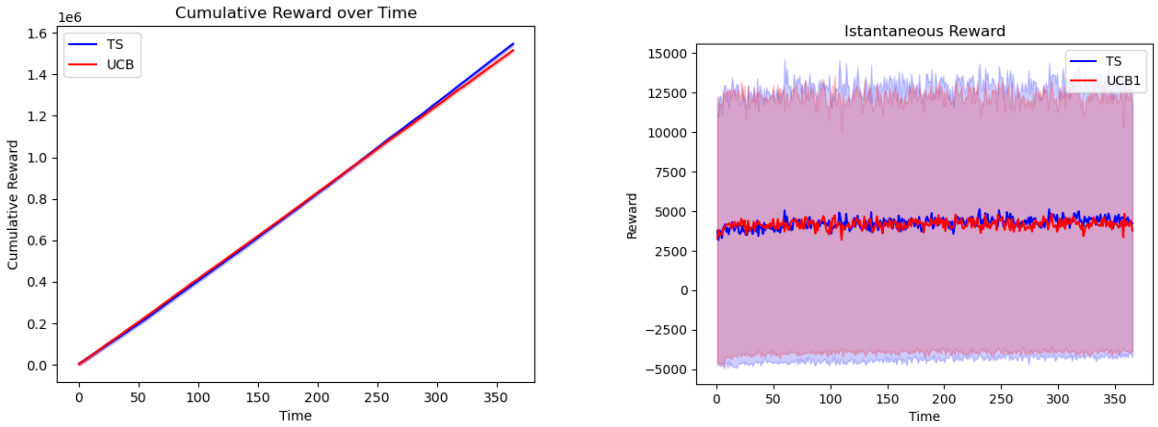

Figure 3.2: Cumulative and Instantaneous Reward

The plots show that both TS and UCB1 learners achieve similar cumulative rewards over time. However, the TS learner has a more stable reward compared to UCB1 since it has

a higher standard deviation. In addition, the TS algorithm has a lower cumulative regret than the UCB1 algorithm, which indicates that it performs better in the long run. These results are expected since the TS algorithm has a better exploration-exploitation trade-off than the UCB1 algorithm.

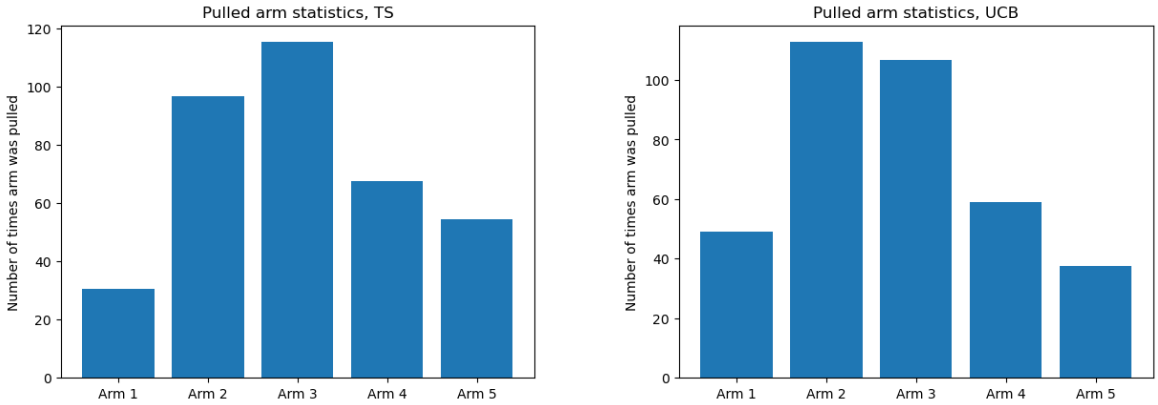

Figure 3.3: Number of times the arms have been pulled

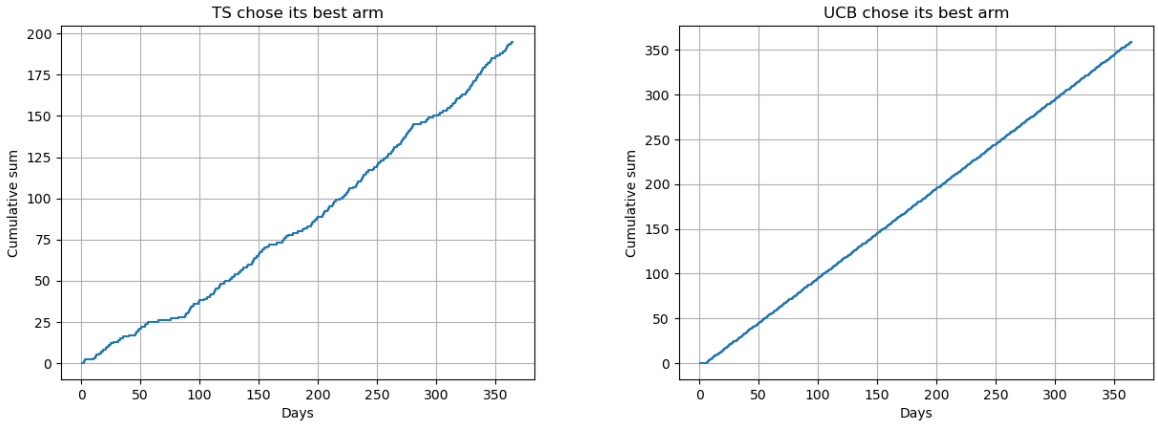

Figure 3.4: Number of times the learner pulled its *best* arm as optimal, over time

It is interesting to focus on this aspect: we observed an interesting trend when comparing the behavior of UCB1 and TS algorithms in our experiments. By plotting the number of times each arm was pulled by both algorithms (Figure 3.3), we noticed that UCB and TS favored different arms most frequently.

Furthermore, to understand how the algorithms evolved over the time horizon, we plotted the cumulative sum of the times each algorithm chose the arm that it considered optimal by the end (Figure 3.4). The results revealed that TS explored significantly more compared to UCB. Over time, TS continued to explore different arms, while UCB almost constantly selected the same arm it perceived as optimal early on.

4 | Step 2: Learning for Advertising

4.1. Request

Consider the case in which all the users belong to class C1. Assume that the curve related to the pricing problem is known while the curves related to the advertising problems are not. Apply the GP-UCB1 and GP-TS algorithms when using GPs to model the two advertising curves, reporting the plots of the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward.

4.2. Solution

Our solution for Step 2 was once again implemented through the exploitation of the TS and UCB1 learners. Given that the price curves are known, we employed the regret minimizes to select the optimal bid for the users in C1. To make it possible to apply Gaussian processes to the task, we had to state how the shape of the two functions was found to be fitting for such a technique. Since the TS algorithm used a Beta distribution in its standard form, some changes had to be applied in order to reflect the actual distribution of the observations, which are not Bernoullian anymore. Thus, to reflect the process' nature, Gaussian distributions were utilized in the algorithm to get the estimations of means and standard deviations. For the UCB1 version, the means and variances are also extracted from Gaussian distributions, even if the final uses are different.

Considering the GP-TS case, within our redefined pull_arm method, we sampled the number of clicks and cumulative cost from the Gaussian distributions based on our learner's previous estimates: the arguments self.n_click_learner and self.cumcost_learner are given from instances of the GP-TS Learner class that model the mean and variance of number of clicks and cumulative costs. In the case of GP-UCB1, the empirical means and confidences obtained from GP-UCB1 Learners are used to get an optimistic estimate

of the reward, calculating an upper bound for the arms' interval and a confidence value.

In both contexts, we then proceeded to calculate a sampled reward for each arm, which considers both click probabilities and financial information: we included the optimal price index, which represents the index of the arm that maximizes the expected reward with user-specific pricing probabilities, prices, and costs. The sampled reward calculation reflects the potential profit or loss happening in the selection of a particular arm, and it is the base for a more coherent maximization.

