Online learning application

Professors: Nicola Gatti, Matteo Castiglioni, Martino Bernasconi de Luca

Pricing-Advertising project

Francesca Arrigoni 10597178,
Enrico Brunetti 10864120,
Stefano Ferrara 10615641,
Davide Gesualdi 10885255,
Stefano Vighini 10622788

July 2023

Repository link: https://github.com/dav-G/OLA-Project

POLITECNICO
MILANO 1863

Contents

1 Step 0

Our project wants to emulate the pricing and advertising strategies of a company that sells computer backpacks through a website for the duration of a year.
The price of the backpack that we want to analyze can vary between 10€ and 50€ and we consider 5 possible prices: {10,20,30,40,50}.
The company will consider 100 possible bids between 0€ and 2€ for the advertising effort.
We assume that the customers seeing the ad and visiting the website can be divided into 3 classes by using the binary features:

- employment: whether the customer is a student or a worker
- residence: whether the customer is a commuter or not

The classes will then be:

- C1: all students, whether they are residents or commuters they have a high need for a backpack but have low funds
- C2: commuter workers, they have a medium need for a backpack of this kind depending on the work they do but they have the funds necessary to easily acquire it
- C3: resident workers, they have a low need for the good but high funds

For each class, we defined some functions that characterize them.

1.1 Number of clicks

This function represents the average dependence between the number of clicks and the bid for each class.
It is defined in the Customer class and given a bid returns the number of new users that click on the ad in a specific day.

$$\begin{cases} 100(1 - exp\{-1.5 \cdot bid - 0.5 \cdot bid^2\}) & \text{for C1} \\ 80(1 - exp\{-0.5 \cdot bid - 1.5 \cdot bid^2\}) & \text{for C2} \\ 65(1 - exp\{5 \cdot bid + 0.3 \cdot bid^2\}) & \text{for C3} \end{cases}$$

We can see that to a higher bid correspond a higher number of new customers, in particular we can see that a higher bid has a bigger effect on C1 as it has a higher need than other classes but less money.
For C3 on the other hand the ad is effective for those that can be swayed but at a certain point it does not have any more effect as the customers belonging to C3 do not really need the backpack even if they see the ad.

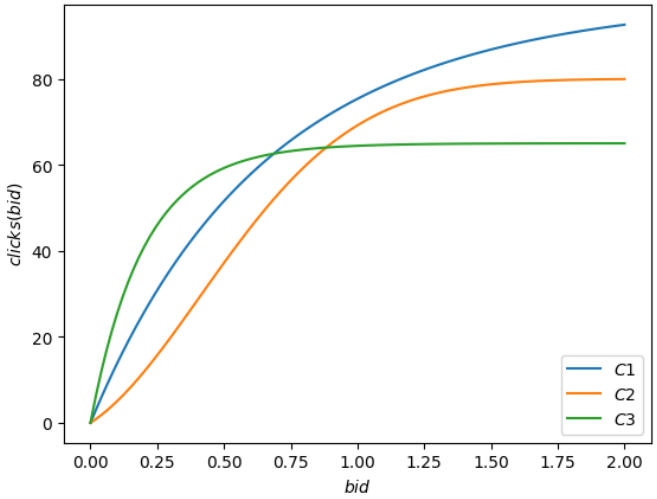


Figure 1: Daily number of clicks for each class

1.2 Daily click cost

This function represents the average daily click cost given a bid for each class.
It returns the cost that the company has to pay for each click on the ad, as we do not have the bids of the others company that compete for the spot this is already the result of the second price auction.

$$\begin{cases} 1.5 \cdot 2 \cdot \log(1 + bid/2) & \text{for C1} \\ 1.5 \cdot 1.6 \cdot \log(1 + bid/1.6) & \text{for C2} \\ 1.5 \cdot 1.3 \cdot \log(1 + bid/1.3) & \text{for C3} \end{cases}$$

An higher bid corresponds to a higher payment but we can see that the payment is always lower then the bid (as the theory suggests).

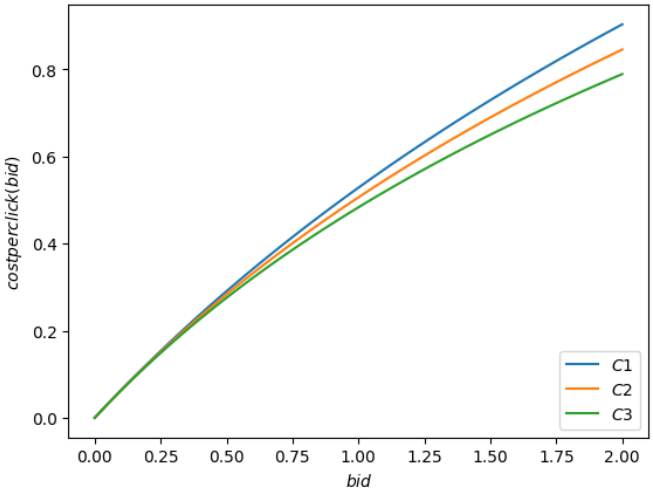


Figure 2: Daily cost per click for each class

1.3 Conversion rate

This function represents the conversion rate for each class.
Given a price for the backpack it returns the probability that a customer visiting the website will buy the backpack.

From the graph we can see that at lower prices C1 (students) will buy more backpacks but as the price increases the probability of conversion lowers significantly as they are the group with lower economics means.
C2 and C3 the workers classes have a more stable curve that decreases when the price increase (as the backpack is not a luxury item).
In particular customers belonging to C3 have a much lower conversion rate as they have a lower need for the item in question.

