POLITECNICO
DI MILANO

Pricing and advertising project

Lorenzo Ferretti (10713719)
Nicolò Fontana (10581197)
Carlo Sgaravatti (10660072)
Marco Venere (10865088)
Enrico Zardi (10659549)

Online Learning Applications
Professors: Castiglioni Matteo, Gatti Nicola, Bernasconi de Luca Martino

A.Y. 2022/2023

Contents

1 Introduction

1.1 Request

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

1.1.1 Environment

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 Clairvoyant optimization algorithm

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 to 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.

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 Environment design

The object sold by the website is a brand watch. The prices space from 200€ to 400€ with intervals of 50€. The production cost of each piece is set to 150€.

Class C1 is represented by buyers with a middle level of income. Their conversion rate function monotonically decreases with the price except for the lowest price at which the conversion rate is very small. The rationale behind this is that selling a brand watch for a low price reduces the luxury of the item even for the middle class, while for higher prices, as one would expect, there is a contraction in demand.

In class C2, there are buyers with a medium-high level of income and their conversion rate function peaks at 300€ a higher price with respect to C1. Hence these buyers seek a quite luxurious item. While for lower prices there is the luxury effect as the cause of lower conversion rates, while for higher prices again, like for the middle-income class, conversion rates are lower.

Class C3 has buyers with the highest income level and their conversion rate increases with the price as is expected for luxury items. The lowest price still has a slightly higher conversion rate than the second price.

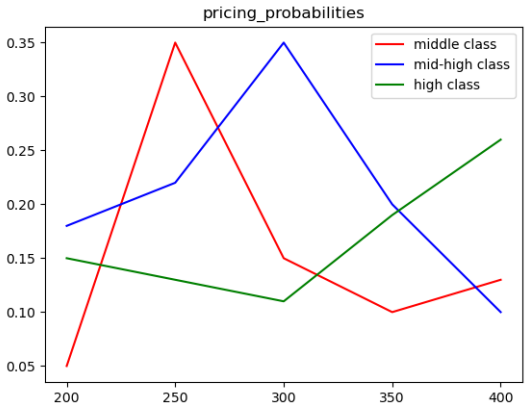


Figure 1: conversion rates per price

The functions representing the number of clicks with respect to the bid and the functions representing the cumulative cost with respect to the bid are exponential bounded functions for each class. It is natural to expect this behaviour for those functions. In fact, for a small increment of the bid with low values, the increase of the exposure of the ad will be higher than the increase caused by the same increment, but at higher values of the bid. The same behaviour is expected for the cumulative cost which is correlated with the number of clicks. Those functions present a higher limit value for lower-income classes. This is reasonable since there are more people representing those classes.

The features collected by the website for each class are the income level (medium or high) and the lifestyle (sober or fancy). The class C1 is characterized by a medium income level and it is independent of the other feature. That means it contains buyers with both fancy and sober lifestyles. The class C2 and C3 share a high income but C3, the richest, has a fancy lifestyle, that means more incline to purchase luxuries.

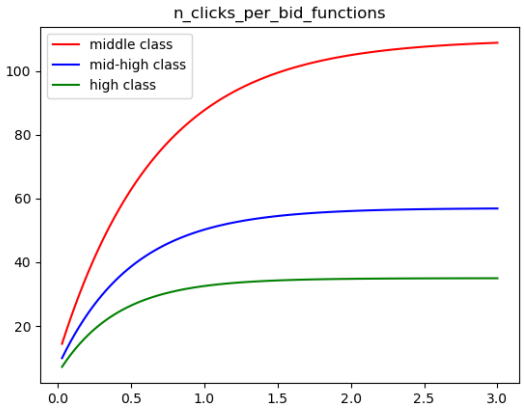


Figure 2: number of clicks per bid

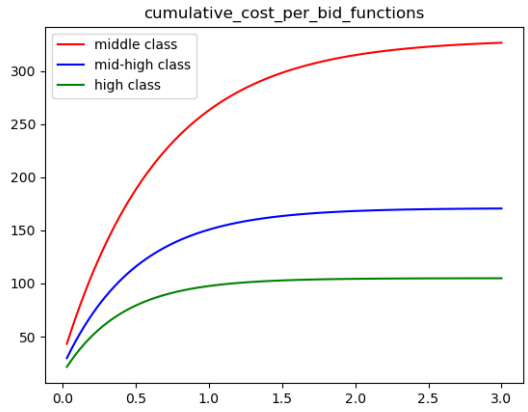


Figure 3: cumulative cost per bid

2.2.2 Technicalities

As mentioned before, the proposed prices in € are [200, 250, 300, 350, 400] with a fixed cost of production of 150€.

Each class of customers is associated with a conversion rate for each price representing their buying behaviour after seeing an ad:

- C1: [0.18, 0.22, 0.35, 0.20, 0.10]
- C2: [0.05, 0.30, 0.20, 0.15, 0.25]
- C3: [0.15, 0.13, 0.11, 0.19, 0.28]

The bids space from 0.03€ to 3.00€ with intervals of 3 cents, therefore 100 possible bids to choose from.

Each class of customers is associated with a function representing the number of clicks with respect to the ad value (depending on the value of the bid):

- C1: $(1 - e^{-1.5bid}) \cdot 100 + 10$
- C2: $(1 - e^{-2bid}) \cdot 50 + 7$
- C3: $(1 - e^{-2.5bid}) \cdot 30 + 5$

During the simulation of the environment to these functions is added, when needed, white Gaussian noise with $\sigma^2 = 3.0$

Each class of customers is associated with a function representing the cumulative cost to be paid (depending on the value of the bid):

- C1: $(1 - e^{-1.5bid}) \cdot 300 + 30$
- C2: $(1 - e^{-2bid}) \cdot 150 + 21$
- C3: $(1 - e^{-2.5bid}) \cdot 90 + 15$

During the simulation of the environment to these functions is added, when needed, white Gaussian noise with $\sigma^2 = 10.0$.

According to the environment specification, we decided to compute the reward as

$$\mathsf{rew} = \mathsf{CR}(p) \cdot n_{\mathsf{clicks}}(b) \cdot (p - c) - c_{\mathsf{cumulative}}(b) \tag{1}$$

where p is the chosen price, $\mathsf{CR}(p)$ is the conversion rate associated with the chosen price, b is the chosen bid, $n_{\mathsf{clicks}}(b)$ is the number of clicks associated with the bid, $c_{\mathsf{cumulative}}$ is the cumulative cost associated with the bid and c is the production cost (used to compute the margin $(p - c)$).

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

Since the solution to the advertising problem is known, the bid b (to compute the reward) is chosen such that the associated number of clicks and the cumulative cost maximize the reward, given the chosen price.
At each round, the two regret minimizers determine the price to choose by maximizing the product between the estimated conversion rate (sampled from the Beta distribution in the Thompson Sampling case, or the empirical mean plus the confidence bound in the UCB case) and the margin associated to that price, defined as the difference between the price and the production cost:

$$p^* \in \arg\max \tilde{CR}_p \cdot (p - cost) \tag{2}$$

where $\tilde{CR}_p$ is the estimated conversion rate for the arm associated with the price p.
After choosing the arms, at each round, a number of samples equal to the number of clicks are observed. All these samples are used to update the estimations of the regret minimizer: at each round t, a number of positive conversions $c_{p,t}$ and a number of negative conversions $c_{n,t}$ is observed (where $c_{p,t} + c_{n,t}$ is the number of clicks of that round). Let a_t be the arm chosen at round t:

- In the Thompson sampling case, the parameters of Beta are updated as follows:

$$\alpha_{a_t} \leftarrow \alpha_{a_t} + c_{p,t} \tag{3}$$

$$\beta_{a_t} \leftarrow \beta_{a_t} + c_{n,t} \tag{4}$$

- In the UCB case, the number of samples of the arm a_t, used for the computation of the upper confidence bound, is updated as:

$$n_{a_t}(t+1) \leftarrow n_{a_t}(t) + c_{p,t} + c_{n,t} \tag{5}$$

Figure 4 summarize the results of 100 different experiments for the previously described process.

As expected, both Thompson Sampling and UCB achieved a sub-linear regret, in line with the theoretical guarantees of the methods. It is clearly visible in all four graphs, in particular in the cumulative regret plot, that Thompson Sampling performed much better than UCB, reaching the optimal solution in a smaller time than UCB and thus achieving a small regret.

The reason for this might be connected to the structure of the real conversion rates for class C1. Considering only the conversion rates, the second arm, which has a conversion rate of 0.35, seems much better than the others. However, by considering the product between the conversion rate and the margin, the second arm is still the best one, providing an expected margin of 35 €, but the fifth arm, which has a conversion rate of 0.13 and a price of 400 €, is close since it provides an expected margin of 32.5 €.

The fact that UCB uses an upper confidence bound for the estimated conversion rate makes the choice of the two arms more difficult since it requires much more exploration

due to the presence of the confidence bound: the number of samples of the second arm must not be much more than the ones of the fifth arm, otherwise, even a slightly higher confidence bound in the fifth arm could make it more promising in terms of expected margin.

On the other hand, Thompson's Sampling adopts a sampling process: as the number of samples is increased, the variance of the Beta distribution is much smaller, thus the sampling becomes much more accurate. Since at each round, hundreds of clicks (samples) are observed, it is sufficient to play an arm just for a few rounds in order to have a very low variance in the sampling process, thus making the sample usually really close to the empirical mean of the arm (which should be a good estimate of real the conversion rate, since the number of samples is high). This high precision in the sampling process makes the algorithm prefer the second arm much more times with respect to what UCB does. Figure 5 is summarizing this.

Indeed, Figure 5a shows that for UCB is much more difficult to identify the optimal arm, and only after 200 rounds it starts to play the second one more frequently.
On the other hand, Figure 5b shows that TS is able, thanks to the high number of samples in a day, to choose almost every day the optimal arm.