If we take a look at the results, we can see how the performances of the two algorithms can be comparable, given that both reach convergence.

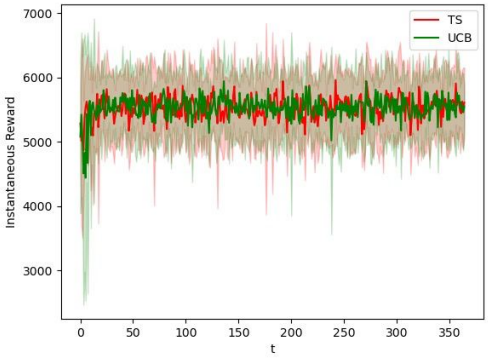

Figure 4.1:
Instantaneous Rewards

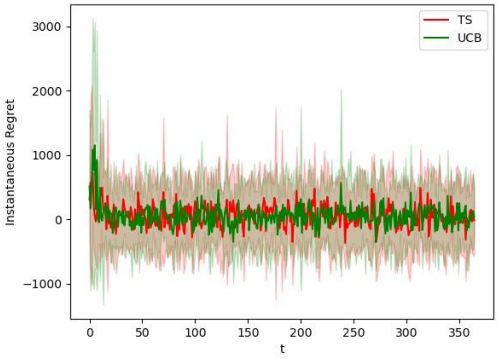

Figure 4.2:
Instantaneous Regrets

However, we can notice how the UCB1 version in the end achieves a better outcome on cumulative regret.

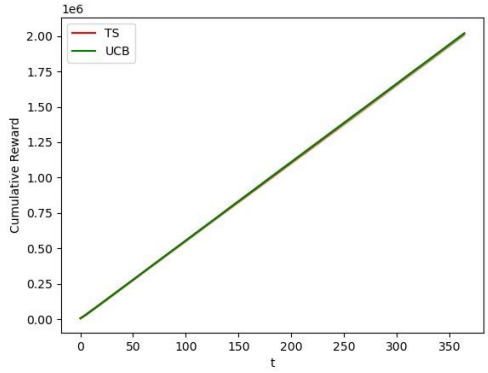

Figure 4.3:
Cumulative Rewards

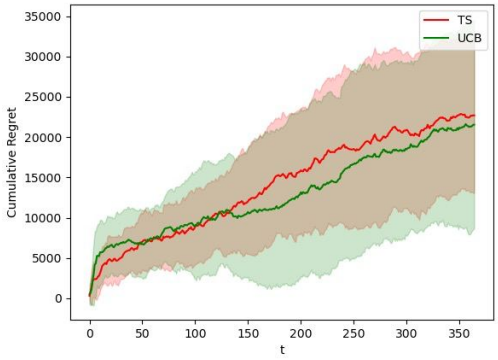

Figure 4.4:
Cumulative Regrets

This result can be attributed to a number of factors:

- Optimistic nature of the UCB1 exploration;
- Unbalanced trade-off between exploration and exploitation, since the problem is relatively simple in the number of arms and does not require extensive search;
- Great availability of the number of samples, which leads to a reduced need for complexity;
- Better ability to handle uncertainties in data and less risk-taking approach.

5 | Step 3: Learning for Joint Pricing and Advertising

5.1. Request

Consider the case in which all the users belong to class C1, and no information about the advertising and pricing curves is known beforehand. Apply the GP-UCB and GP-TS algorithms when using GPs to model the two advertising curves, reporting the plots of the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward.

5.2. Solution

In this phase, it is essential to undertake a learning process for both pricing and advertising, as the conversion rates and advertising effectiveness are currently unknown. The chosen approach involves a two-level optimization strategy. Initially, we address the pricing challenge, and once the regret minimization algorithms recommend a price, we proceed to optimize the advertising aspect using the suggested price along with the empirical conversion rate. As no specific constraints were provided for solving the pricing problem, we opted for simplicity and adopted the well-established Thompson Sampling algorithm, which had proven to be the most effective algorithm in the first step of our process.

According to the reward calculation method,

$$CR(p) * n_{clicks}(b) * (p - c) - c_{cumulative}(b)$$

it is preferable to optimise for both the best pricing and related conversion rate as well as the best bid at the same time. This suggests that two variables are being optimised simultaneously.

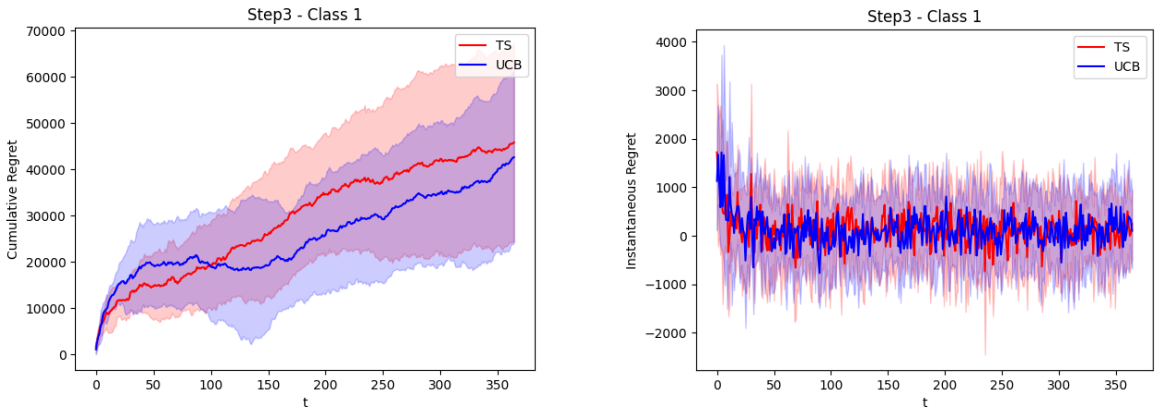

Figure 5.1: Cumulative and Instantaneous Regret

The advertising element of the reward function adds complexity because it is not certain that the reward will increase steadily with rising bids even after taking the cumulative cost into account. However, splitting the optimisation into two layers still assures that we discover the ideal solution in our particular scenario, where revenue values are high and cumulative costs are low. As a result, a two-level optimisation strategy is frequently used since it is simpler, both technically and computationally, and frequently yields the best result.

The following describes how the optimisation process works:

1. We must determine which arm to pull for the conversion rate and which bid to employ in each round. As discussed in Step 1, the Thompson Sampling (TS) learner chooses the arm with the highest draw from the linked Beta distributions.

2. The chosen arm provides us with CR_p, the estimated conversion rate (the drawn value), and p_d, the price linked to that arm.

3. With these values, we determine the best-estimated bid, denoted as b, which maximizes the reward. Here, $n_{clicks(b)}$ and $c_{cumulative(b)}$ are functions estimated by two separate Gaussian processes.

4. The chosen pricing and bids are subsequently implemented, and the environment gives real conversion rates, click counts, and total expenses. These results are utilised to update a TS learner as well as the two Gaussian processes for the number of clicks and cumulative cost functions.