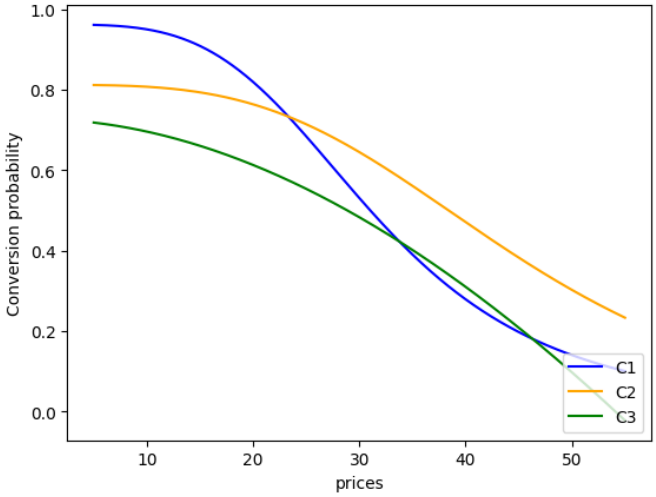


Figure 3: Conversion rate for each class

1.4 Clairvoyant algorithm

For each class, we ran the clairvoyant algorithm to compute the best choice for the price and the bid.
First, we found the best price for each class as the one maximizing the gain :

$$p = \text{price chosen}$$
$$c = 8 = \text{cost of production of one backpack}$$
$$conv = \text{conversion rate function as described in 2.3}$$
$$gain(p) = (p - c) \cdot conv(p)$$

The final reward function for each class considering also the advertising problem is:

$$bp = \text{best price found maximizing the gain}$$
$$c_{ad} = \text{cost per click of the ad as described in 2.2}$$
$$n_{ad} = \text{number of clicks as described in 2.1}$$
$$reward(bid) = n_{ad}(bid) \cdot (gain(bp) - c_{ad}(bid))$$

The total reward will be given by the weighted sum of the rewards of the single classes.
The results the clairvoyant algorithm found are as such:

Class	Best price	Best bid	Total reward
C1	30€	2.0€	1008.17€
C2	40€	1.78€	1136.25€
C3	30€	0.99€	551.78€

Table 1: Results of the clairvoyant algorithm

2 Step 1

The first assignment was to optimize the pricing part of the problem when the bid is fixed using an online learning algorithm.
We assume that all clients that come to the site belong to class C1 and that the curves related to the advertising part of the problem are known. This means that the bid that we utilize is the one that maximized the reward, found through the clairvoyant algorithm.
To solve this online problem we employed two different Multi Armed Bandits (MAB) in order to minimize the regret,Thompson Sampling and UCB-1 (Upper Confidence Bound) and we compared the results obtained.
Every time that we pull an arm we sample from a Bernoulli distribution to get the number of customers that bought the backpack out of all the ones that entered the site (in our case as the bid is fixed 92 customers enter the site each day).

2.1 Algorithm comparison

2.1.1 UCB-1

Upper Confidence Bound is a deterministic algorithm that associates an upper confidence bound to every arm, at each round the arm with the higher ucb is chosen.
The upper bound is computed for each iteration as:

$$ucb = \bar{x} + \sqrt{\frac{2 \cdot \log(t)}{n_{a_t}(t-1)}}$$

Where t is the time index and $n_{a_t}(t-1)$ is the number of times the arm a_t has been pulled.

2.1.2 Thompson Sampling

Thompson Sampling is a stochastic algorithm that sets a prior on the expected value for every arm, and selects the arm with the best sample, according to the updated parameters.
These are the parameters of a Beta function and take account of the number of successes (α_{a_t}) and failures (β_{a_t}) of the arm a at time t.
So for each round:

$$(\alpha_{a_{t+1}}, \beta_{a_{t+1}}) \longleftarrow (\alpha_{a_t}, \beta_{a_t}) + (s_t, f_t)$$

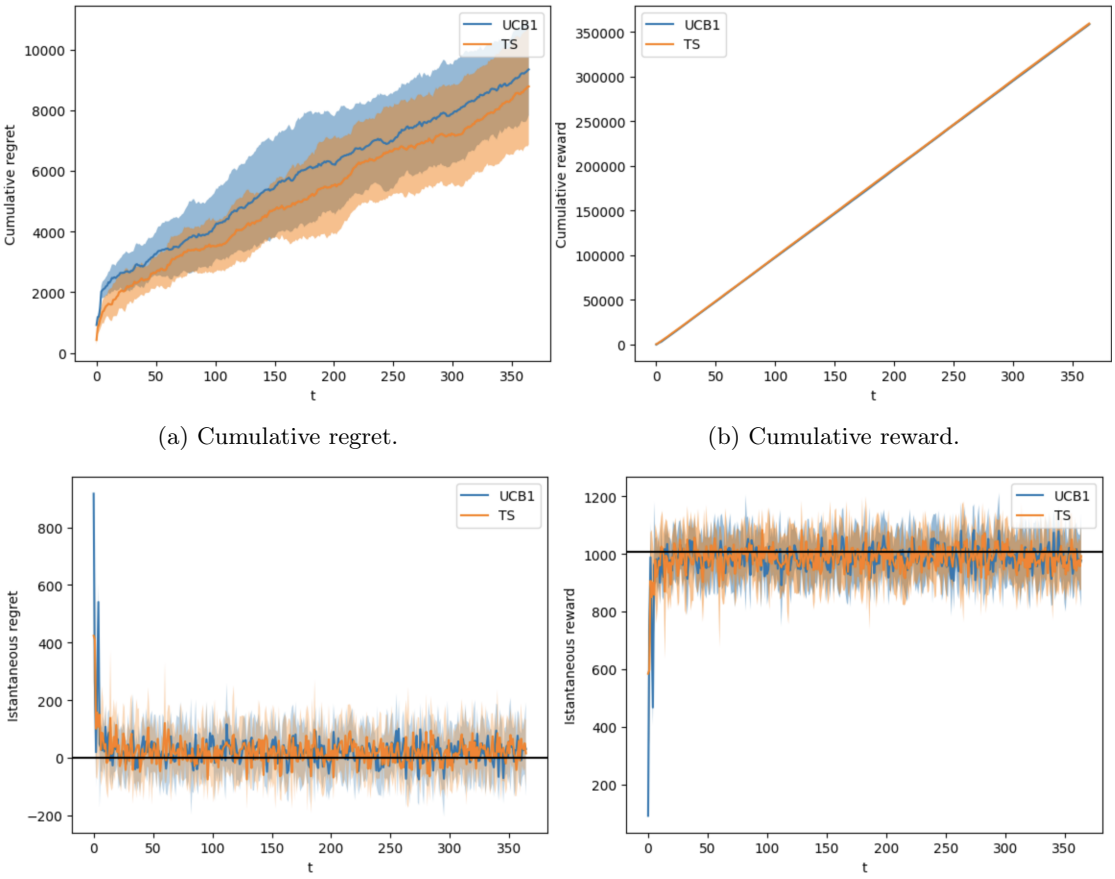
Where s_t are the number of successes of round t and f_t are the number of failures of round t.