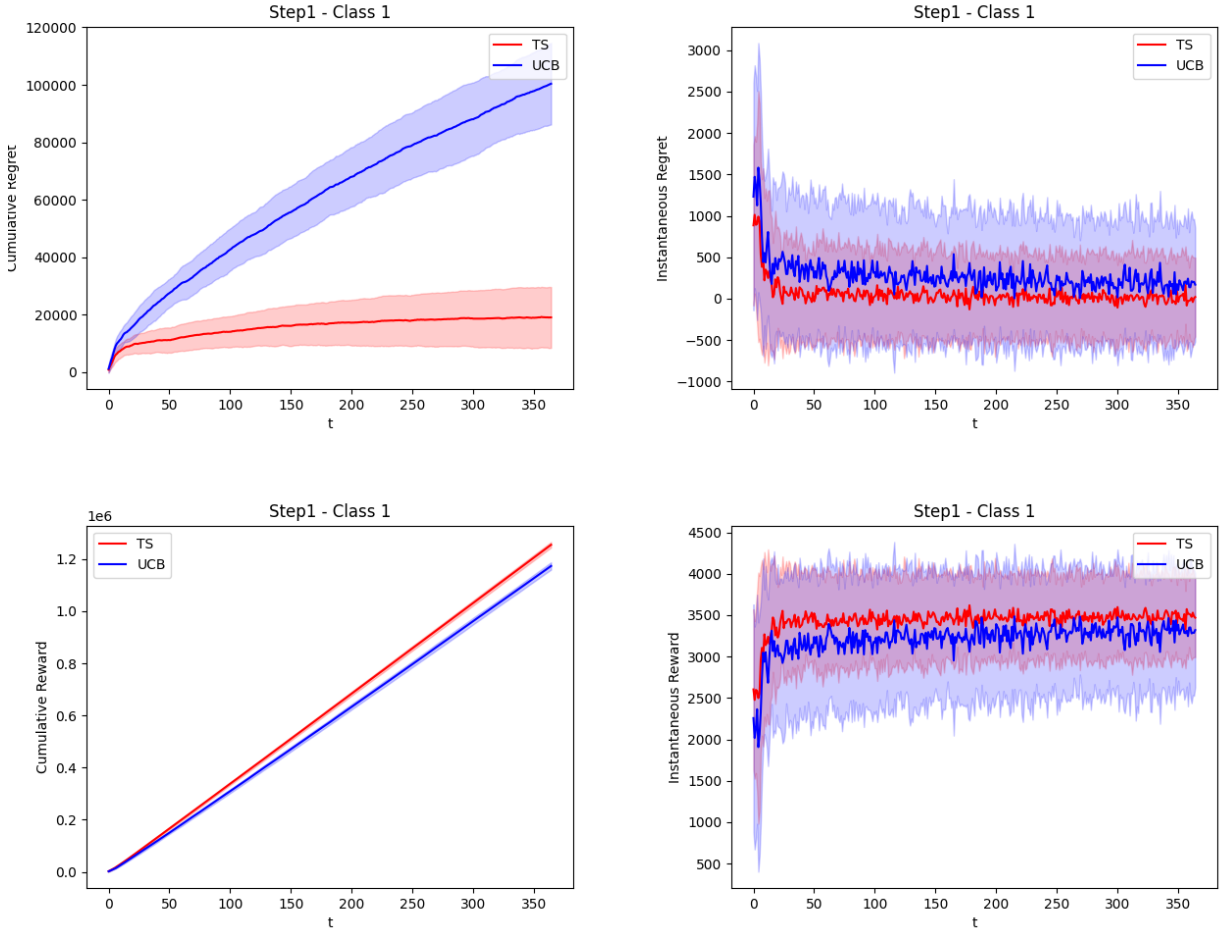
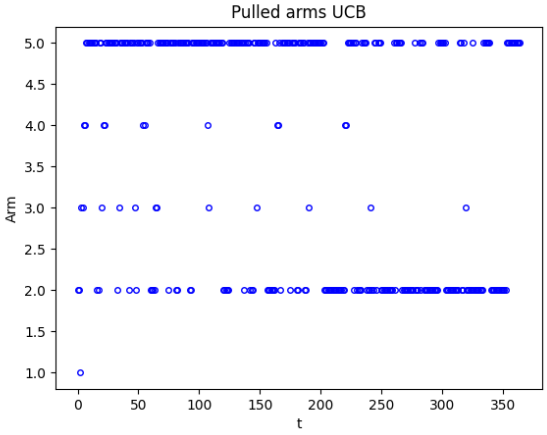
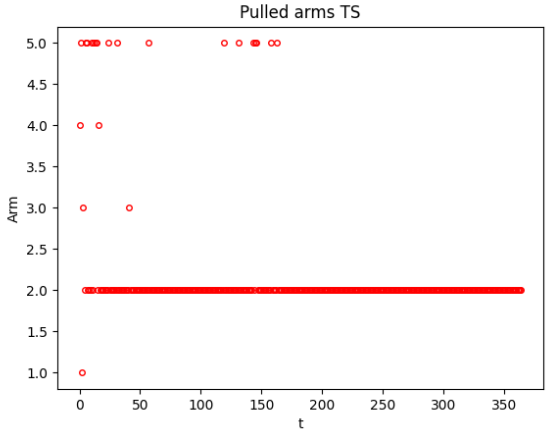


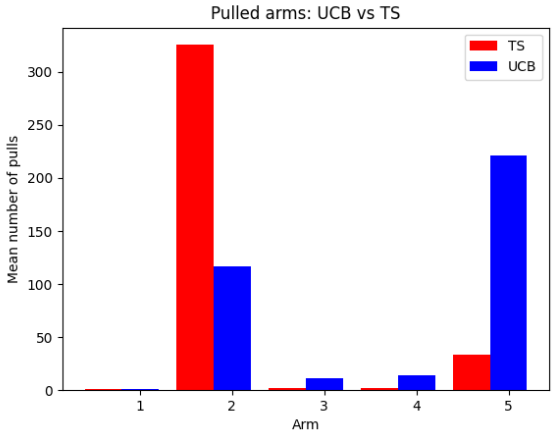
Figure 4: Step 1 results for 100 independent experiments.



(a) UCB pulled arms over time in an experiment



(b) TS pulled arms over time in an experiment



(c) Mean number of times each arm is pulled over the experiments

Figure 5: Comparison of the pulled arms of UCB and TS.

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-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****.*

4.2 Solution

This time we fixed the price to be the one giving the best result in terms of conversion rate multiplied by the margin (price minus the production cost) and used the two regret minimizers to choose the bid to maximize the reward. Both regret minimizers use Gaussian processes to store information about the functions describing, respectively, the number of clicks and the cumulative cost with respect to the bids.

In order to use the Gaussian processes we had to make some assumptions about those functions. In particular, we had to assume that there exists a correlation between arms near each other and this means that the functions describing the number of clicks and the cumulative cost must be sufficiently smooth. Since it makes sense that a small change in the bid value does not reflect a drastic variation in the placement and quality of the ad (i.e. in the number of clicks on that ad) and in the cost to be paid (i.e. in the cumulative cost), the modelization made at Step0 (section 2) with exponential curves (which are smooth) is a reasonable one, thus it is possible to use the Gaussian processes.

The main advantage of using Gaussian processes is to obtain a measure of the uncertainty over the prediction for each regret minimizer. In the UCB algorithm, this measure is used to compute the confidence bound around the predicted mean, while, in the TS algorithm, it is used as the standard deviation of the Gaussian distribution with the predicted mean.
For the TS algorithm, a Gaussian distribution is used, instead of a Beta one like in the case of the pricing problem, because now the observations on which the update is performed are not Bernoulli distributed anymore, but they represent the means used by the Gaussian processes.
Since each regret minimizer would suggest pulling an arm that maximizes the reward only with respect to the function it is learning (i.e. only with respect to the number of clicks or only the cumulative cost), while we want a bid that maximizes the reward with respect to both parameters, we decided to use an intermediate optimizer which combines the results of both minimizers and the best (known) price to maximize the reward.

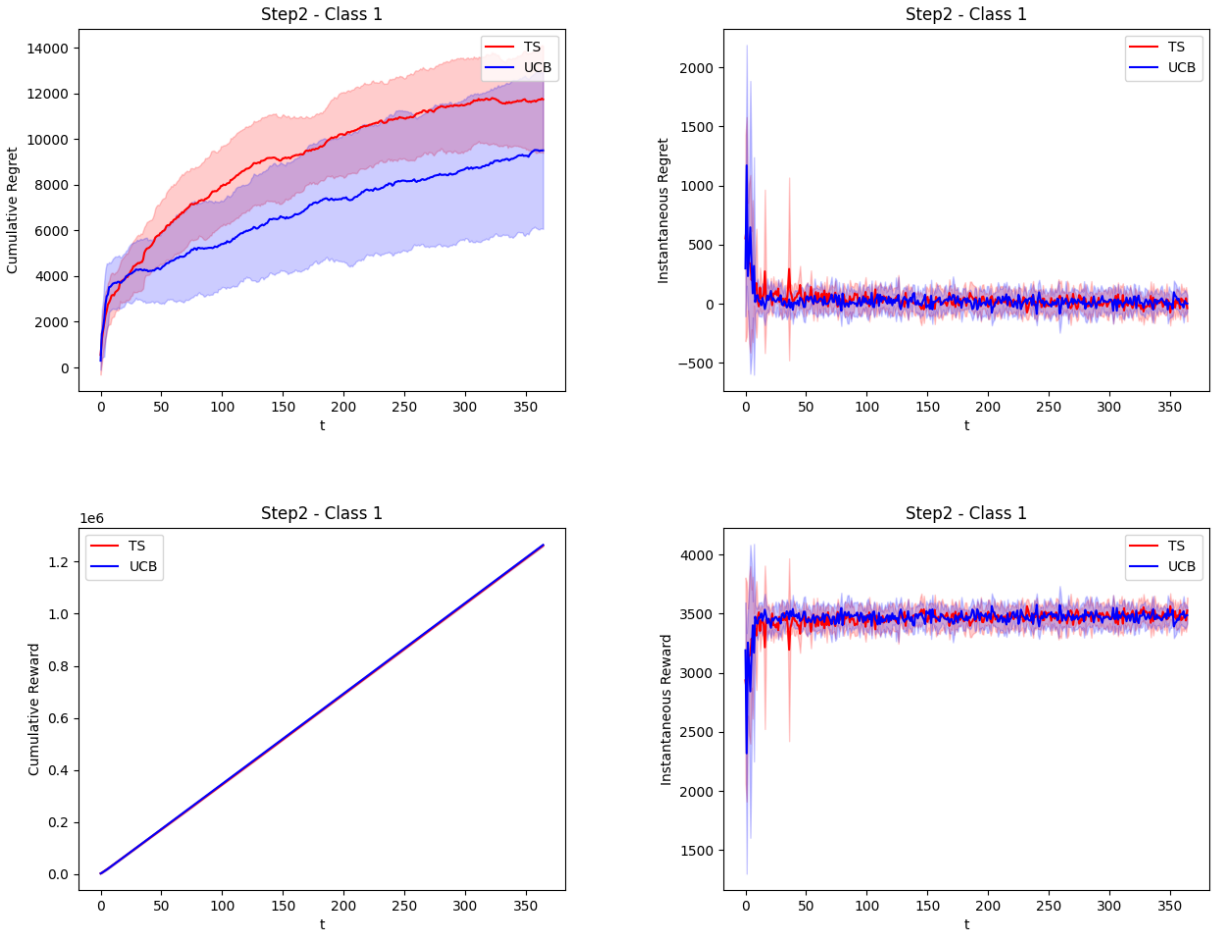
In particular, for the TS case, it uses the means and standard deviations of both regret minimizers to get the samples from the Gaussian distributions of the number of clicks and the cumulative cost, while, for the UCB case, it uses them to get the bounds. It is worth noticing that, in the UCB case, both bounds are optimistic, so the one over the number of clicks is the upper bound and the one over the cumulative cost is the lower bound.
Finally, the bid to be chosen is the one maximizing the reward with respect to the optimal price (which is known), its corresponding conversion rate and the distributions or bounds for the number of clicks and the cumulative cost.
Our expectations for the resulting performances were that GP-UCB1 would perform better than GP-TS. This is because the functions to be learned are simple ones so it is better to favour the exploitation of good arms (typical of UCB) rather than the exploration (characteristic of TS due to the sampling approach).

Usually, Gaussian processes are useful when there are many arms and few samples as they represent a good tradeoff between the accuracy of the predictions (needed to better exploit the samples) and the efficiency of the computation (needed to face the high number of arms), but in our case, this feature is not of particular utility because for each round many samples, in the order of hundreds (the number of clicks), are collected, so there is no scarcity of this kind.

Another reason why UCB can be expected to perform better than TS is that, while the exploration approach of TS is useful in non-stationary cases (which is not ours), the bound methodology of UCB is more robust to the noise that we added to the functions in the environment.

In addition, as expected, both algorithms achieve a sub-linear regret and reach convergence around regret zero in very few rounds (thanks to the high amount of samples for each of them).

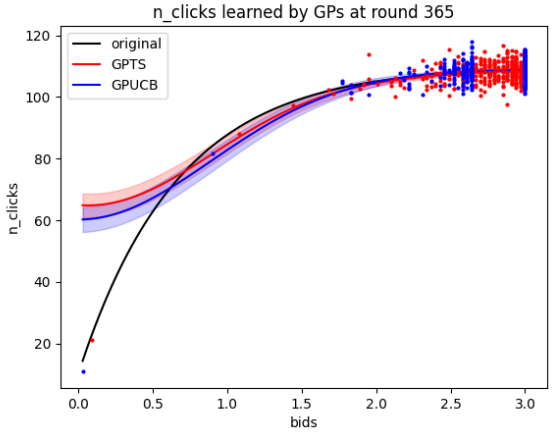
The following plots summarize the results of 10 experiments.