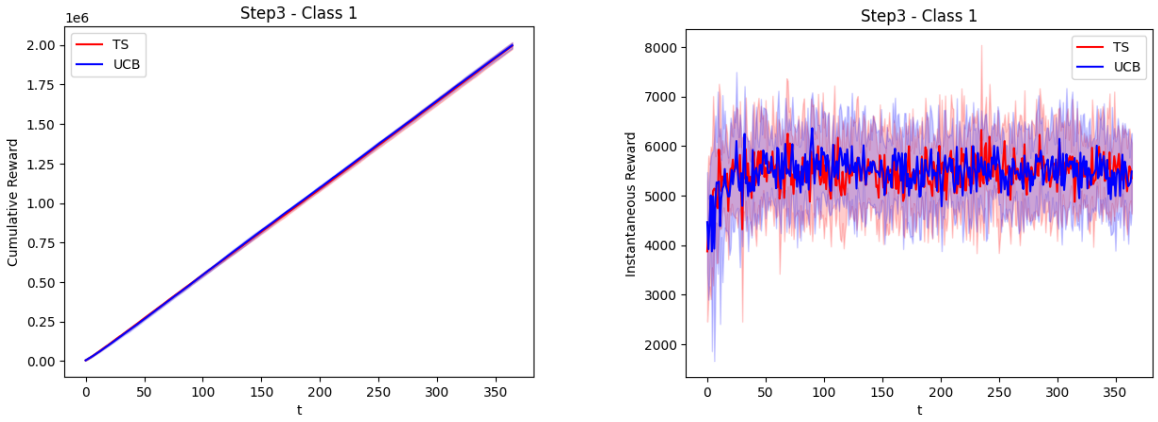

Figure 5.2: Cumulative and Instantaneous Reward

6 | Step 4: Contexts and their generation

6.1. Request

Consider the case in which there are three classes of users (C1, C2, and C3), and no information about the advertising and pricing curves is known beforehand. Consider two scenarios. In the first one, the structure of the contexts is known beforehand. Apply the GP-UCB and GP-TS algorithms when using GPs to model the two advertising curves, reporting the plots with the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward. In the second scenario, the structure of the contexts is not known beforehand and needs to be learnt from data. Important remark: the learner does not know how many contexts there are, while it can only observe the features and data associated with the features. Apply the GP-UCB and GP-TS algorithms when using GPs to model the two advertising curves paired with a context generation algorithm, reporting the plots with the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward. Apply the context generation algorithms every two weeks of the simulation. Compare the performance of the two algorithms — the one used in the first scenario with the one used in the second scenario. Furthermore, in the second scenario, run the GP-UCB and GP-TS algorithms without context generation, and therefore forcing the context to be only one for the entire time horizon, and compare their performance with the performance of the previous algorithms used for the second scenario.

6.2. Solution

Here we break down the solutions for the two scenarios, separately.

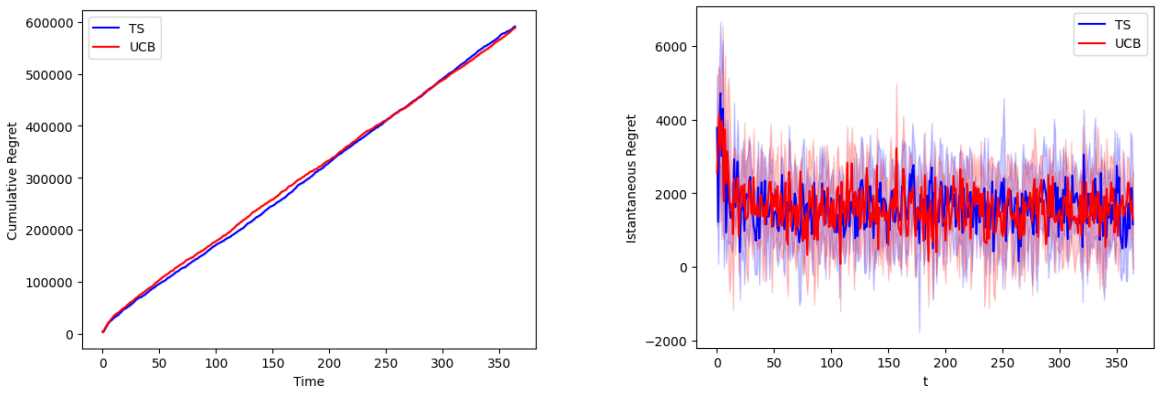

Figure 6.1: Cumulative and Instantaneous Regret

6.2.1. Scenario 1

In this step, we analyze the performance of two algorithms (GP-UCB and GP-TS) in a scenario with three user classes (C1, C2, and C3) and no prior knowledge of the advertising and pricing curves. The purpose is to learn the best pricing and two advertising curves for each class, with the goal of maximizing cumulative reward and minimizing cumulative regret.

We treat each class independently and assign a different learner to each of them. In each round, we retrieve both a price and a bid for each class of users.

We define the total regret as the sum of the regrets coming from the three learners.

$$R_{\text{tot}} = R_{C_1} + R_{C_2} + R_{C_3}$$

As it happens for the other steps, the environment is used to generate the rewards, based on the learners' price and bid. Being computationally heavy, we execute only five experiments over 365 days to average the findings. Considering the previous steps, we expect TS to perform slightly better than UCB1 in this setting too.

From Figure 6.1, we can observe that both GP-UCB and GP-TS achieve a slightly sub-linear regret: both algorithms are effective to learn the optimal pricing and advertising curves in a scenario where no prior information is known. The regret is minimized as the algorithms learn more about the advertising and pricing curves. Interestingly, both algorithms achieve very similar results. GP-TS has a slightly lower regret for the first 250 days, after which GP-UCB takes the lead. The reward achieved is twice the reward we had in Step 1, indicating that the algorithms have successfully learned the optimal pricing and advertising curves.

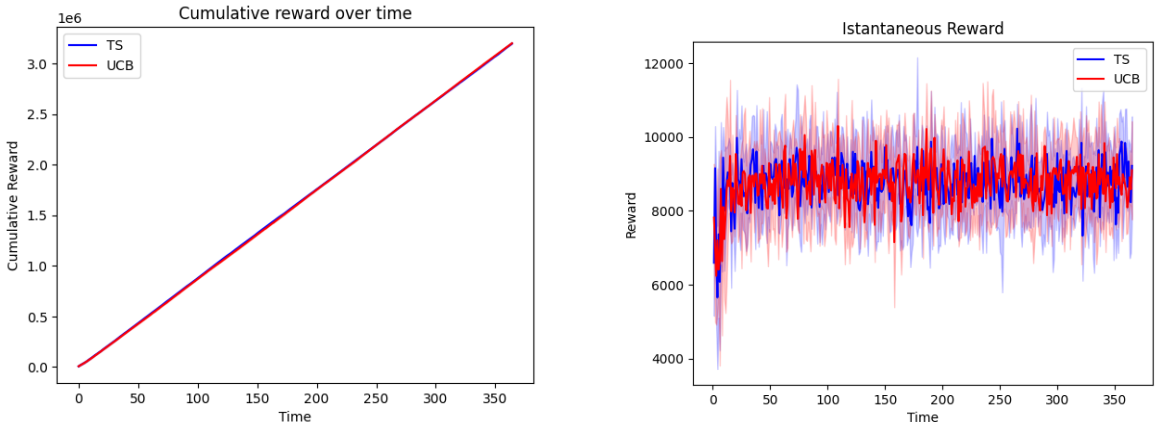

Figure 6.2: Cumulative and Instantaneous Reward

6.2.2. Scenario 2

In this scenario, the structure of the contexts is not known beforehand and needs to be learned from data. We apply the context generation algorithms every two weeks of the simulation and compare the results with respect to the same algorithms, used without context generation.

In contrast to previous scenarios, we anticipate substantial distinctions between the two versions, with a clear advantage in favor of the contextual versions. By incorporating contextual information, the algorithms should demonstrate improved performance and better adaptability to the evolving non-stationary environment, leading to more efficient decision-making and enhanced cumulative rewards.

In the beginning, we have a general aggregated model. Every two weeks, the context generation algorithm is performed: new contexts will be introduced, and some existing contexts could be removed. Every possible combination of features is explored, and for each combination, we compute a probabilistic lower bound for the reward that we would get from that context. For each context, we determine the best price and the best bid.