2.1.3 Results

In order to compare the two algorithms we compare some metrics:

- Regret: The difference between the expected reward of the optimal policy and the reward of the algorithm.
  A lower regret means that the algorithm is performing better.
- Cumulative regret: The sum of the regrets that the algorithm has received over time.
  A lower cumulative regret means that the algorithm is performing better.
- Reward: The rewards that the algorithm has received over time.
  A higher reward means that the algorithm is performing better.
- Cumulative reward: The sum of the rewards that the algorithm has received over time.
  A higher cumulative reward means that the algorithm is performing better.
- Convergence time: The time it takes for the algorithm to converge to the optimal policy.
  A faster convergence time means that the algorithm is learning more quickly.
- Computational complexity: The amount of computing resources required to run the algorithm.
  A lower computational complexity means that the algorithm is easier to run.

We expect both algorithms to achieve a sub-linear regret and reach convergence to 0 for the regret and to the reward of the clairvoyant algorithm for the reward.
From the graphs, we can easily see that both algorithms converge to the optimal solution, the convergence time is lower for the TS algorithm but it is not so dissimilar than the UCB-1 one.
In the beginning, the rewards of the Thompson sampling algorithm are higher but in less than 10 days the UCB-1 rewards reach the same values.
From the cumulative regret graph we can see that the Thompson Sampling algorithm has a lower mean cumulative regret but the curves are very near each other and they often are in the range of the standard deviation of the other.
Given that from theory we know that the UCB-1 algorithm is less computationally complex in this part of the problem we cannot say that one algorithm is significantly better than the other.



(a) Cumulative regret.

(b) Cumulative reward.

(c) Instantaneous regret.

(d) Instantaneous reward.

Figure 4: Results of step 1.

3 Step 2

In the second assignment, we want to optimize the advertising part of the problem when the curves related to the pricing problem are known.
Like in step 1, all customers belong to C1.
We employ Gaussian Processes in our algorithms as the number of possible bids is very high.
A Gaussian process is a collection of random variables such that any finite number of them can be jointly modeled as a multivariate normal distribution. This allows us to greatly reduce the computational cost of these algorithms as we assume that there exist a correlation between the reward of two neighbours arms.
In order to use the Gaussian processes we have to assume that the number of clicks function and the daily click cost function are smooth as to obtain a smooth reward function.
By utilizing the smooth functions described in step 0, we can apply a Gaussian Process without any issues.
The objective of the Gaussian Processes is to maximize the final reward which is calculated inside the learner.
As the daily click cost and the number of clicks have very different orders of magnitude (as we can see in the respective graphs) we employed a different σ for each of these functions when sampling from the real function.
In particular, we used $\sigma = 10$ when sampling from the number of clicks function and $\sigma = 0.2$ when sampling from the daily click cost function.

3.1 Algorithm comparison

3.1.1 GPUCB

Gaussian Process Upper Confidence Bound (GPUCB) is a Bayesian optimization algorithm that uses Gaussian processes to model the uncertainty about the best arm to pull. It works by constructing an upper confidence bound on the expected reward of each arm and then choosing the arm with the highest upper confidence bound.
The confidence bound is constructed as:

$$ucb = \bar{x} + \sigma \cdot \sqrt{\beta}$$

Where the beta function is defined as shown in the article Srinivas et al. [2009] *"Gaussian Process Optimization in the Bandit Setting:No Regret and Experimental Design"* :

$$\beta = 2\log(\frac{N \cdot t^2 \cdot \pi^2}{6 \cdot \delta})$$

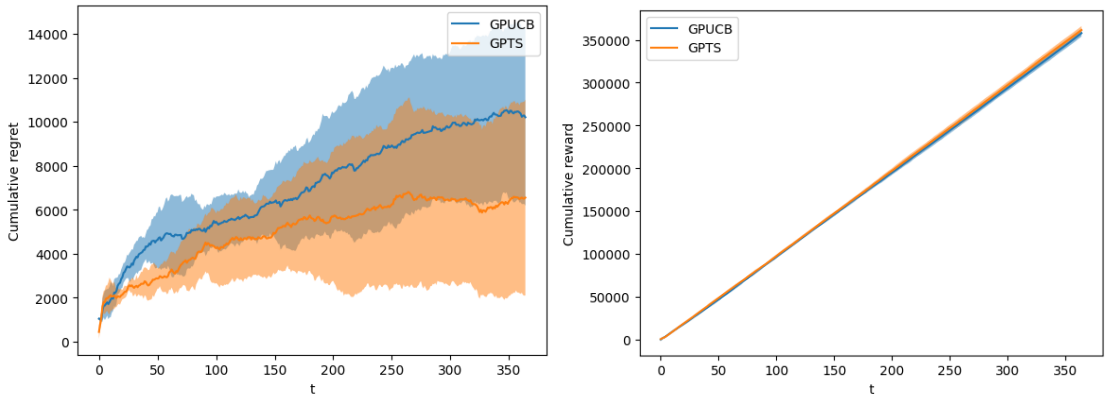
Where $\delta \in [0, 1]$ and $1 - \delta$ is the confidence level that we want, in our case δ was set at 0.05, N is the number of arms (100) and t is the time instant.