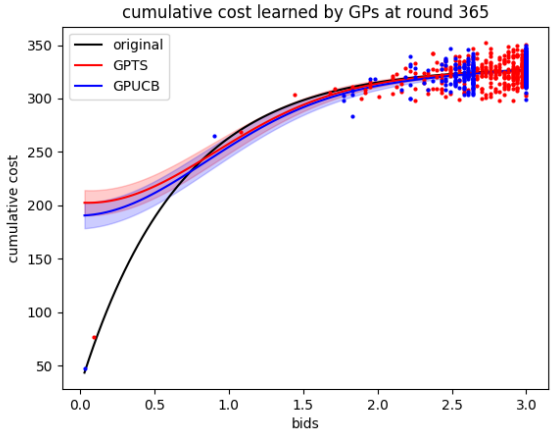


As expected, the performance of the UCB algorithm is better than the one of the TS algorithm. This is more evident in the plot of the cumulative regret, where the difference between the two is more noticeable. In the other plots, it is quite difficult to distinguish which one has better results.

Nevertheless, both algorithms achieve a sub-linear regret, in accordance with the theoretical guarantees and they do it with a high speed of convergence, which matches the assumptions made about the impact of the large amount of samples available in each round and about the simplicity of the functions to be learned.

In the next plots, it is possible to notice how for both the number of clicks and the cumulative cost the best bids are at the right end of the curve (from 2.0 and above). Also, the number of bids tried by UCB is lower than the one tried by TS, with UCB focusing more on bids around 2.6/2.7€ and the bid of 3.0€ while TS spans over the whole interval; this displays the more explorative approach of the latter algorithm.





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 step, there is a need to perform a learning procedure in both pricing and advertising since conversion rates and advertising functions are unknown.
The approach taken is to perform a two-level optimization. The first is to solve the pricing problem and once the regret minimizers suggested a price, then optimize for the advertising problem using the suggested price and the empirical conversion rate. Since no constraint was given on how to solve the pricing problem we decided, for simplicity, to adopt the classic Thompson Sampling algorithm: the best-performing algorithm in Step1 (section 3).

By recalling the formula to compute the reward: $\mathsf{CR}(p) \cdot n_{\mathsf{clicks}}(b) \cdot (p-c) - c_{\mathsf{cumulative}}(b)$ it can be noted that the optimization for the best price (and conversion rate associated) and for the best bid are in principle to be performed together. This means a joint optimization of 2 variables.
This is due to the advertising part in the reward function. Subtracting the cumulative

cost makes in principle the reward function not monotonic increasing with the bid. However in our case with large values of revenue and small cumulative cost performing a split optimization still guarantees learning the optimal solution. Therefore having two-level optimization is what is usually done in this case because it is easier mathematically, and computationally and often retrieves the optimal solution.

The optimization process is as follows.
At each round, we need to choose the best arm to pull for the conversion rate and the best bid to pull. According to the TS learner, it is selected the highest draw from the Beta distributions associated with each arm, as in Step1 (section 3).
The arm selected allows us to have $C\tilde{R}_p$, the estimated conversion rate (the drawn value) and p_d, the price associated with that arm. Then we use those values to select the best-estimated bid b^* which is the one which maximizes the reward,

$$b^* = \underset{b}{\operatorname{argmax}}\ C\tilde{R}_p \cdot n_{\mathsf{clicks_e}}(b) \cdot (p_d - c) - c_{\mathsf{cumulative_e}}(b) \tag{6}$$

where now $n_{\mathsf{clicks_e}}(b)$ and $c_{\mathsf{cumulative_e}}(b)$ are functions estimated by two different Gaussian processes. Then the prices and bids selected are played and the environment returns the effective conversion rates, number of clicks and cumulative cost which are used to upgrade a TS learner and the two Gaussian processes associated with the number of clicks and cumulative cost functions.

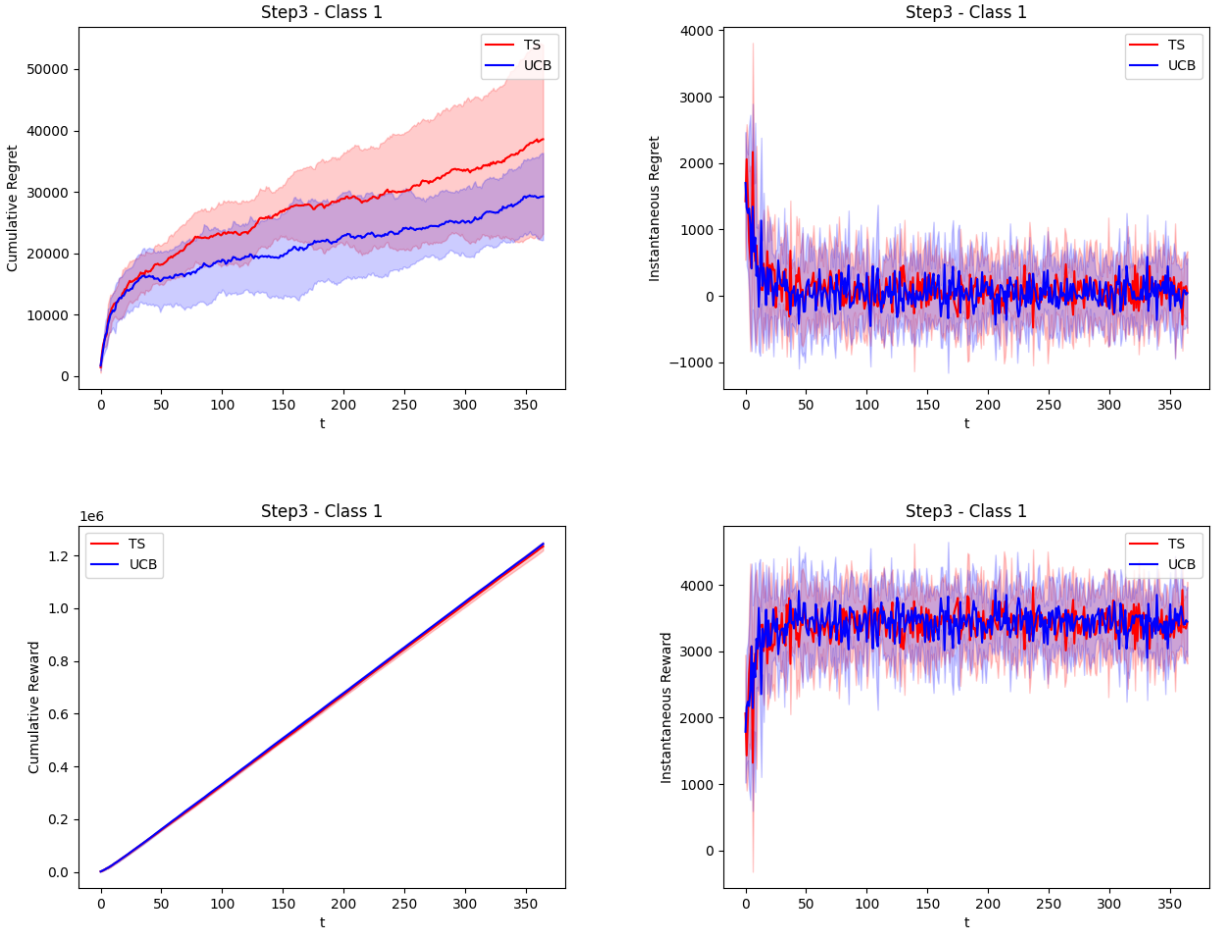


Figure 6: Step 3: Results for 10 independent experiments.

By looking at 10 simulations it can be noted that the performance of TS+GPs is comparable to the one of Step2 (section 4) where the prices were known and GP-TS and GP-UCB were used.
The instantaneous reward reaches 3500€ at the limit value as in Step2. This is expected because also in that case we used the optimal price to perform the calculations of the reward. The same can be observed for the instantaneous regret which approaches 0 as in Step2.
In this scenario, the learning of the conversion rates in addition to the advertising functions does not slow down by much the overall learning of the best joint reward.
In fact, by comparing the results of Step1, Step2 and Step3 (section 3, section 4, section 5) it can be said that for all three scenarios, the learning phase is around 20 days, for Step3 see Figure 7 .
Even for Step3 this is not a surprise because we perform a splitting optimization of the best price and of the best bid independently. As shown in Step1 and Step2 also there it takes around 20 days individually to learn the optimal values.
The variances of the results in Step3 are larger than Step1 and Step2 since there is variability in both prices and advertising.

The cumulative regret can be described as sublinear even though the lines for both TS and UCB approaches are slightly linear. The reason is that the optimal estimate is computed with the exact functions however the number of clicks and cumulative costs functions have a white noise added by the environment. Such behaviour is accepted in terms of error.

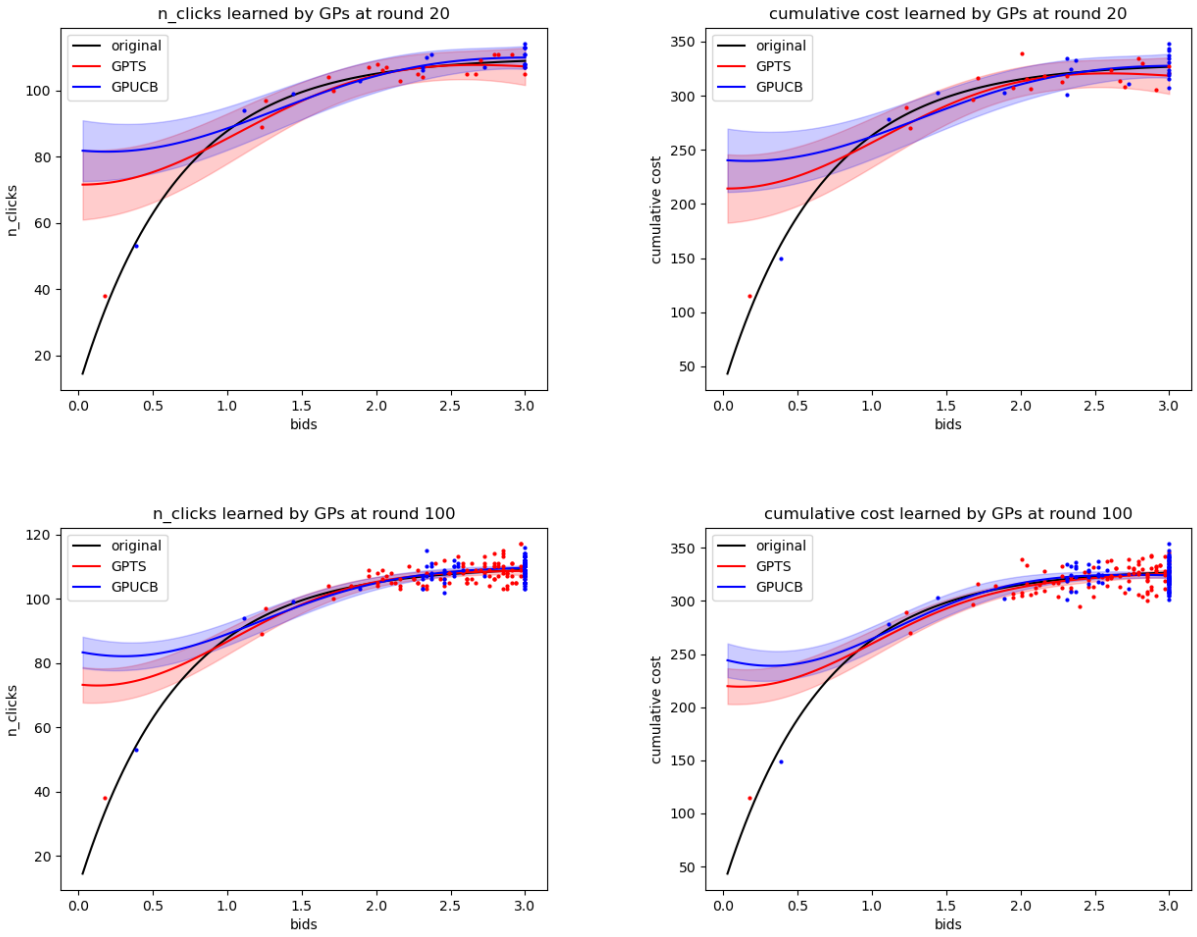


Figure 7: Step 3: number of clicks and cumulative cost learned by GPs after 20 rounds and after 100 rounds.