It is possible to explore every combination of features because we only have two binary features. As for the context generation algorithm, a feature tree is built upon the split condition

$$\underline{p}_{c_1}\underline{\mu}_{a^*_{c_1},c_1} + \underline{p}_{c_2}\underline{\mu}_{a^*_{c_2},c_2} \geq \underline{\mu}_{a^*_{c_0},c_0} \tag{6.1}$$

In particular,

- c_0 is the current context structure. In the beginning, c_0 is the aggregated model.
- c_1 and c_2 together represent the new context structure, which is obtained by splitting

on a certain feature.

- $\underline{p}_c$ is the lower bound of the probability of occurrence of a user that belongs to context c
- $\underline{\mu}_{a_c^*.c}$ is the lower bound of the reward of the optimal arm (a^*) of context c

When a feature is chosen, we decide whether it is worth splitting. We split when the lower bound of the expected reward is not smaller than the lower bound of the expected reward when we do not split.

The bounds specified in Equation 6.1 are computed using Hoeffding's bound (Equation 6.2) with 95% confidence.

$$\hat{x} - \sqrt{-\frac{\log(0.95)}{2|Z|}} \tag{6.2}$$

In general, the conversion rate is computed as

$$\text{CR} = \frac{\text{total conversions}}{\text{total visitors}}$$

Therefore, we can estimate the conversion rate for a single context as

$$CR(p) = \frac{\text{conversions}}{\text{clicks}}$$

And, applying the lower bound, we get

$$CR = \frac{\text{conversions}}{\text{clicks}} - \sqrt{-\frac{\log 0.95}{2\text{clicks}}}$$

We can determine the optimal price $\bar{p}$, given the estimated conversion rate, as

$$\bar{p} = \arg\max_p CR(p) \cdot (0.7 \cdot p)$$

Similarly, we have to estimate the two advertising parameters: the number of clicks and the cumulative cost. Since, originally, these two curves are thought to have gaussian noise in them, we consider themselves as gaussian. We refer to the general formula (where $Z = 1.96$, to get a confidence of 95%)

$$\text{Confidence interval} = \text{Sample Mean} \pm \left(Z \cdot \left(\frac{\text{Population Standard Deviation}}{\sqrt{\text{Sample Size}}} \right) \right) \tag{6.3}$$

Let's start with the number of clicks parameter. By applying Equation 6.3, we get the

click-lower-bound (CLB)

$$\text{CLB} = \frac{\text{clicks}}{t} - 1.96\frac{\sigma_{\text{clicks}}}{\sqrt{t}}$$

And the cumulative-cost-upper-bound ($CCUB$)

$$\text{CCUB} = \frac{\text{cost}}{t} + 1.96\frac{\sigma_{\text{cost}}}{\sqrt{t}}$$

This time we are considering the upper bound since the cumulative cost is going to be subtracted and not added. Moreover, like before, we can now find the optimal bid by maximizing the total reward per click

$$\bar{b} = \arg\max_{p} \frac{1}{CLB_b} \left(CR_{\bar{p}} CLB_b(\bar{p} \cdot 0.7) - CCUB_b\right)$$

Finally, given the optimal price and the optimal bid, we compute the context reward as

$$\underline{\mu}_{a^*_c,c} = \frac{1}{CLB_{\bar{b}}}(CR_{\bar{p}} CLB_b(\bar{p} \cdot 0.7) - CCUB_{\bar{b}})$$

One last parameter needs to be analyzed regarding Equation 6.1, which is $\underline{p}_c$. This lower bound on the probability (specific to a context) is computed as

$$\underline{p}_c = \frac{\text{clicks}_c}{\sum_{i \neq c} \text{clicks}_i} - \sqrt{-\frac{\log 0.95}{2\text{clicks}_c}}$$

When the contexts are computed, each one is assigned a learner. For already existing contexts, the learner remains the same.

We run the GP-UCB and GP-TS algorithms with and without context generation over a sufficiently large number of runs, computing the average and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward.

We observe that both GP-UCB and GP-TS achieve a slightly sublinear regret in both the contextual and non-contextual versions. The regret decreases as the algorithms learn more about the environment. Interestingly, in the contextual version, GP-TS outperforms GP-UCB, while in the version with no context generation, GP-UCB outperforms GP-TS. The difference in performance is not significant between the two versions.

The average cumulative reward achieved is five times the reward we had in STEP1, indicating that the algorithms have successfully learned the optimal pricing and advertising curves. The instantaneous regret and reward show more variability over time, with occasional spikes and dips. However, the overall trend is towards a decrease in regret and an

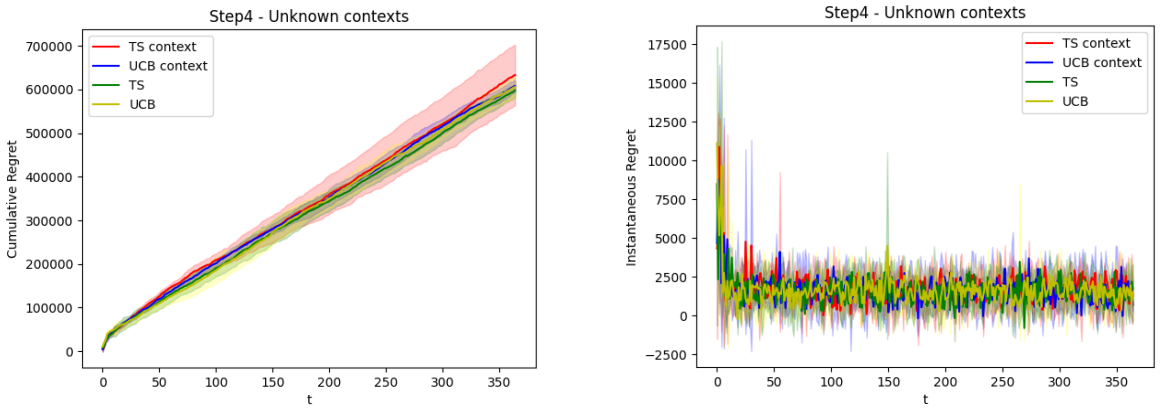

Figure 6.3: Cumulative and Instantaneous Regret No Context

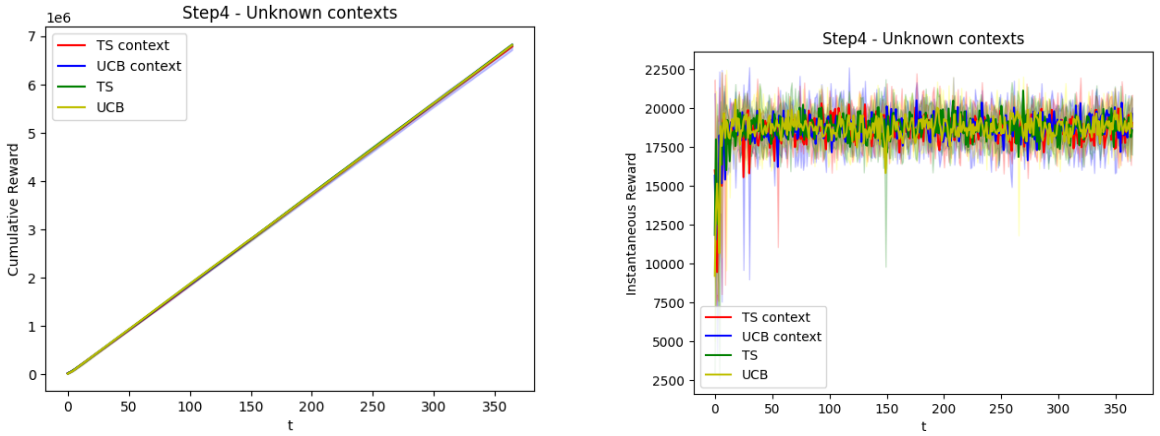

Figure 6.4: Cumulative and Instantaneous Reward No Context

increase in reward, indicating that the algorithms are learning over time.