3.1.2 GPTS

Gaussian Process Thompson Sampling (GPTS) is a Bayesian optimization algorithm that uses Gaussian processes to model the uncertainty about the best arm to pull. It works by sampling from the posterior distribution of the Gaussian process and then choosing the arm with the highest expected reward.
Instead of a Beta distribution like in a normal Thompson Sampling algorithm the distribution of the value for each arm around is a Gaussian.

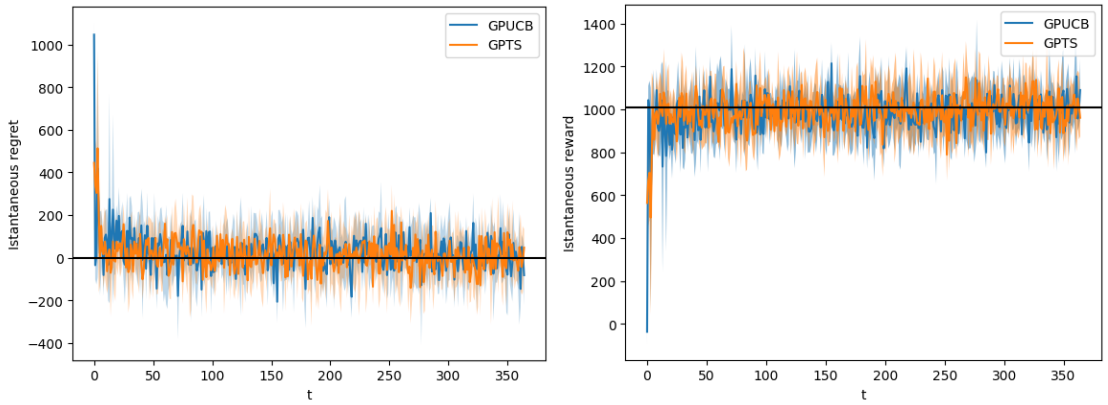
3.1.3 Results

As we expected both algorithms show a sub-linear regret and they both converge to the optimal solution in a reasonable amount of time.
From the graphs, we can observe that the Thompson Sampling algorithm has lower cumulative regret.
This is because in the first 50 days we can see that the GPUCB algorithm has a higher regret than the Thompson sampling algorithm. This is a result of the random sampling that the algorithm employs.
We still cannot say that one algorithm performs better that the other because the mean cumulative regrets are often distant from the other curve less than the standard deviation of it.



(a) Cumulative regret.

(b) Cumulative reward.

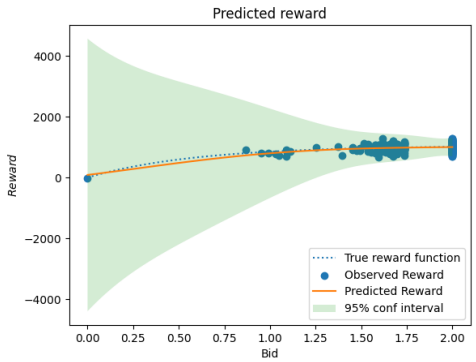


(c) Instantaneous regret.

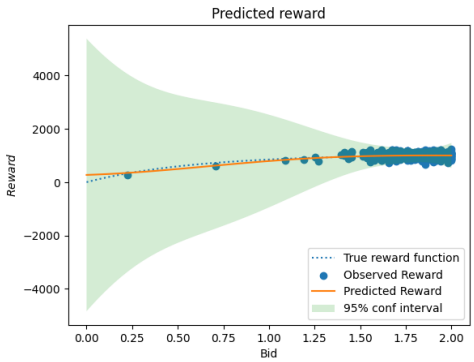
(d) Instantaneous reward.

Figure 5: Results of step 2.

In the following graphs we can see which arms were pulled in each algorithm and the predicted reward function that the algorithms find.
As we can see both algorithm are able to obtain a good approximation of the reward function. From the observed rewards it is evident that the GPTS employed a more explorative approach as the theory suggest.



(a) GPUCB algorithm.



(b) GPTS algorithm.

Figure 6: Pulled arms in each algorithm.

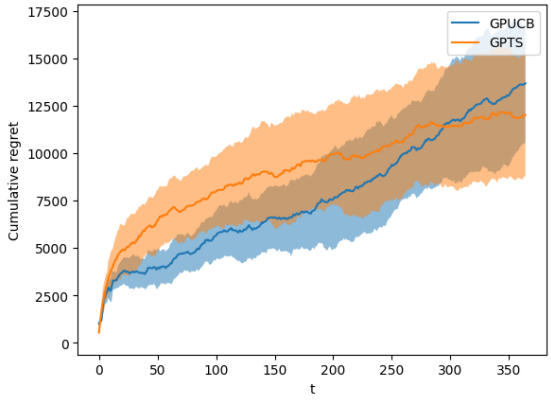
4 Step 3

In the third step, both the curves of pricing and advertising are unknown and all customers belong to class C1.
This is a mixture of the previous two steps, we employ a common environment that first samples from the advertising curves and uses the results obtained (in particular the number of clicks) to sample from the pricing curve.
For the advertising part of the problem, we employ the Gaussian processes used in step 2 while for the pricing part, we use the algorithms from step 1 with the respected update conditions.
For each round, we pull respectively from the pricing and advertising learner the arm that maximizes the reward.
The selected arms are then used to sample from the real distributions, we calculate the reward for the round and we update the parameters as described in the previous steps.