In Figure7 it can be seen that after 20 rounds the shapes of the curves related to the number of clicks and to the cumulative costs are captured with a reasonable accuracy. GP-TS and GP-UCB refine the approximation for values above 1.5€ of the bid, because it is there where the optimal bid of 2.79€ is found.
Therefore it is natural to have a better estimate for high values of the bid than for low values. After 100 rounds the estimate of the curve at the optimal bid is almost exact to the original, in fact, many points have been drawn in that region to be approximated.
The TS and UCB approaches construct almost the same estimates after 100 rounds. In all four graphs, it can be seen that UCB samples a lot of bids of 3€, the highest value. This is because using an upper bound selection approach can have a more robust selection of the same arm hence reducing the exploration in favour of exploitation. While TS, by

drawing from a Gaussian distribution, estimated by the process favours exploration.

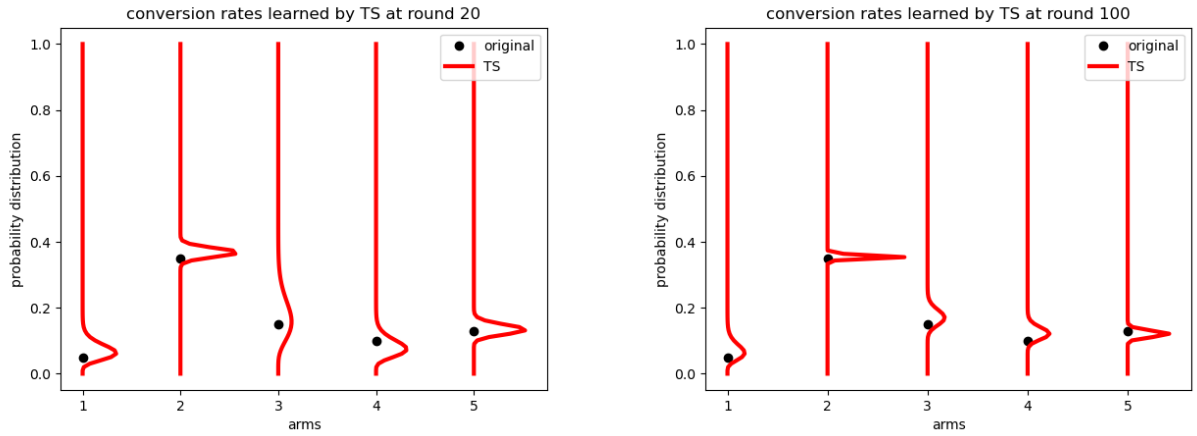


Figure 8: Step 3: conversion rates learned by TS after 20 rounds and after 100 rounds. The density probabilities of the right graph have been scaled by a factor of 2 with respect to the left.

In Figure 8 we report also the conversion rates learned after 20 and after 100 rounds. As explained in Step1 (section 3) the second and fifth arm offer by far the higher revenue. The TS after 20 rounds return an accurate prediction of the two values. After 100 rounds it returned an almost exact estimate for the two best values while there has not been much improvement in the other arms' accuracy in the probability distribution.

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

6.2.1 First scenario: known context structure

In the first scenario, since the context is fixed and known, we assigned a regret minimizer (following the same approach used in the previous step) to each class of users. This means that each class is always optimized independently, pulling at each round a price and a bid for each class of users. The regret is then defined as the sum of the regrets of the three regret minimizers.
Figure 9 is showing the results of such a process, where a sublinear regret is obtained for both GP-TS and GP-UCB. The results are in line with the ones obtained in Step 3. It is possible to notice that the regret and the reward are almost three times the ones obtained in Step 3, since here we are optimizing all three classes and summing all their regret, while previously we were considering only class C1. The figure shows that here Thompson Sampling is performing better than UCB, while in the previous two steps, it was the opposite. The reason why this happens is that optimizing classes C2 and C3 give better performances from the TS side, while for class C1 UCB performs better. Thus, the total regret for the three classes is lower for TS.

6.2.2 Second scenario: context generation

In the second scenario, we start by working on an aggregated model, by assigning each combination of features to the same context, i.e. we start with one single regret minimizer. At the end of every two weeks, we run the context generation algorithm, which will decide to build the new context structure from zero (i.e. a completely disaggregated structure can, in principle, even return to a completely aggregated structure or to something in the middle, and vice versa) based on all the data collected from the beginning. This means that the context generation algorithm will use not only the data collected in the previous two weeks but all the samples collected from time t = 0. Thus, at each round and for each combination of features, we save the pulled arms (price and bid), the number of clicks, the cumulative cost and the number of positive conversions associated with that combination of features at that round.

We have chosen to use the greedy algorithm since we have binary features. The algorithm builds a feature tree based on the following split condition:

$$\underline{p}_{c_1}\underline{\mu}_{a^*_{c_1},c_1} + \underline{p}_{c2}\underline{\mu}_{a^*_{c_2},c_2} > \underline{\mu}_{a^*_{c_0},c_0} \tag{7}$$

where c_0 is the current context structure in a node of the feature tree (i.e. the aggregated model if the node is the root of the tree) and $\{c_1, c_2\}$ is the context structure that will be obtained by splitting on a certain feature. The values of $\underline{p}_c$ and $\underline{\mu}_{a^*_c,c}$ are, respectively, the lower bound of the probability of occurrence of a user that belongs to the context c and the lower bound of the reward of the optimal arm of context c. In this case, we can consider an arm for the context as a pair of price and bid, and we can obtain the reward per click for that arm. In order to obtain the bound on the reward of the optimal arm for

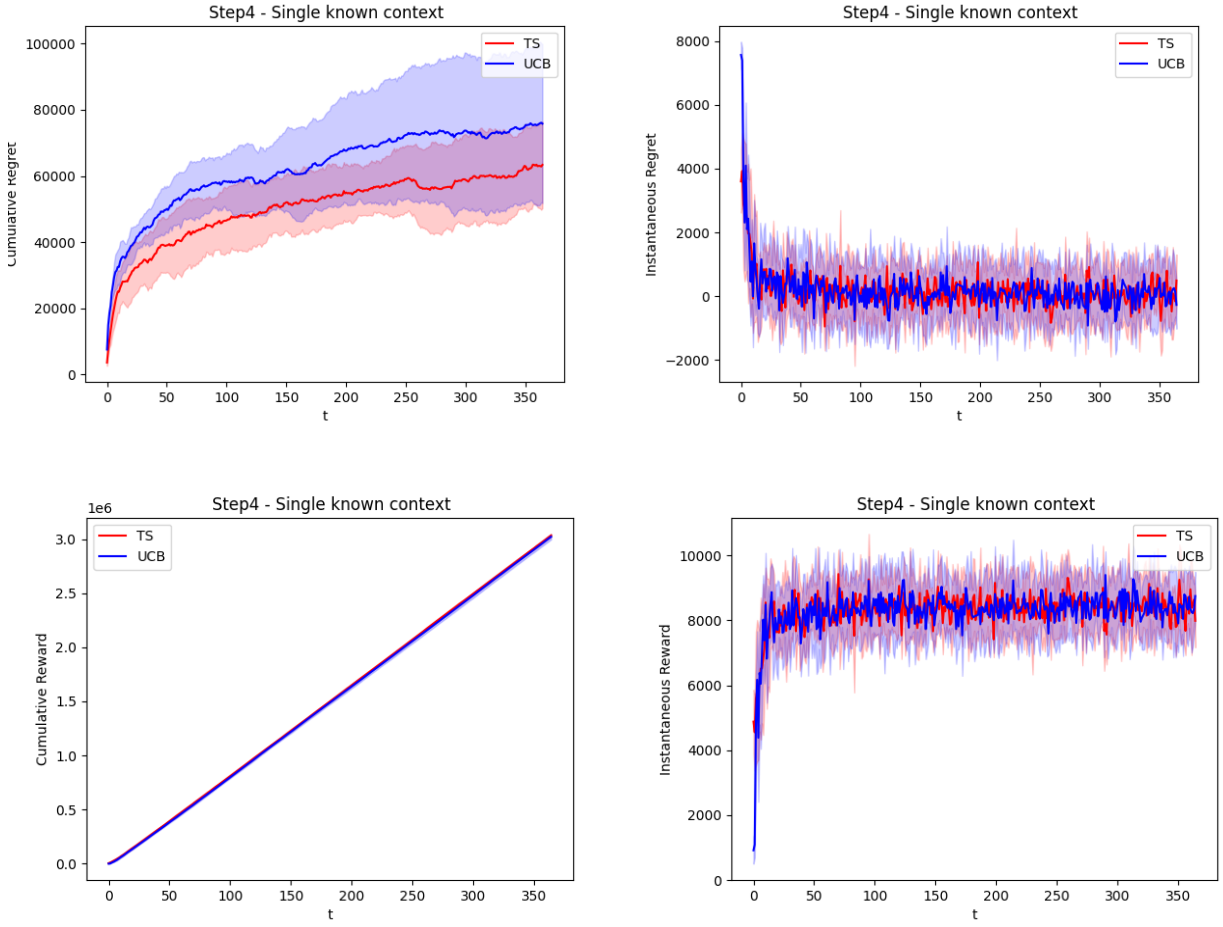


Figure 9: Results of the first scenario of Step 4.

a certain context, we reasoned with a two-step procedure by computing first the optimal price and then the optimal bid:

1. The lower bound on the conversion rate for a price p at time t for a context c can be obtained using the Hoeffding bound for a Bernoulli distribution:

$$CR_p = \frac{\sum_{k=1}^{t} nconv_{k,c}}{\sum_{k=1}^{t} nclicks_{k,c}} - \sqrt{-\frac{log(\delta)}{2\sum_{k=1}^{t} nclicks_{k,c}}} \tag{8}$$

   where $nconv_{k,c}$ and $nclicks_{k,c}$ are, respectively, the number of positive conversions and the number of clicks obtained at time k, and $\delta \in [0,1]$ is the confidence for the bound. The optimal price is then given by the one with the highest product between the price and lower bound of the conversion rate:

$$p^* \in \arg\max_p CR_p \cdot (p - cost) \tag{9}$$

2. For the bid, we have two parameters to estimate: the number of clicks and the cumulative cost. We have considered the lower bound of the number of clicks and

the upper bound of the cumulative cost (since the cumulative cost has a minus in the formula of the reward). Both are assumed to be Gaussian variables, thus we have obtained them using the bounds of 95% confidence interval for a Gaussian variable:

$$NC_b = \frac{1}{t}\sum_{k=1}^{t} nclicks_{k,c} - 1.96\frac{\sigma_{nclicks_c}}{\sqrt{t}} \tag{10}$$

$$CO_b = \frac{1}{t}\sum_{k=1}^{t} cumcost_{k,c} + 1.96\frac{\sigma_{cumcost_c}}{\sqrt{t}} \tag{11}$$

where $cumcost_{k,c}$ is the cumulative cost at time k, $\sigma_{nclicks_c}$ is the standard deviation of the number of clicks, $\sigma_{cumcost_c}$ is the standard deviation of the cumulative cost, NC_b is the lower bound of the number of clicks, and CO_b is the upper bound of the cumulative cost. The optimal bid is then defined as the one that maximises the total reward per click:

$$b^* \in \arg\max_p \frac{1}{NC_b}[CR_{p^*} NC_b(p^* - cost) - CO_b] \tag{12}$$

Given p^* and b^*, the lower bound of optimal the reward per click for a context c is given by:

$$\underline{\mu}_{a_c^*,c} = \frac{1}{NC_{b^*}}[CR_{p^*} NC_{b^*}(p^* - cost) - CO_{b^*}] \tag{13}$$

In order to compute the lower bound on the probability of the context c_1 (or c_2), we used the Hoeffding bound, where the empirical mean is defined as the fraction of clicks belonging to that context among the ones of the aggregated context c_0 (equal to the sum of the clicks belonging to c_1 and c_2):

$$\underline{p}_{c_1} = \frac{\sum_{k=1}^{t} nclicks_{k,c_1}}{\sum_{k=1}^{t} nclicks_{k,c_1} + \sum_{k=1}^{t} nclicks_{k,c_2}} - \sqrt{-\frac{log(\delta)}{2\sum_{k=1}^{t} nclicks_{k,c_1}}} \tag{14}$$

After the execution of the greedy algorithm, we assign to each context (i.e. the leaves of the feature tree) in the generated context structure a regret minimizer: if the context is new (the previous context structure didn't have it) a new regret minimizer is assigned to it, otherwise the previous regret minimizer will continue to be associated to it. If a contest is not present in the new structure its regret minimizer will be deleted. The new regret minimizers will not start from zero; indeed we initialize them with all the past samples belonging to the associated context, performing a bulk update after creating them. We do this in order to not have a useless initial exploration since we can exploit past information to initialize the estimates.