6.2.3. Comparison

In the first scenario where the structure of the contexts was known beforehand, both GP-UCB and GP-TS algorithms achieved a slightly sublinear regret, with a very similar performance throughout the simulation. The reward obtained was twice the reward achieved in the initial stage.

In the second scenario where the structure of the contexts was not known beforehand, GP-UCB and GP-TS algorithms were paired with a context generation algorithm, applied every two weeks of the simulation. The results showed a slightly sublinear regret similar to the first scenario. However, in the contextual version, GP-TS outperformed GP-UCB, while in the version with no context generation, GP-TS had a slightly higher regret at the end of the time horizon. The reward obtained in this scenario was significantly higher

than the reward achieved in the initial stage.

Despite the differences in the algorithms' performance, the results of the two scenarios were surprisingly similar, and we expected to see more differences between the two versions.

7 | Step 5: Dealing with non-stationary environments with two abrupt changes

7.1. Request

Consider the case in which there is a single-user class C1. Assume that the curve related to the pricing problem is unknown while the curves related to the advertising problems are known. Furthermore, consider the situation in which the curves related to pricing are non-stationary, being subject to seasonal phases (3 different phases spread over the time horizon). Provide motivation for the phases. Apply the UCB1 algorithm and two non-stationary flavors of the UCB1 algorithm defined as follows. The first one is passive and exploits a sliding window, while the second one is active and exploits a change detection test. Provide a sensitivity analysis of the parameters employed in the algorithms, evaluating different values of the length of the sliding window in the first case and different values for the parameters of the change detection test in the second case. Report the plots with the average (over a sufficiently large number of runs) value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward. Compare the results of the three algorithms used.

7.2. Solution

7.2.1. Learning

In this step, since the advertising curves are known, we used a learner to learn the price of our shoe given the highest reward. For the computation of the reward we used the numbers of clicks and the total costs given by the advertising environment, i.e. the bid relative to the selected price. More or less nothing very different from what we did in step 1. As learners, we used three different versions of UCB (Upper Confidence Bound):

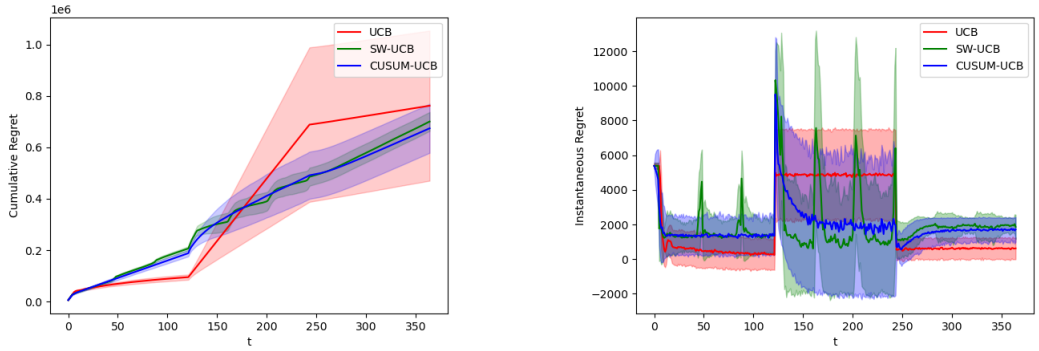

Figure 7.1: Step 5 - Cumulative and Instantaneous Regret

a standard version, a version using a sliding window and a version applying the change detection algorithm via the CUSUM mechanism.

In order to describe a scenario with 2 abrupt changes and to recreate a situation plausible for our product, we opted for the following modelling. The experiments start in about April, in a standard purchase situation, where the best price is approximately the middle (150€). Soon the winter holidays arrive, Christmas, when young people start to give presents, and for presents the purchasing power increases slightly (200€). It ends with the winter sales, when people want to take advantage of the offers, and the best products will be those around 100€.

Regarding the results in terms of regret, all 3 algorithms should achieve sub-linear regret. By algorithm structure and given the abrupt changes standard UCB should also be the one with a higher regret. Then the Sliding Window algorithm should have slightly better results, with a lower regret. Finally, UCB with change detection should have the best results of the three, since, by algorithm structure, as soon as it recognises the change of scenario it should immediately explore all arms trying to find the new possible best one. In the plots in figure 7.1 we can see the cumulative regret of the three algorithms, showing the expected results: change detection the best in terms of regret, followed by sliding window, and finally the normal UCB.

In figure 7.1 we can see the instantaneous regrets of the 3 algorithms. It is interesting to see that we have, for sliding window case, some peaks of increased regret, which could be due to worst-case arm pulls within the considered window (thus also giving us some sort of window shape). In particular, at the beginning of each window, these will have the highest upper confidence bound, and therefore, in order to undergo the algorithm's exploration, will be pulled.

Following the considerations made for regret, we can see in figure 7.2 respectively the

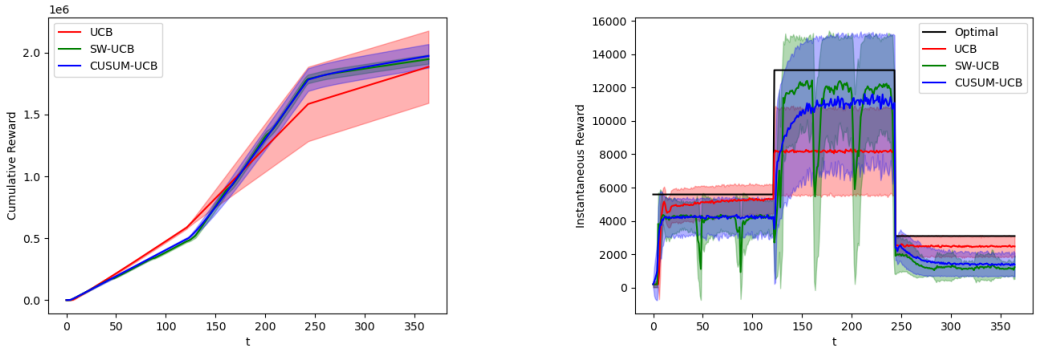

Figure 7.2: Step 5 - Cumulative and Instantaneous Reward

plot of the cumulative reward and the plot of the instantaneous reward.

7.2.2. Sensitivity Analysis

As far as the parameters for the two versions of UCB are concerned, we drew on the theoretical considerations made in lectures and some small tests, by means of experiments conducted on our dataset and scenarios, to find numbers that were meaningful for our case. For the sliding window, we opted for a value proportional to the root of the days. This is because in this way it ensures that the algorithm gives more weight to recent observations while still considering a reasonable amount of historical data. With regard to UCB with change detection, again through a few experiments, we opted for a constant value of M (Sensitivity Parameter) as well as epsilon (Threshold for Change Detection), to provide consistent and reliable change detection behaviour. For h (Confirmation Threshold) we opted for a value proportional to $\log T$ by taking into account the balance between detecting changes promptly and avoiding false positives. for alpha (Exploration Probability), on the other hand, a value proportional to $\sqrt{\frac{\log T}{T}}$, based on the desired balance between exploration and exploitation. To wrap up we used the following parameters:

$$\begin{cases} \text{window\_size} = 2 * \sqrt{T} \\ M = 25 \\ \text{eps} = 0.01 \\ h = 0.5 * \log(T) \\ \alpha = \sqrt{5 * \frac{\log(T)}{T}} \end{cases}$$

For the sensitivity analyses, we tried 5 different values for each parameter to analyse the behaviour of the algorithms, keeping the other parameters fixed and the same as those

chosen for our experiments.