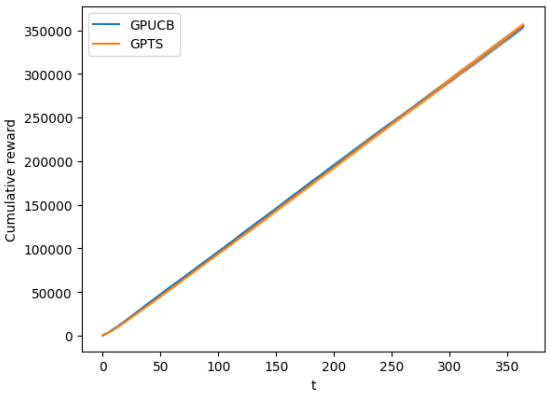
4.1 Algorithm comparison

4.1.1 Results

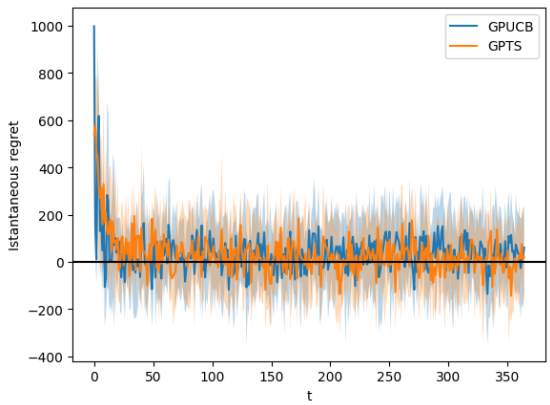
We can see from the graph that the Thompson sampling algorithm combined to the GPTS algorithm performs slightly worse than the UCB ones.
This can be seen both in the cumulative regret graph as it has a higher cumulative regret for the majority of the year but also from the instantaneous regret graph where we can see that the Thompson sampling algorithms takes longer to converge to the optimal solution.
Having said this both algorithms reach convergence and have a sublinear regret in conformity with the theory.



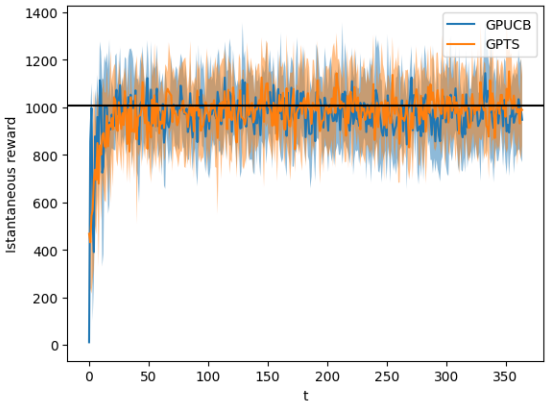
(a) Cumulative regret.



(b) Cumulative reward.



(c) Instantaneous regret.



(d) Instantaneous reward.

Figure 7: Results of step 3.

5 Step 4

In this section, we will explain how we dealt with disaggregated demand curves.
Every experiment will consider 3 different types of customers that can be extracted with the following probabilities:

Customer	Probability
C1	0.33
C2	0.33
C3	0.33

Table 2: Probability that an ad gets shown to a class of customer

5.1 Definition of context

A context is a list of set of features. Sets are implemented as dictionaries. As in our case we just have binary features, the true context is represented by the following object:

```
[ {'student': True}, {'student': False, 'commuter': True},
{'student': False, 'commuter': False} ]
```

A context represents a division of features' values so that each set of features can identify a different class of customer, therefore a different set of pricing and advertising curves.
The intersection of every pair of sets is the null set, and the union of all sets is the total set of features.

5.2 Adapting Environment and Learner to the context problem

First of all, we modified our Environment class in order to return the features representing the customer that has been chosen for a given round. Before pulling an arm from the Learner, the latter is fed with the features of the customer that will be pulled. In this way, the learner can fit the curve of the specific class of customer given his features, pull the relative arm and then observe the reward. This behaviour wants to model real life scenario in which, for example, a social network knows beforehand the features of a customer for which it will display an advertising and will optimize the pulled arm (aka the advertising) for that very customer class.
It's important to notice that ContextLearner will use only the subset of samples related to the pulled customer instead of the total set of rewards. This is expected to create a slower convergence, but more precise.

5.3 Scenarios

We may run the algorithm in one of the following scenarios:

- the context is not considered: the algorithm runs on 3 different types of customers using a single aggregate curve.
- the context is known: before the start of the learning, Learners are fed with the known context. In this way it will be able to learn from the beginning the specific curves of every type of customer.
- the context is not known: after a fixed period of time, the algorithm will run a Context Generation algorithm and, if some decision can be made, it will update the context of the Learner, splitting the actually existing samples for every context generated.