Figure 10 shows the results of the described process. In particular, it is clear the effect that the context generation algorithm has. Considering the first rounds, the cumulative regret has a very high slope and the instantaneous regret is very high. After a couple of rounds (approximately after two weeks), it is possible to notice a big increase (decrease) in the instantaneous reward (regret). This means that the context generation algorithm has found a new context structure that is better than the aggregated structure and is at least closer to the optimal one. Starting from that point, the regret is always very small, with a bit of oscillation (especially for UCB, which bounces from almost zero regret to more than 1000€ of instantaneous regret) until more or less 100 rounds (i.e. 14 weeks, equal to 7 runs of the context generation). After this period, the regret is more stable around zero, meaning that the context generation algorithm is able to converge to the

optimal context structure much more times.

The fact that the context structure is unknown makes the problem more difficult to optimize, indeed it is possible to notice that the instantaneous regret (and reward) has a much higher variance in this case, with respect to the previous scenario with a known context structure. Indeed, it might happen that even after many weeks the context generation algorithm produces a suboptimal context structure. It can also be noticed that the cumulative regret at the end of the time horizon is much larger than in the previous scenario. Indeed, here more than 100k €of regret has been obtained by TS, while previously the regret was around 60k €.

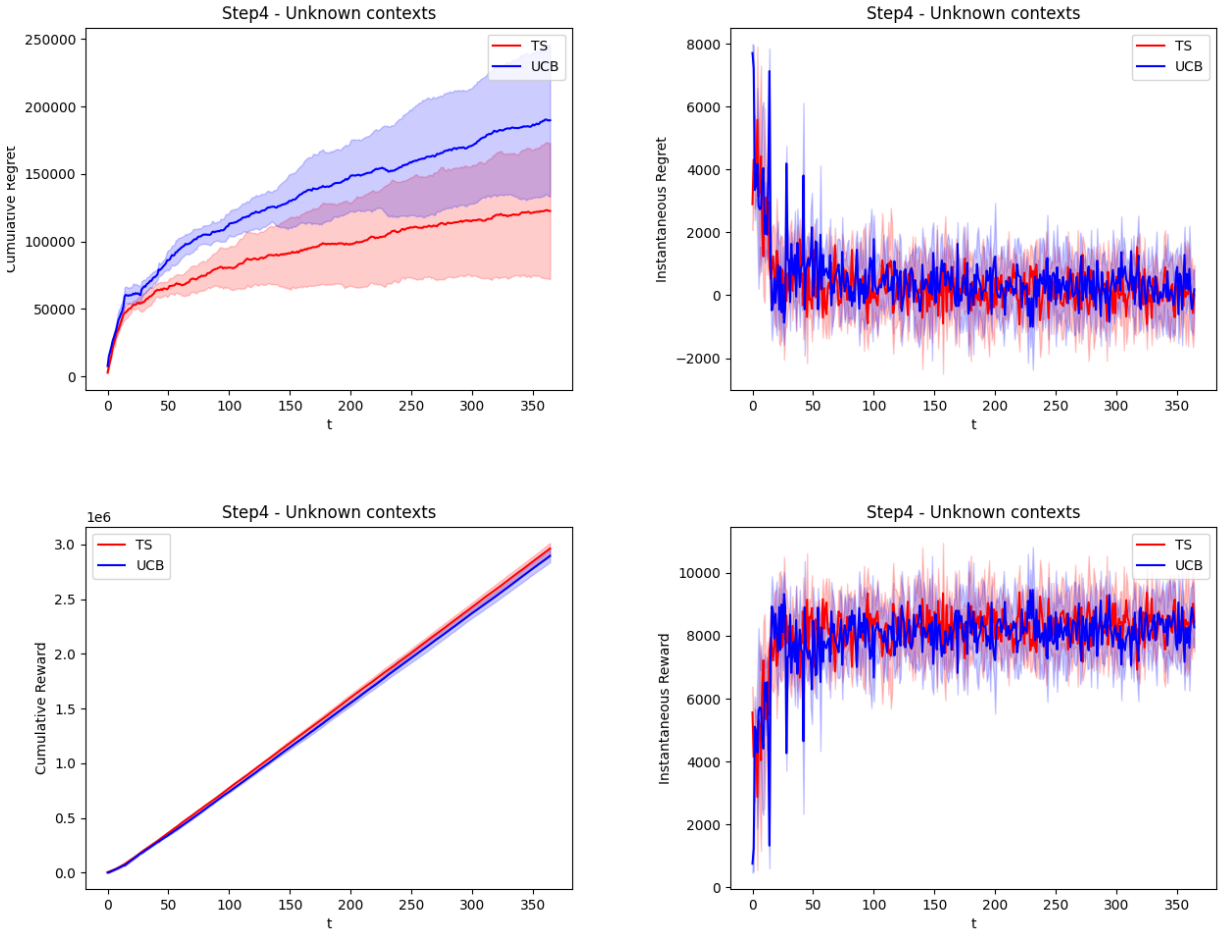


Figure 10: Results of the second scenario of Step 4.

Figure 11 is, instead, comparing the performance of the context generation with the aggregated context structure. In particular, for the aggregated context structure we have only one regret minimizer (the same as Step 3) for all the combinations of features. It is possible to observe that the cumulative regret is in this case linear. Indeed, by looking at the instantaneous regret plot, the two regret minimizers for the aggregated models converged to a constant non-zero instantaneous regret of around 2000 €per day. This happens because the optimal arms are different between classes, thus there is no arm

that permits to have a zero-regret for all types of users. From the cumulative regret plot, it can be observed that UCB has a higher variance with respect to TS. One reason for this is probably that the context generator associated with the UCB learners has selected a suboptimal context structure more frequently than TS.

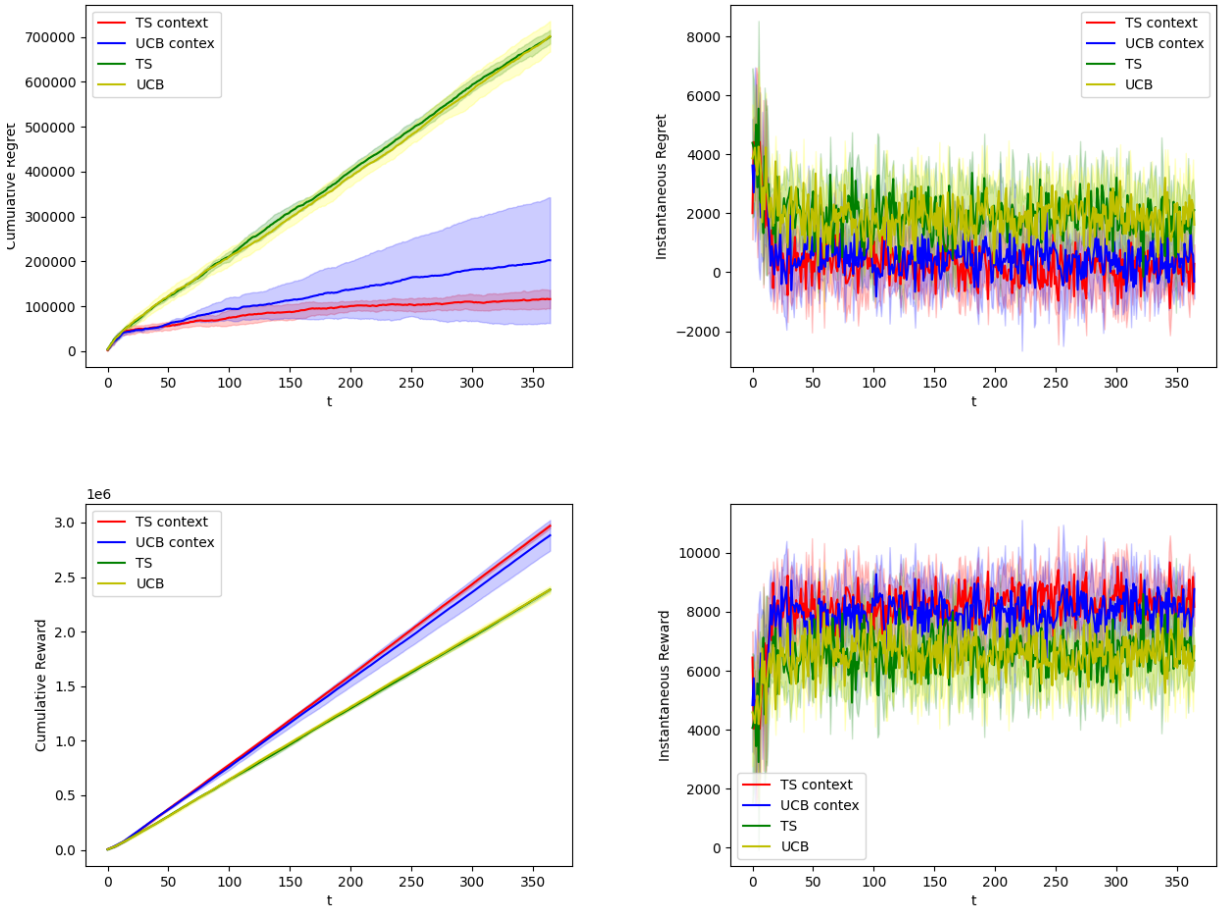


Figure 11: Comparison between the aggregated context structure and the context generation. The curves labelled as "context" refer to the solution with context generation.

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

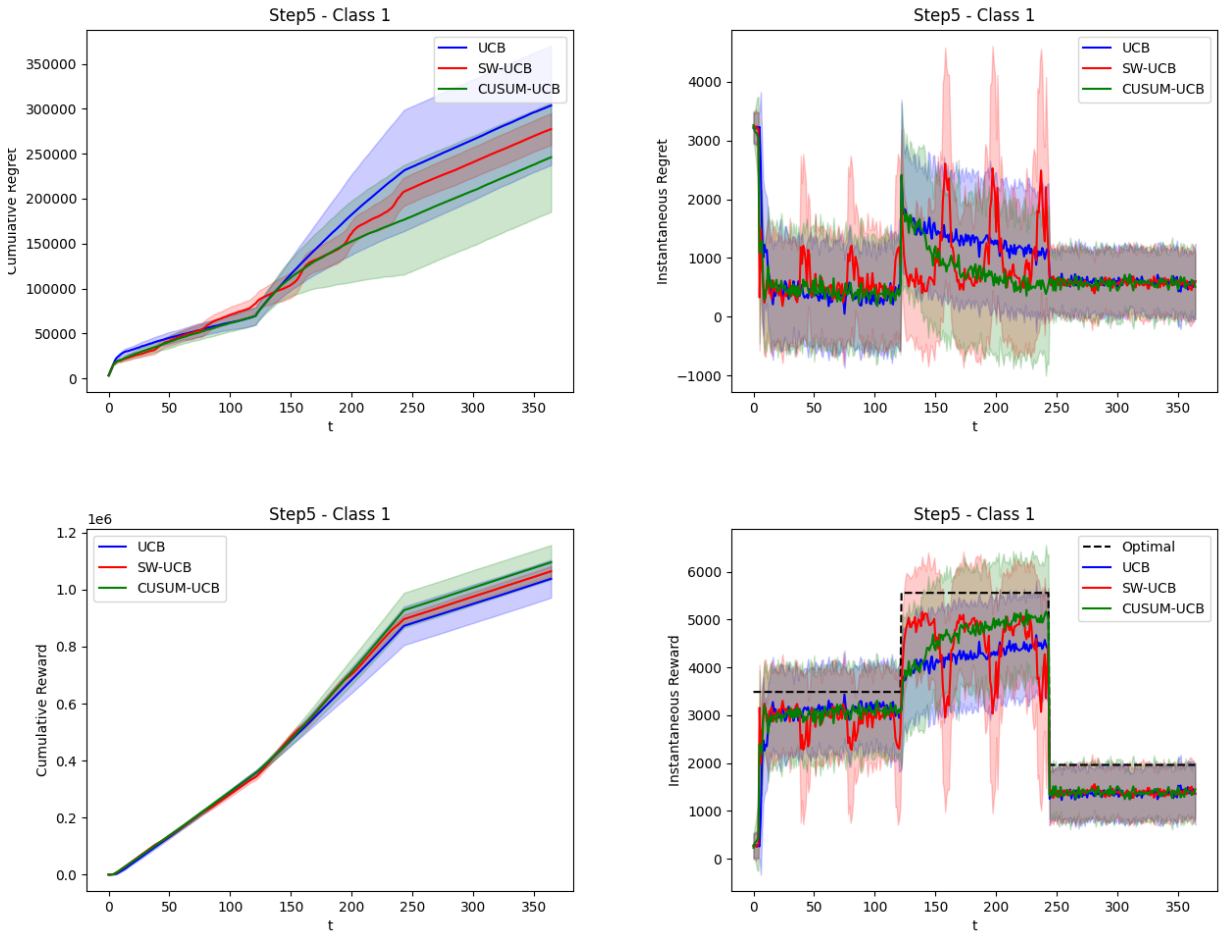
7.3 Pricing learning

In this step, the advertising part is known, so we used an optimizer to learn the price giving the highest reward computed using the real values of the number of clicks and cumulative cost given by the bid related to the chosen price.