SW - Window Size

The window size parameter in sliding window algorithms determines the number of recent observations or time steps considered. It is critical to the analysis's responsiveness, stability, noise sensitivity, and computing complexity. When a small window size is used, the algorithm responds quickly to short-term changes and is more sensitive to abrupt shifts or anomalies. It does, however, become more subject to noise and random oscillations, potentially resulting in less trustworthy estimations. Because of the emphasis on recent observations, the volatility of derived metrics or estimations may increase. A large window size, on the other hand, gives a smoothed and steady representation of the data by limiting the impact of short-term volatility. It aids in the removal of random noise and outliers, resulting in more robust estimates. However, because more data points or time steps are required to depict the current condition, the response to immediate changes is diminished. Rapid changes or anomalies may take longer to identify, thus slowing adaptation.

We tried respectively values:

$$window\_sizes = \{1, 2, 5, 10, 20\} * \sqrt{T}$$

On the plot in figure 7.3 we can see how medium sliding window sizes perform better then the largest and smallest ones in terms of regrets.

CUSUM - M

The parameter M in the CUSUM algorithm represents the magnitude threshold used for change detection. It determines the number of consecutive observations needed to trigger a change detection event. The choice of M influences the sensitivity and adaptability of the algorithm to detect changes in the data.

When M is set to a small number, the algorithm becomes more sensitive to small changes and can detect subtle variations in the underlying data distribution more quickly. This high sensitivity allows for rapid adaptation to changes, but it may also increase the possibility of false-positive detections, in which minor variations are mistaken for large changes.

A bigger value for M, on the other hand, provides a more conservative strategy by needing a greater magnitude of change to trigger a detection event. This minimises the likelihood of false-positive detections but may result in slower adaptability to actual data changes. A greater M is appropriate for instances when detecting significant and meaningful changes

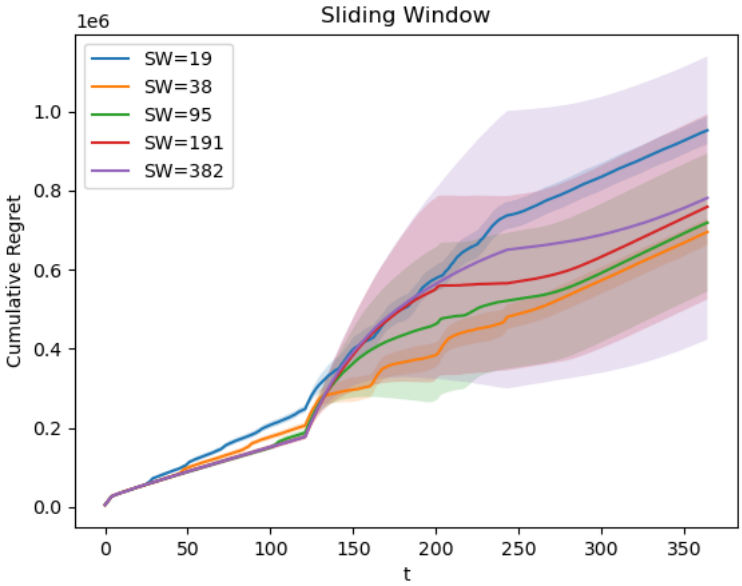

Figure 7.3: Sliding Window - Sensitivity Analysis

is critical, even if it means potentially ignoring smaller changes.

We tried respectively values:

$$M = \{1, 5, 25, 80, 200\}$$

On the plot in figure 7.4 we can see that there is no much difference between the different values (even very different) of M. This may emphasise the fact that M as a parameter has less relevance and importance than others.

CUSUM - eps

The parameter epsilon (eps) in the CUSUM algorithm represents the threshold for detecting a change in the observed data. It determines the minimum magnitude of change required to trigger a change detection event. The choice of epsilon influences the sensitivity and specificity of the algorithm in detecting changes.

When epsilon is set to a low value, the algorithm becomes extremely sensitive to even minor changes in the data. It has a high responsiveness to identify tiny shifts in observed values and trigger change detections. However, enhanced sensitivity may increase the risk of false-positive detections, in which minor variations are misidentified as substantial changes.

A bigger value for epsilon, on the other hand, provides a more conservative strategy

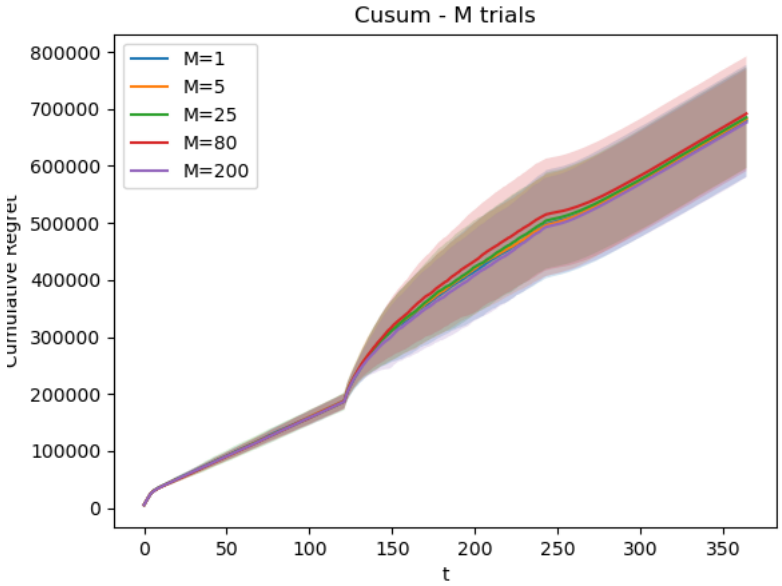

Figure 7.4: Cusum M - Sensitivity Analysis

by needing a greater magnitude of change to trigger a detection event. This minimises the likelihood of false-positive detections but may also reduce the algorithm's sensitivity to minor changes. A higher epsilon is appropriate for cases in which detecting major and meaningful changes is more crucial, even if it means potentially overlooking smaller changes.

We tried respectively values:

$$\epsilon = \{0.001, 0.01, 0.1, 0.3, 0.5\}$$

On the plot in figure 7.5 we can see that higher regret is obtained from higher epsilon values, with a decreasing regret for smaller values. It makes sense as, especially in our case, the jumps from one scenario to another are quite large, so a small value can lead the algorithm to make detection of the change.

CUSUM - h

The parameter h in the CUSUM algorithm represents the decision threshold for change detection. It determines the level of deviation required to trigger a change detection event. The choice of h influences the sensitivity and responsiveness of the algorithm to detect changes in the data.

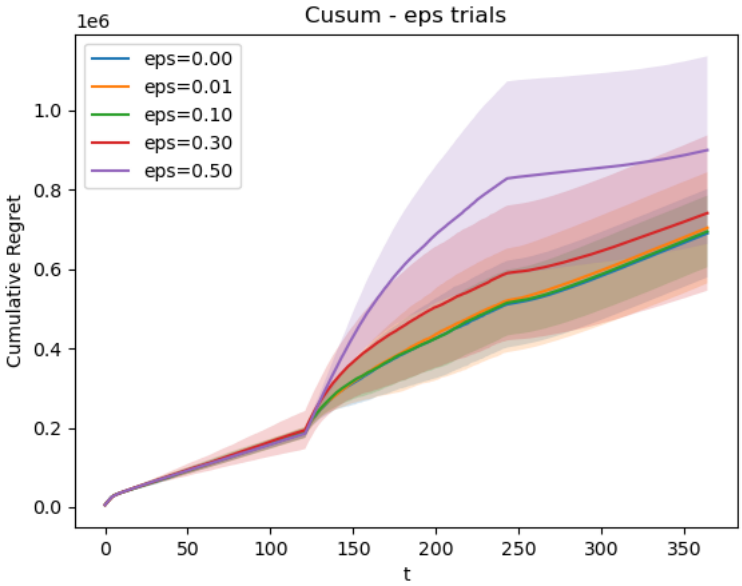

Figure 7.5: Cusum eps - Sensitivity Analysis

When h is small, the algorithm becomes extremely sensitive to departures from expected behaviour. It detects tiny changes in observed data fast and triggers change detections with great responsiveness. This great sensitivity, however, may increase the likelihood of false-positive detections, in which random fluctuations are detected as substantial changes.