5.4 The Context Generation Algorithm

The Context Generation algorithm runs every 14 days. It takes as input the collected rewards, pulled arms, pulled features and a set of features on which the algorithm can split.
For every feature, the algorithm splits the rewards in as many lists as the number of possible values of the features (in our case, just 2 because our features are binary). Then, it computes

the expected reward (for every feature split) of the optimal arm and the lower bound of the probability of customers to be pulled. If the following condition applies:

$$\mu_{a*,1} \cdot P_1 + \mu_{a*,2} \cdot P_2 >= \mu_{a*}$$

then the curves can be disaggregated and features split. Then, the algorithm is run recursively for every remaining feature. In the end, it will return a list of dictionaries where every dictionary is a class of customers.
The lower bound is computed with the formula seen in the lectures:

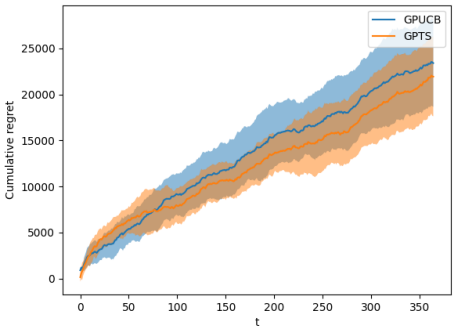
$$x - \sqrt{\frac{log(\delta)}{2 \cdot |Z|}}$$

where |Z| is the cardinality of the set of samples taken into account.
As we have 4 different learners (2 for pricing, 2 for advertising) we run the context generation algorithm for every learner, considering its rewards and its pulled arms.

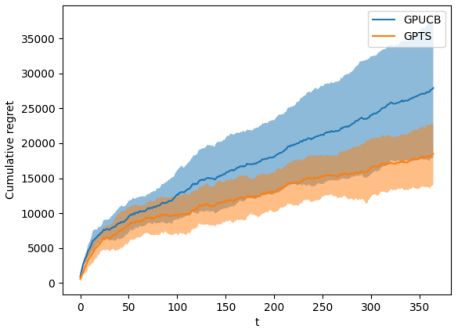
5.5 Results

5.5.1 Graphs

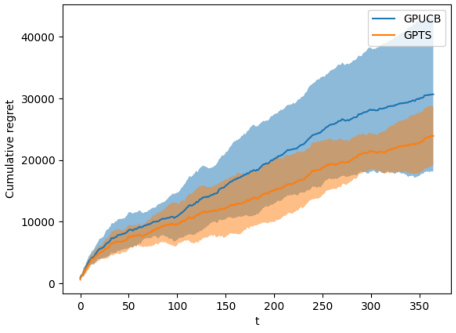
Before discussing the results, a consideration must be done.
The reward we get as an average of the different customers' best rewards is 898.73.
Similiarly, the reward we get using only one bid and one price over the aggregate curve is 869.00.
We can notice that the difference is only 29.73, that is not perceivable from the graphs. The explaination resides in the way the best arms are distributed. In fact, we can notice the best prices have a distance of only one index, while the best bid is noticeably different only for the class with the lowest reward.
This said, we can't really compare the reward difference between using an algorithm that considers the context or that doesn't. What we can do instead is compare the cumulative regret, that is way more interpretable.