To model the changes in the customers' behaviour, each phase can be associated with a specific time of the year.
We assumed that our time horizon does not cover the year from the first of January to the last of December, but it represents a generic period of 365 days. For example, we considered that it starts around May with an initial phase characterized by regular purchasing behaviour where the preferred price (for middle-class customers) is the second cheapest.
After that, around September, the holiday period starts, when people are more willing to purchase even if at a higher cost. The consequences are that the chosen price of the second phase is the middle one (higher than the regular one) and every price meets an increase in their conversion rate.
Finally, the third phase is the spring one, characterized by discounted sales. During this period, the cheapest price is the preferred one with a huge peak in its conversion rate, while the others suffer a reduction in their conversion rate, the cheaper ones because they are too similar to the regular one (representing little to no saving to the customers' eyes) and the more expensive ones because in contrast with the trend of the whole phase.

To choose the parameters of the two variants of UCB we relied on the theoretical suggestions.
In particular, for the sliding window size, we opted for a value directly proportional to $\sqrt{T}$; for the change detection variant, we used CUSUM as the change detection mechanism using a threshold for the detection which is proportional to $\log T$ and an exploration parameter proportional to $\sqrt{\frac{\log T}{T}}$.
It is important to highlight that we did not detach from the theoretical suggestions and we did not perform any kind of tuning because both algorithms are deployed over a single case of functions; in other words, there is no variety or randomicity in the functions modelling the behaviours of customers (apart for the noise added when receiving the reward). This limitation in the variety of cases analyzed could have been the cause of serious overfitting if we decided to tune the parameters of the algorithms with respect to the results achieved in our experiments.

In terms of results, we expected all three versions to achieve sub-linear regret, with the vanilla variant of UCB being the worst one.
In detail, we expected the regret of UCB to increase during the second and third phases since the algorithm has no mechanism to face the changes in the environment other than the slow increase of the not-pulled arms' bounds because of the term related to the time step.
For the sliding-window case, since there exists the window mechanism to deal with the changes we expected to get a lower regret; while for the CUSUM case, we supposed that it could reach the lowest level of regret because as soon as a change is detected the algorithm explores all the arms searching for the best one.



As expected, the algorithm achieving the best regret is CUSUM, with sliding-window being the second best and standard UCB the worst one.
It is interesting to notice that the sliding-window version is affected by some spikes with decreased reward and increased regret. It is safe to assume that these peaks are caused by the worst arms which are pulled just once for each window; indeed when the time step at which their samples were obtained exits the window of validity those arms have no valid samples anymore and, thus, they get infinite confidence bound and, as a consequence, the algorithm is forced to suggest those arms for the next pulls. Once they are pulled and a

poor reward is observed SW-UCB can suggest again better arms.

7.4 Sensitivity analysis

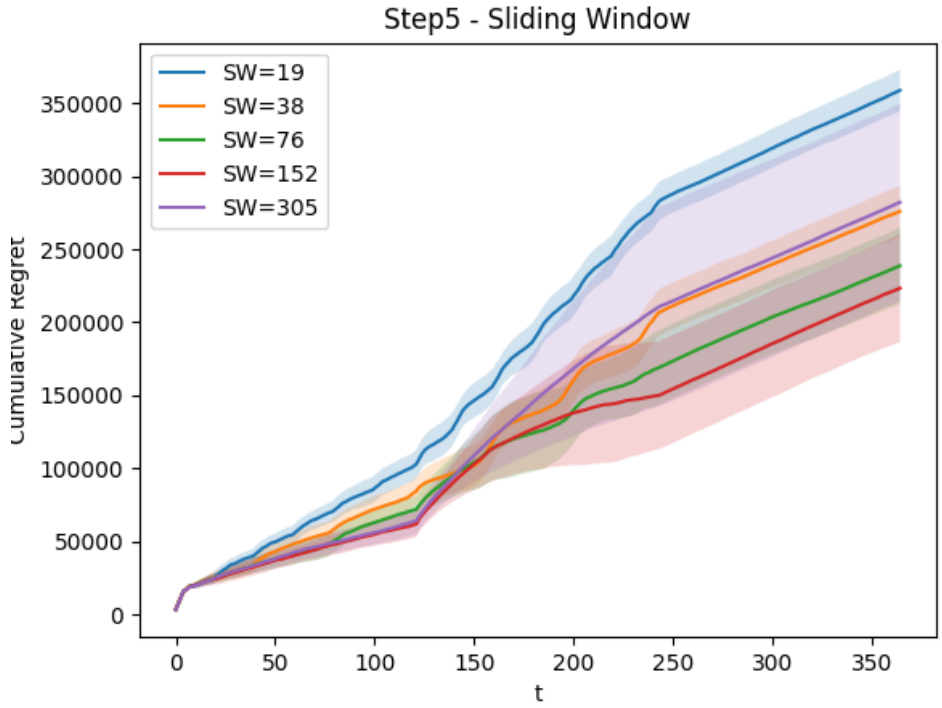
Both variants of UCB are regulated by some parameters: the window size is the only one for the sliding window algorithm, while CUSUM has four different parameters: M, h, ϵ ("eps" in the legends) and α ("alpha" in the legends).
We tested some values for each one to analyze how the algorithms change their results with respect to the theoretical expectations.

7.4.1 Sliding window: analysis of window size

For sliding window UCB, we tested 5 different values of window size, all of them computed proportionally to $\sqrt{T}$, but with different multiplying factors (respectively: 1, 2, 4, 8, 16).

With a small window, the number of samples used to learn is limited, thus more exploration is performed (because it is more probable for an arm to have no valid samples). The consequence of a small window size is that for the algorithm to be effective the changes in the environment behaviour must be frequent and more abrupt.
On the other hand, a big window allows using more samples to learn, but this is counterproductive in the case in which the changes are too frequent because too much history of an arm is taken into consideration, even samples coming from a previous, different, behaviour.

As shown in the following graph, too small window sizes (19 and 38) got high regret, while bigger sizes (like 76 or 152) reduced it.
As expected, if the size got too big (the case of 305), the regret grew again.

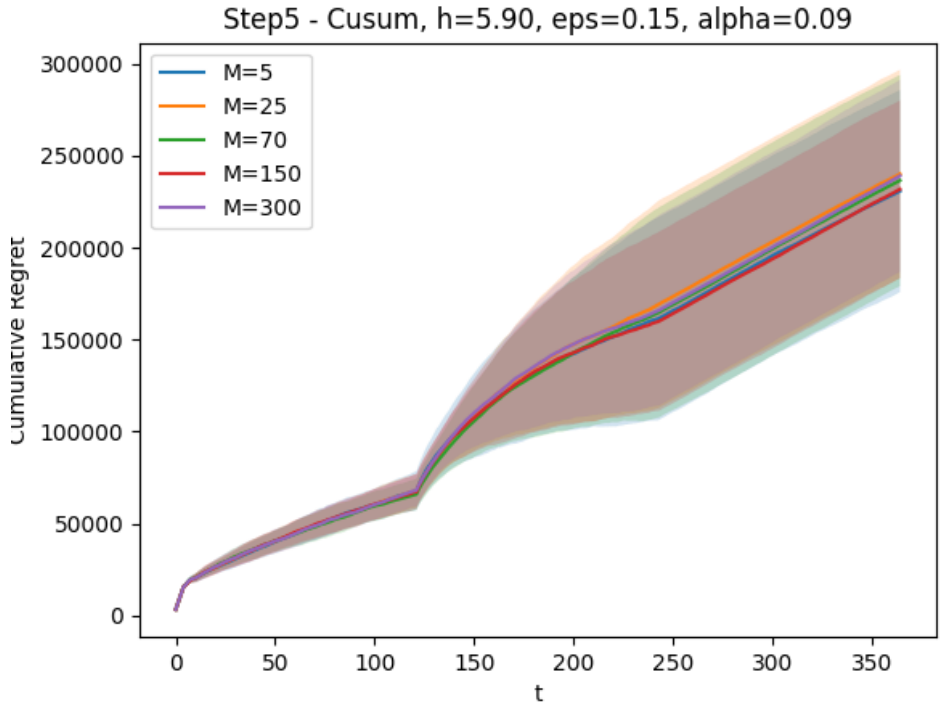


7.4.2 CUSUM: analysis of M

The parameter M of CUSUM represents the number of samples to be used to compute the empirical mean that will be used as a reference for the current behaviour in the actual change detection phase.
We tested 5 different values for M, all of them of the same order of magnitude of the time horizon.

The higher the value of M, the higher the number of samples used to compute the mean. This means that the mean obtained is more indicative of the real behaviour.
The downside of bigger values of M is that the detection test is not performed until all the M samples are collected, thus if the value is too high it could be that a change in the behaviour is detected too late or even not detected at all.

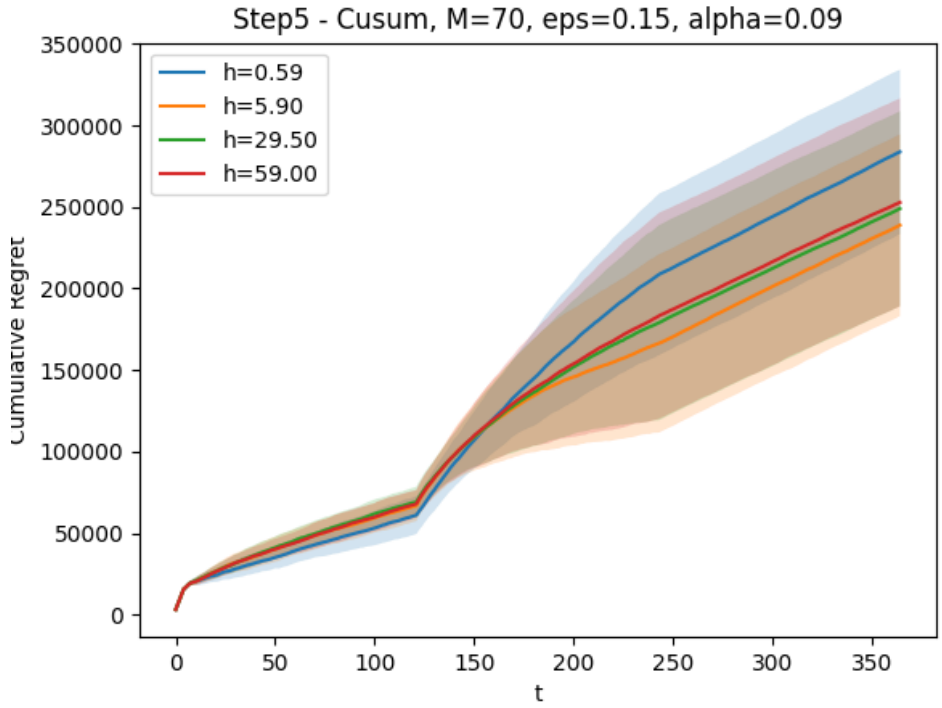
As plotted in the next chart, the performance of the algorithm in terms of cumulative regret is quite similar even for very different values of M; indeed if compared with the gaps in performance obtained in the other sensitivity runs, the difference between $M = 5$ and $M = 300$ is of no significance.
Our hypotheses for this result are that other parameters have a much more important impact, making the difference variating M almost negligible, and also that it is relevant to remember that this is a variation of UCB, so there is still a learning algorithm behind the change detection add-on. Hence it makes sense that the performances do not change too much after the variation of a less influential parameter.