A bigger value for h, on the other hand, provides a more conservative strategy by needing a higher amount of deviation to trigger a detection event. This minimises the likelihood of false-positive detections but may also reduce the algorithm's sensitivity to minor changes. A bigger h is appropriate for circumstances in which detecting significant and noteworthy deviations is more critical, even if it means potentially overlooking minor changes.

We tried respectively values:

$$h = \{0.01, 0.1, 0.5, 1, 5\} * \log T$$

On the graph in Figure 7.6, it can be seen that, as in the case of the parameter M, the values of h are all almost similar, again underlining a rather marginal relevance of the parameter.

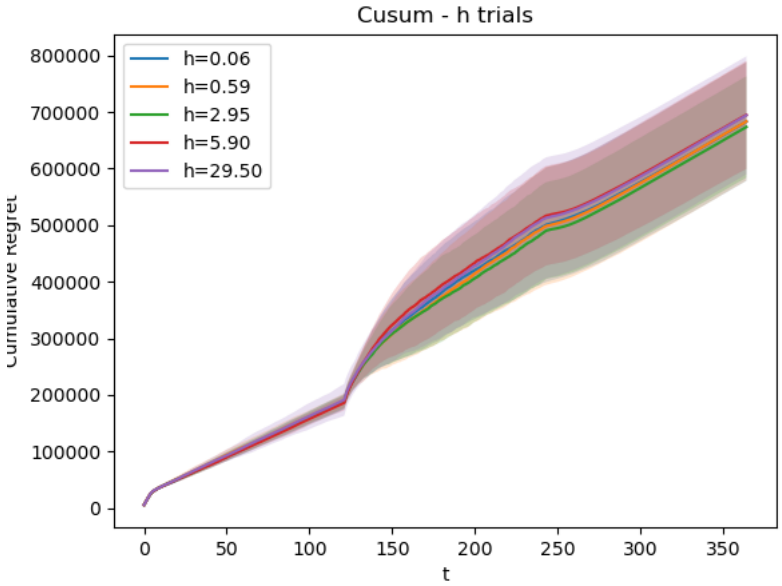

Figure 7.6: Cusum h - Sensitivity Analysis

CUSUM - alpha

The parameter alpha in the CUSUM algorithm represents the significance level or the type I error rate. It determines the threshold for considering a change as statistically significant. The choice of alpha influences the balance between the detection of true changes and the control of false-positive detections.

When alpha is set to a low value, the algorithm becomes more conservative in proclaiming a change. Stronger proof is required before a change is considered statistically significant, minimising the possibility of false-positive detections. This cautious approach, however, may reduce the algorithm's sensitivity, potentially causing it to overlook certain true changes in the data.

A higher alpha value, on the other hand, permits the algorithm to be more permissive in declaring changes. It is more sensitive at identifying possible changes, but it also increases the possibility of false-positive detections. This trade-off between sensitivity and specificity must be carefully considered in light of the individual application and the costs or implications of erroneous detections.

We tried respectively values:

$$\alpha = \{0.1, 1, 5, 10, 25\} * \sqrt{\frac{\log(T)}{T}}$$

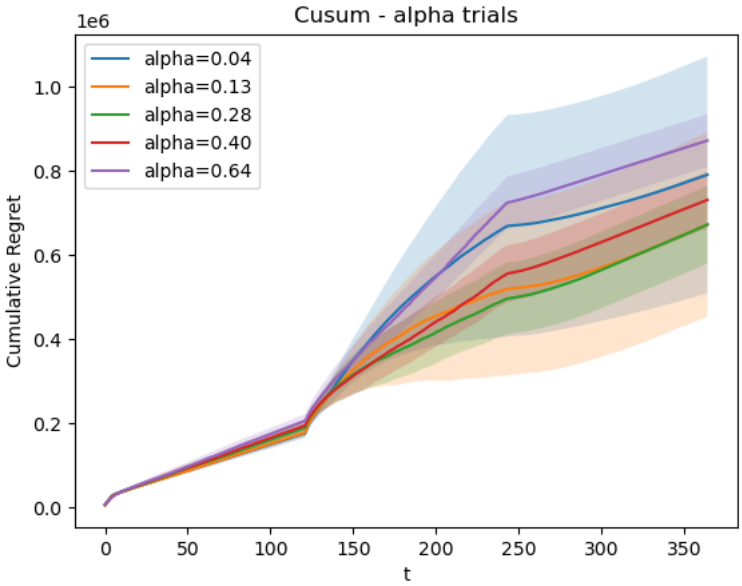

Figure 7.7: Cusum alpha - Sensitivity Analysis

On the graph in figure 7.7 it can be seen that the best values (lower regret) for alpha are those in the middle, while the regret increases for both too small and too large values. Basically, the exploration part works well when balanced, so if you explore too little, this leads to less convergence, as well as if you explore too much at random.

8 | Step 6: Dealing with non-stationary environments with many abrupt changes

8.1. Request

Develop the EXP3 algorithm, which is devoted to dealing with adversarial settings. This algorithm can be also used to deal with non-stationary settings when no information about the specific form of non-stationarity is known beforehand. Consider a simplified version of Step 5 in which the bid is fixed. First, apply the EXP3 algorithm to this setting. The expected result is that EXP3 performs worse than the two non-stationary versions of UCB1. Subsequently, consider a different non-stationary setting with a higher non-stationarity degree. Such a degree can be modeled by having a large number of phases that frequently change. In particular, consider 5 phases, each one associated with a different optimal price, and these phases cyclically change with a high frequency. In this new setting, apply EXP3, UCB1, and the two non-stationary flavors of UBC1. The expected result is that EXP3 outperforms the non-stationary version of UCB1 in this setting.

8.2. Solution

The EXP3 algorithm has been created to solve Robbins's problem: a gambler must decide which arm of K non-identical slot machines to play in a sequence of trials so as to maximize the reward without any assumptions on the statistics of the slot machines (aka conversion rate per class in our case).

The problem is an example of the trade-off between exploration and exploitation.

Exp3 stands for Exponential-weight algorithm for Exploration and Exploitation. It works by maintaining a list of weights for each of the actions, using these weights to decide randomly which action to take next, and increasing (decreasing) the relevant weights

when a payoff is good (bad).

Follows the description of the algorithm, as shown in the original paper "The non-stochastic multi-armed bandit problem":

Given $\gamma \in [0, 1]$, initialize the weights $w_i(1) = 1$ for $i = 1, ..., K.$, per each round:

1. Set $p_i = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}$ per each i
2. Draw the next action i_t randomly according to the distribution of $p_i(t)$
3. Observe reward $x_{i_t}(t) \in [0, 1]$
4. Define the estimated reward $\hat{x}_{i_t}(t)$ to be $\frac{x_{i_t}(t)}{p_{i_t}(t)}$
5. Set $w_{i_t}(t + 1) = w_{i_t}(t)e^{\gamma \frac{\hat{x}_{i_t}(t)}{K}}$
6. Set all other $w_j(t + 1) = w_j(t)$

In this step, we explored two different scenarios in which we analyzed the performance of different algorithms in non-stationary MAB problems with different degrees of non-stationarity.

8.2.1. Scenario 1

In the first scenario, we simplified Step 5 by fixing the bid and choosing the optimal bid value using the Clairvoyant Algorithm, which is 1.25€. In this case, 100 experiments have been done over a period of 365 days.

EXP3 actively explores the arms and adapts better to abrupt changes.