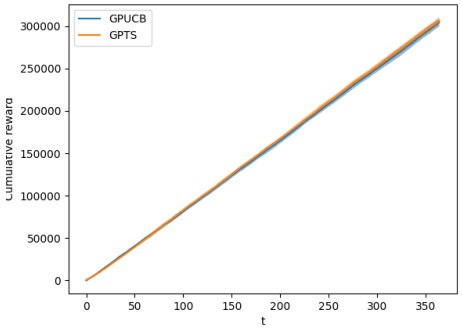
(a) Without context



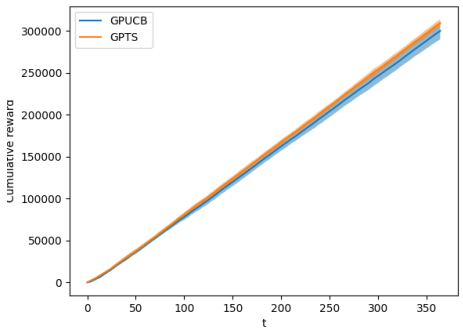
(b) Known context



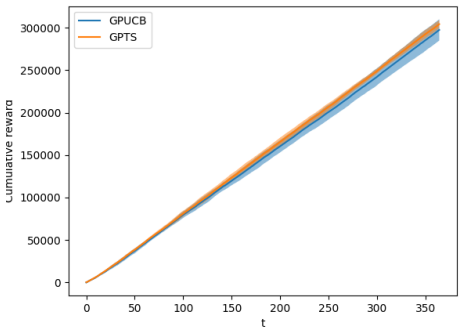
(c) Generated context



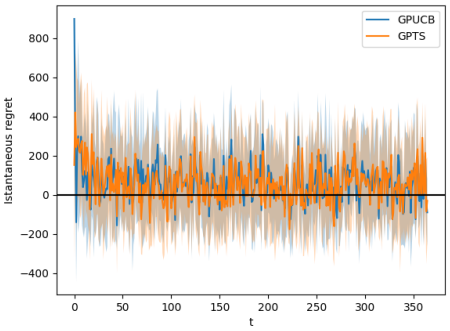
(d) Without context



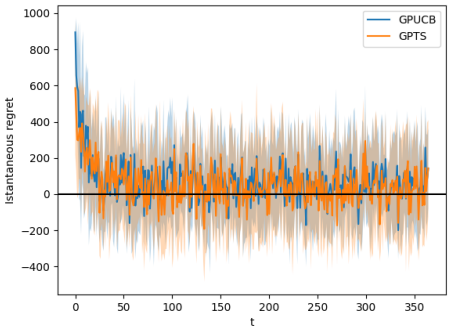
(e) Known context



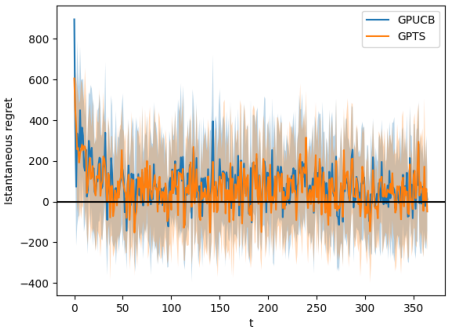
(f) Generated context



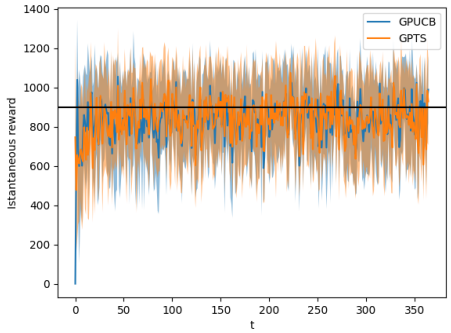
(g) Without context



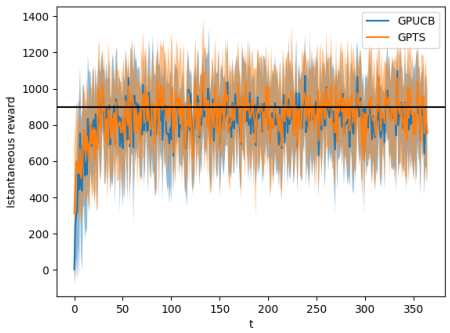
(h) Known context



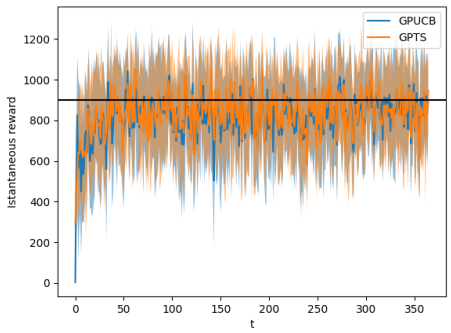
(i) Generated context



(j) Without context



(k) Known context



(l) Generated context

From graphs a, b and c we can observe that the algorithm without context is the one that achieves the worst results, very similar to the context generation algorithm. The one run with the known context experiences a smaller regret and a smaller variance. The one run with context generation instead has more regret in the first stages with respect to the one without context, but his trend shows a more significant reduction later during the year. These results are all

predictable from the theory.
From other graphs we can't really deduce anything, as the difference is so little that it can't be considered significant. We could note, for example, that the reward of the algorithm run without context is slightly less than the other two. This could signal the difference between the reward of the aggregate and disaggregate curve, but again being it less than 30 we cannot conclude anything based on this result.
We can notice that TS in general produces better or at least equal results with respect to UCB. Note that every graph has a significant variance. This is given by how the experiment has been conducted. In fact, at every round only one type of customer is pulled, for simplicity of implementation. So the variance of the results includes also the variance of the expected reward of every customer. What is significant in this experiment is the average value over the experiments.

5.5.2 Generated contexts

Using a context generation algorithm doesn't really improve our reward in this scenario. For the same reason we cited before, that is the low difference between the average reward on the aggregate curve and the average of the rewards of the disaggregated curves, situations similar to this one usually pop up:

Context	**Probability**	**Expected reward**
Context 1	0.64	984.52€
Context 2	0.36	435.91€
Context 1 + 2	1	908.51€

Table 3: Expected values of optimal arms over a split

As we can see, features can't be split. Even using a delta equal to 1 (that is, not using the lower bound of the expected value but only its average value) the split is not done.
Furthermore, the fact that rewards of the same arm are subject to both the noise of the unknown curves and the input of the other arm (transparent for the learner) makes it even more difficult to recognize the context. In conclusion, splits can take place but they are not predictable and they don't add much value to the reward.

6 Step 5

In this section, we are asked to evaluate the performances of three flavors of the UCB1 algorithm in a non-stationary pricing environment, while having at our disposal the advertising curves; in particular, we are dealing with two abrupt changes over the entire time horizon, corresponding to three different phases.

We decided to model such changes in the customer's behaviour by considering three different periods of the year:

- from July to October,in which students are more eager to purchase a new backpack since it coincides roughly with the end of the previous school year and the start of the subsequent one.
- from November to February, which is mostly covered by winter holidays, characterized by a higher probability of purchase associated to more expensive backpacks.
- from March to June, which is considered as regular customer behaviour, with less need for backpacks.

6.1 Sliding Window UCB1 algorithm

The sliding window variant of the UCB1 algorithm is a passive choice for dealing with non-stationary problems, in which the size of the sliding window is fixed a priori and there is no adaptive change detection algorithm employed. It works by computing the usual upper confidence bound only over the samples contained in the window.

Adjusting the window size influences the algorithm's exploration-exploitation balance. A smaller window size encourages more exploration as arms are more likely to lack valid samples within the window, leading to their selection. Conversely, a larger window size promotes the exploitation of the best arms but makes it less likely to detect frequent changes in the underlying distribution.

Theoretically, it is suggested to fix the size of the sliding window proportionally to $\sqrt{T}$, hence we tried these six different multiplying factors:$1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}$. We also attempted larger proportionality constants, though they all systematically provided worse regret than the ones above.

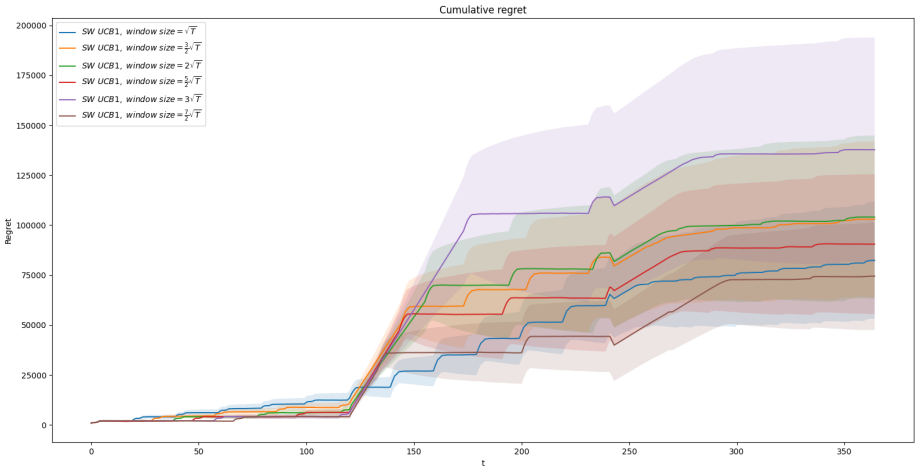


Figure 8: Cumulative regret for different window sizes

As we can gather from the plot, the window size associated to the lowest cumulative regret is the largest one ($\frac{7}{2}\sqrt{T} \approx 66$ days)