7.4.3 CUSUM: analysis of h

Another parameter of CUSUM is the one used as the threshold to detect any change in the customers' behaviour: h. This means that bigger values of it require the changes to be more evident (i.e. more abrupt), while smaller ones represent a lower threshold to be passed to raise a change detection.

We tried 4 different values of h, each one proportional to $\log T$, respectively with multiplying factors of 0.1, 1, 5 and 10.
What we observed is shown in the chart below and shows that, in our case, both too-small and too-big values achieved high regret compared with a middle-way one, but also that the lowest one suffered much more regret than the higher ones (always with respect to the median one).

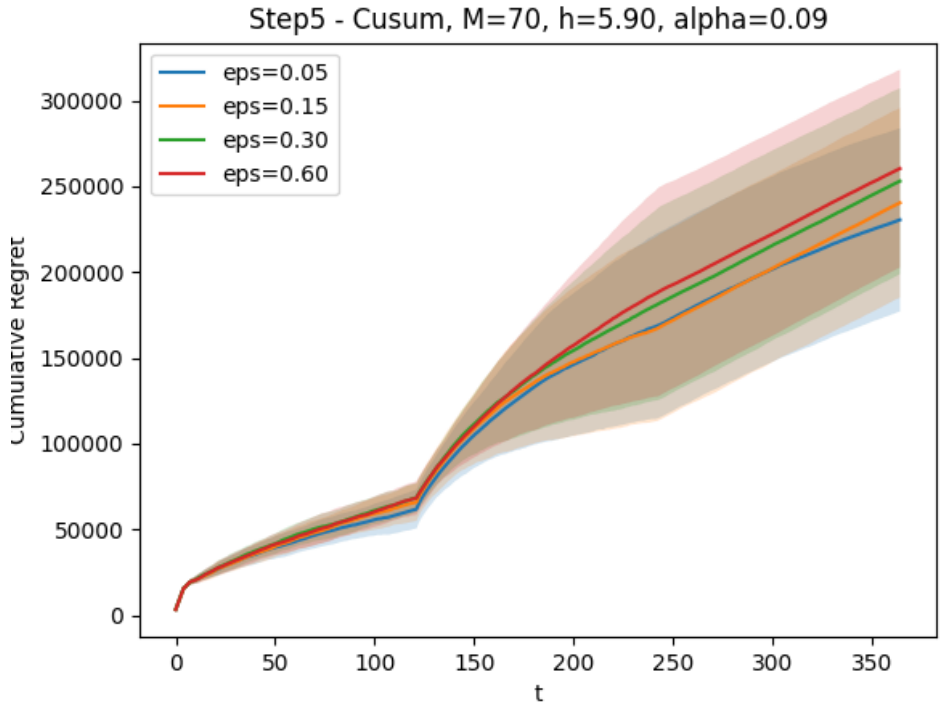


7.4.4 CUSUM: analysis of ϵ

The third parameter which regulates CUSUM-UCB is ϵ ("eps" in the charts). It represents the critical level used to adjust the sensitivity of the change; in other words, is the value used to adjust the difference between the empirical mean and a new sample before it is added to the bound used for the change detection test.

Bigger values of ϵ mean lower sensitivity to the changes, that is more samples or samples with a bigger difference with respect to the empirical mean are needed to put the bounds over the threshold h.
The values we tried (0.05, 0.15, 0.3 and 0.6) were all proportional to what we expected to be the range of difference between a new sample (binary, 0 or 1) and the empirical mean (oscillating between 0 and 1).

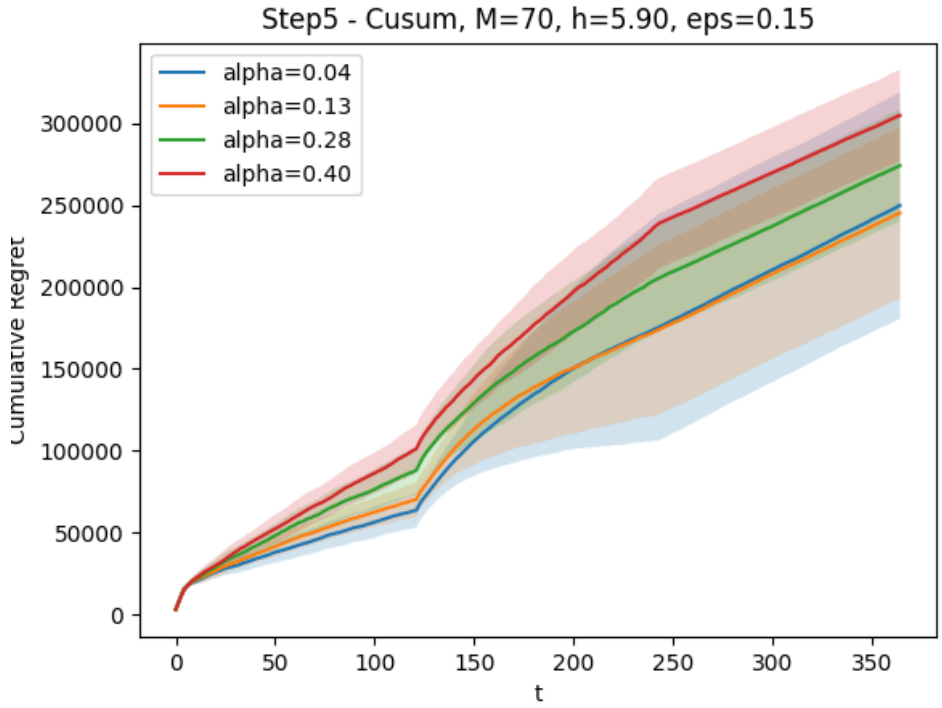
As exhibited by the following graph, in our case, all 4 values scored similar results in terms of regret, with the lower values of ϵ achieving better performances.



7.4.5 CUSUM: analysis of α

α is the parameter of CUSUM directly responsible for the amount of exploration of the algorithm, indeed it represents the probability of pulling a random arm instead of the one suggested by the UCB check over the confidence bounds.
Since it represents the exploration tendency of CUSUM, higher values allow us to detect changes happening at higher frequency but also increase the risk of pulling high-regret arms in place of the optimal one.

We performed our tests over 4 different values for α, all proportional to $\sqrt{\frac{\log T}{T}}$ and with various multiplying factors (specifically: $\sqrt{0.1}$, 1, $\sqrt{5}$, $\sqrt{10}$).
We obtained that under a certain value, small α (0.04 and 0.13 for us) give all almost the same regret, while if they are increased (0.28 and 0.4), CUSUM accumulates more regret, as displayed in the subsequent chart. This matches what we expected from theory because the higher values of α force more frequently the pulling of random (and thus potentially sub-optimal) arms.



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 modelled 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

A scenario with many abrupt changes poses harder obstacles in the learning process. In the setting proposed the bid is fixed and therefore we select a value of the bid which is the optimal value of 2.79€ computed in the previous steps. This choice allows us to have

comparisons with previous results. The parameters we need to estimate to maximize the monetary reward are the conversion rates which are unknown and rapidly changing.

Such a rapidly changing scenario is called adversarial bandit. It differs from the classical bandit problem for the changes in the quantities that for a classic bandit are fixed and learned over time. In fact, in an adversarial bandit setting, we cannot expect to learn quantities changing over time. We can only try to play a good arm without the hope to learn anything useful from its reward. The algorithms proposed to deal with these settings still contain a learning component because in practice there are no abrupt changes over each round and therefore they still need to make an educated guess.

8.2.1 EXP3

The EXP3 algorithm proposed is an algorithm designed to play in an adversarial bandit setting. Without entering into much details we describe its working.

At each round the arm to be played is selected by a random draw from a probability distribution over each arm computed at the previous step. The probability distribution of arm i at round t is computed by this formula

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K} \tag{15}$$

where $K$is the number of arms and $w_i(t)$ is the weight associated to arm i at round t and its value is computed as

$$w_{i_t}(t+1) = w_{i_t}(t) \cdot e^{\frac{\gamma}{K}\hat{x}_{i_t}} \tag{16}$$

where i_t is the arm pulled at time t and $\hat{x}_{i_t}$ the expected reward obtained from that arm at time t: $\hat{x}_{i_t} = \frac{x_{i_t}}{p_{i_t}}$. The remaining weights are not updated.

The initialization of the weights is 1 for each arm and the rewards have to be rescaled to the interval $[0, 1]$. The parameter $\gamma \in (0, 1)$ is a hyperparameter to be tuned. Closer to 1 the probability distribution over the arms tends to be uniform, far from 1 and closer to 0 it gives more probability according to the weights and therefore to the rewards obtained by the game.

EXP3 being an adversarial bandit algorithm presents, as said before, learning inclinations. In fact, when an arm is selected its weight increases if the reward is high and as a consequence, the probability of drawing it is higher. Compared to classical bandits, such as TS or UCB, EXP3 explores more between all the arms. This behaviour is needed to compensate for the lack of fixed rewards in time.