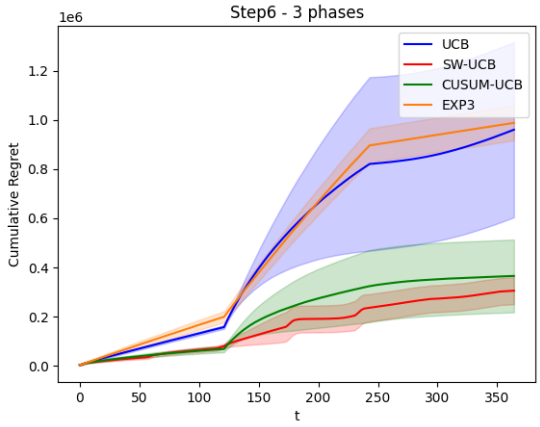

Figure 8.1: Step 6.1: Cumulative Regret

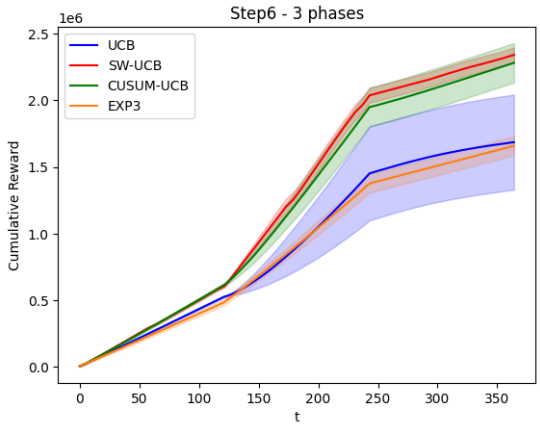

Figure 8.2: Step 6.1: Cumulative reward

The results showed that EXP3 performed worse than the two non-stationary versions of UCB1, even when tuning its gamma parameter. UCB1 learned faster due to its less exploratory behavior. Then, SW-UCB performed the best in this scenario.

As it has been shown also in the figures 8.1 and 8.2, there is no a big discrepancy between UCB1 and EXP3 curves, even if the first outperforms the second one. This is the result of a good tuning of the γ parameter, that allows a better learning.

8.2.2. Scenario 2

In the second scenario, we introduced five phases with different optimal prices that cyclically changed with a high frequency. Each phase imitates a different season or time of the year, with different conversion rates and behaviors of potential customers.

- The first phase simulated a regular period, where the best price for the shoe was 150€.
- The second phase simulated the spring, where conversion rates were lower, and the best price for the shoe was 250€.
- The third phase simulated summer, where the weather was too hot, and the conversion rates were again lower, with the best price for the shoe being 150€.
- The fourth phase simulated the sales period in autumn, where the best price for the shoe was 100€.
- Finally, the fifth phase simulated Christmas time, where higher prices were expected due to people buying gifts, and the best price for the shoe was 200€.

By including these five phases with different conversion rates and optimal prices, the

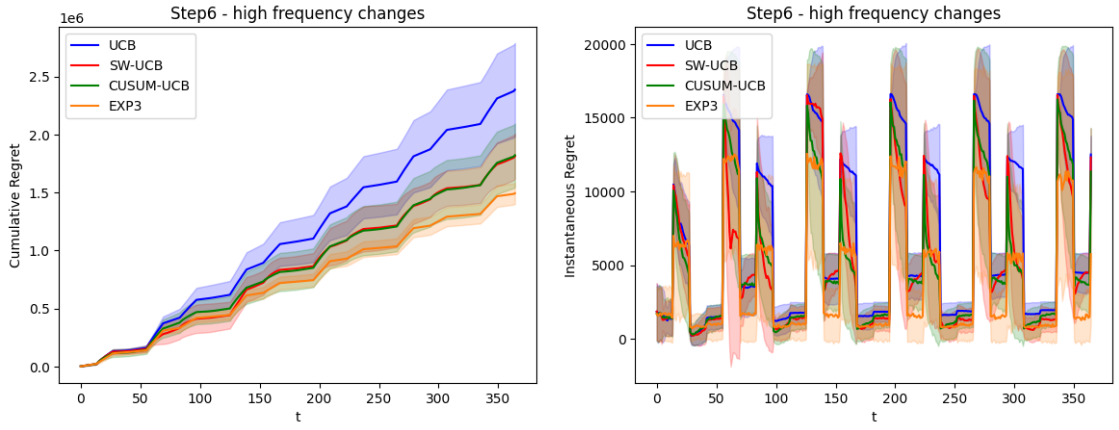

Figure 8.3: Step 6.2: Regret

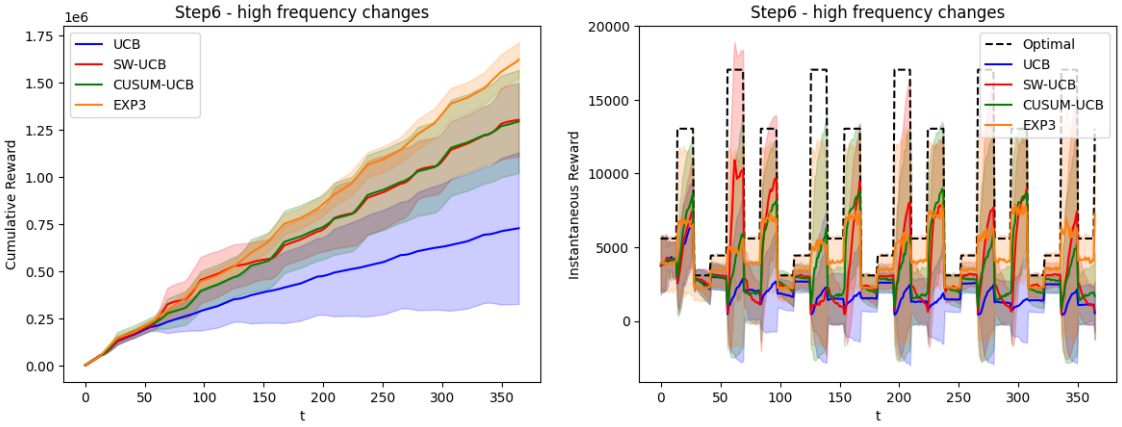

Figure 8.4: Step 6.2: Reward

scenario became more complex and realistic. It allowed us to test the algorithms' ability to adapt to different situations and perform well in highly non-stationary environments. The results showed that the EXP3 algorithm performed the best in this scenario, achieving the highest cumulative reward and the lowest cumulative regret. In the figure 8.4 is quite clear that EXP3 outperforms UCB as expected, because this algorithm better handles the unknown thanks to an exploration phase.

We used three different versions of UCB (standard, sliding window, and change detection) and EXP3 to learn the optimal price of our shoe given the highest reward. The results showed that all five algorithms achieved sub-linear regret, and EXP3 performed the best, achieving the highest cumulative reward and the lowest cumulative regret, implying that it correctly balanced exploration and exploitation. CumSumUCB had similar performances, while the other two algorithms could not achieve optimal results. The overall cumulative reward decreased compared to the previous scenario due to the various abrupt changes.

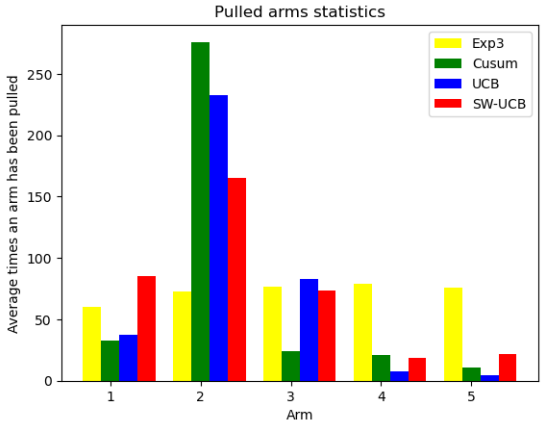

Figure 8.5: Step 6.2: Reward

The EXP3 algorithm pulled the arms almost equally, while the CumSumUCB algorithm pulled arm 2 the most, as shown in figure 8.5.