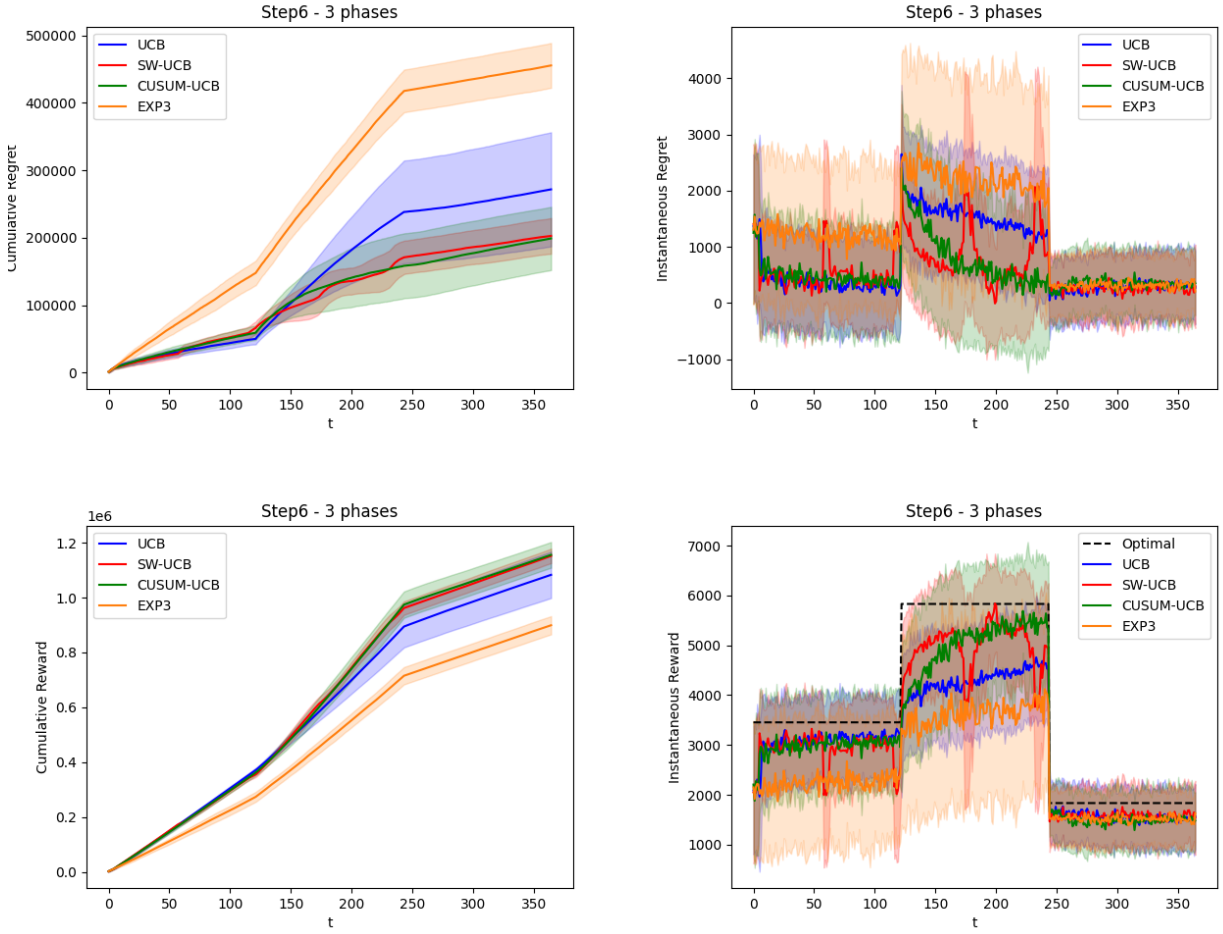


Figure 12: Step6: results of 100 experiments in a 3-phase changing environment

We analyse the performance of EXP3 in the Step5 setting, where there are 2 abrupt changes so 3 phases each one lasting around 120 rounds. As we expect the EXP3 algorithm does not perform well in a setting where there are only 2 changes over 365 rounds. In fact, the regret we observe is linear in all three phases. This is in line with the fact that EXP3 even in a single phase scenario has chances to draw each arm even if a high weight is assigned to the best arm. By looking at the instantaneous reward and the instantaneous regret it can be noted that the learning velocity of EXP3 is very slow compared to UCB,SW-UCB and CUMSUM-UCB. By looking at the probability distribution assigned to the arms over the rounds we observed it does not evolve much from the uniform distribution assigned at the beginning, even for small values of γ. With this knowledge it can be explained a slow learning rate for EXP3.

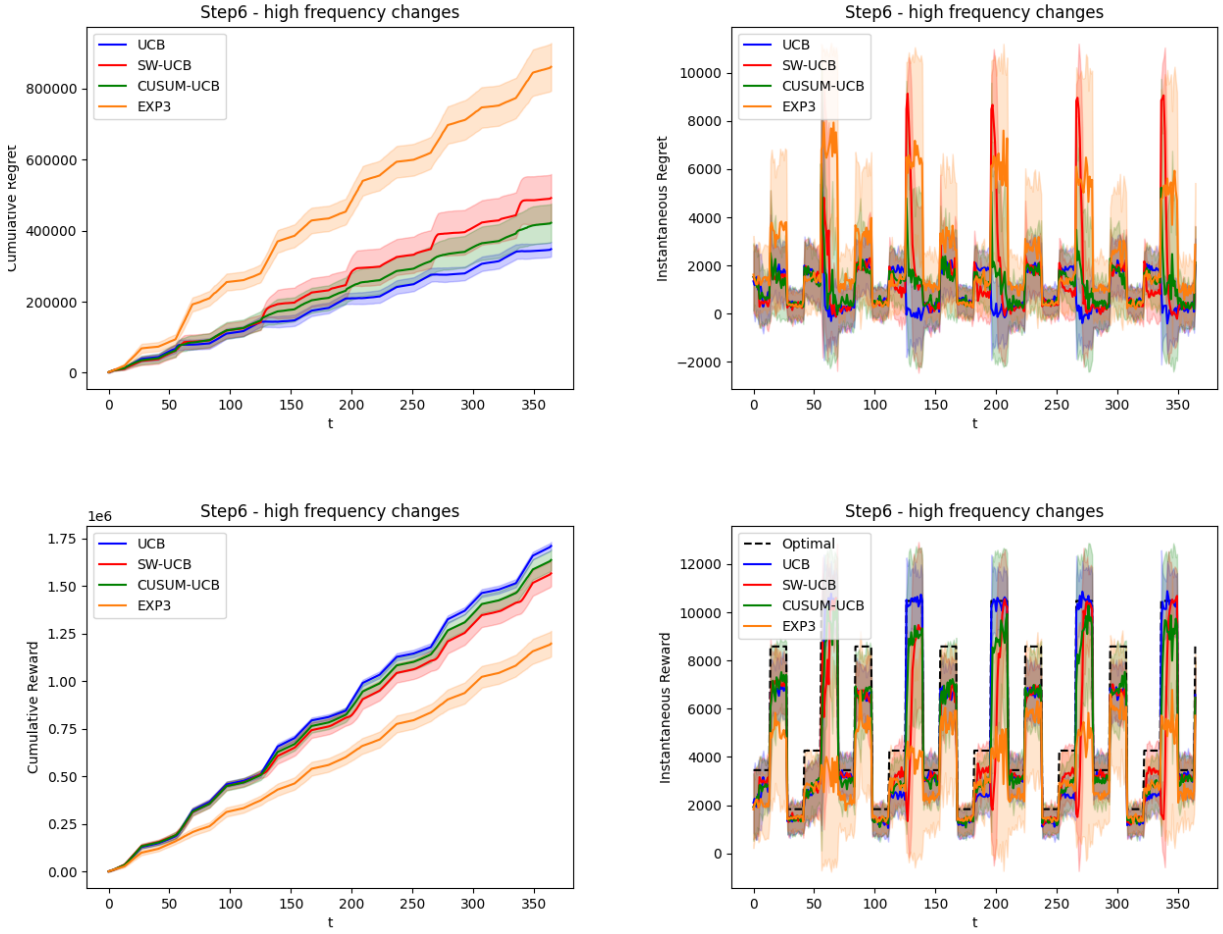


Figure 13: Step6: results of 50 experiments in a high frequency changing environment

We challenge EXP3 in a setting with high-frequency changes. We created 5 different conversion rates, 3 of which are the same as the previous scenario. Each phase has a different optimal arm. Each phase last 15 round each. The results we found are that Exp3 does not perform better than the other methods. In fact, the EXP3 algorithm has theoretical good performance in expectation when the reward is between 0 and 1 with extreme values achievable in practice. This is not our case being the learning reward is the conversion rates times the selling price. To be able to use EXP3 algorithm we need to normalize the reward. To limit the reward between 0 and 1 the rescale factor takes into account a maximal theoretical reward of 1 for the conversion rate multiplied by the maximum for all prices, which means rescaling by $1 \cdot 250$. This results in a scaling between 0 and 1 where in practice the monetary rewards per click are around 50€ so the rescaled reward is of the order of below 0.2. This could be a reason for the weak performance of EXP3. We note also that the other algorithms have comparable performances in such scenario. None of the algorithms reaches a sublinear regret but linear as we would expect in such instances of high-frequency changes. By having a linear regret we can verify that in such scenario the cumulative reward is of the order of magnitude of the cumulative regret.

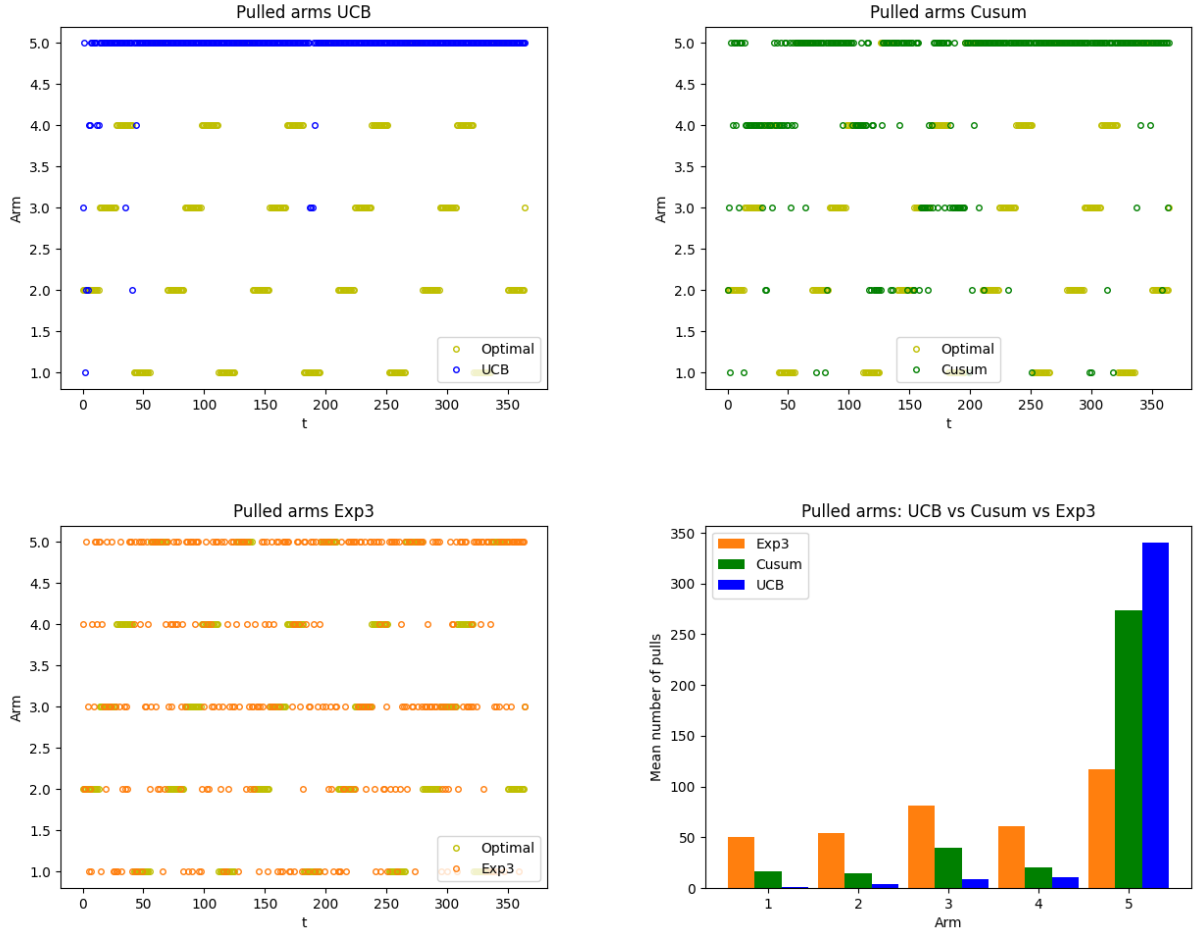


Figure 14: Step6: pulled arms in a high frequency changing environment

To understand what happens in a high-frequency changing environment it is interesting to track the history of arms pulled by UCB, EXP3 and CUMSUM. The UCB algorithm clearly plays almost exclusively the fifth arm even though the optimal arm is different for each phase. UCB selects this arm because even if it has a small conversion rate in some of the phases to it there is associated a high monetary payment. This results in the total being the best arm to pick if we fixed the choice of the arm forever. The CUMSUM algorithm has slightly more variation towards other arms than UCB. It still chooses the fifth arm as the best policy for the same reason of UCB. The change detection mechanisms allow it to explore different arms at the early stage of the run while UCB has almost no exploring stage. In the second part of the run, CUMSUM behaves like UCB not detecting changes to each phase. EXP3 behaves very differently with respect to the other algorithms. We can see there is a more uniform distribution between all the arms. The highest probability is still assigned to the fifth arm as the other algorithm suggests us to pick. Our final take on EXP3 is that it is an algorithm that needs to be tuned correctly. It is not a good algorithm to choose if the reward we expect is not possible to bound from above and below a priori. Even if a good bound is found the reward has to spread between all the interval $[0, 1]$ otherwise we have a very slow learning speed.