6.2 CUSUM UCB1 algorithm

The CUSUM UCB1 algorithm employs an active strategy to detect phase changes, in which the CUSUM values for each arm represent a cumulative measure of deviation from the expected

reward. When these values exceed a specified threshold, the algorithm will switch to exploring different arms. The procedure is defined by selecting four parameters:

- M is the number of samples over which the empirical means are computed. Higher values of M will result in a more accurate expected reward for each arm, but changes are less likely to be detected since we have to gather more samples.
- ϵ is used to represent the sensitivity to changes, adjusting the difference between the current mean and a new sample; larger values imply that more samples are needed to surpass the threshold for the detection.
- h is the threshold, larger values mean larger deviations from the expected reward are needed to detect a change in the underlying rewards' distributions.
- α is the probability to pull a random arm in the current round; larger values increase the exploration capabilities of the algorithm while decreasing the exploitation of the current best arms.

6.2.1 Testing on M

For M we tried six possible values: $10, 20, 30, 50, 80.100$. The fixed values of the other parameters we started with are: $\epsilon = 0.4$, $h = \log T$ and $\alpha = 0.1$.

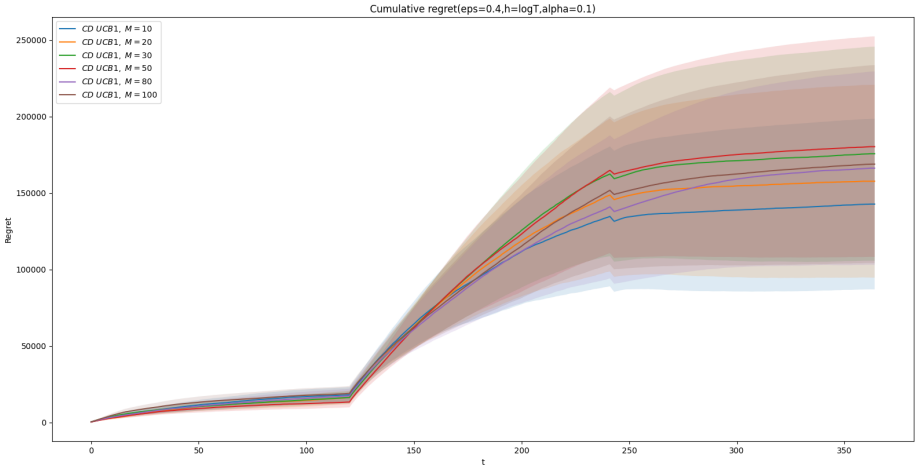


Figure 9: Cumulative regret for different values of M

The results show that, for these initial starting values of the other parameters, having a smaller window size in the CUSUM-UCB algorithm provides a better performance in terms of cumulative regret. In particular $M = 10$ seems like the better choice.

6.2.2 Testing on ϵ

For ϵ, we tried these six different values: $0.1, 0.2, 0.3, 0.4, 0.5, 0.8$. This stems from the fact that we utilized normalized rewards in $[0, 1]$.

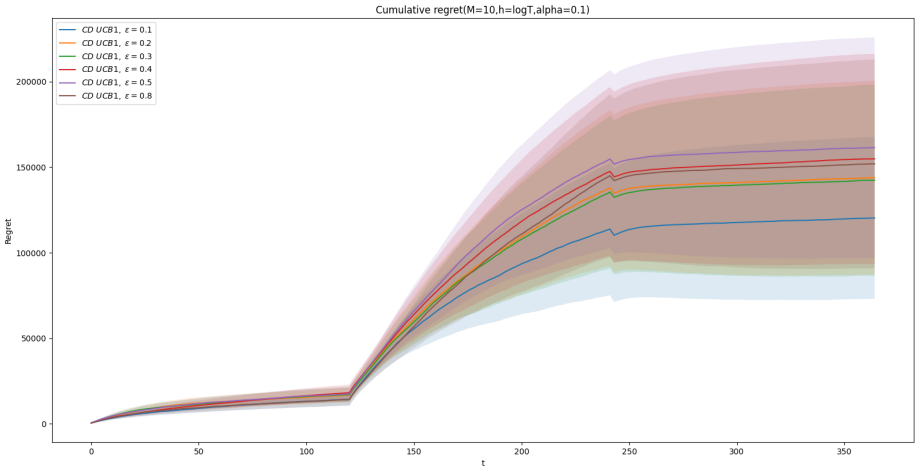


Figure 10: Cumulative regret for different values of ϵ

Despite the plot suggesting that $\epsilon = 0.1$ is the optimal choice, multiple runs have revealed an intriguing phenomenon. Running the algorithm multiple times has led to different values of ϵ achieving the lowest regret, adding complexity to our assessment of this parameter. Despite this uncertainty, we ultimately decided to adopt $\epsilon = 0.1$ as the designated value. This decision was based on its high frequency among the trials and its ability to consistently showcase the most pronounced difference compared to other values during a single run.

6.2.3 Testing on h

For h we tried different values multiplying $\log T$: $1, 2, 4, 5, 8, 10$.

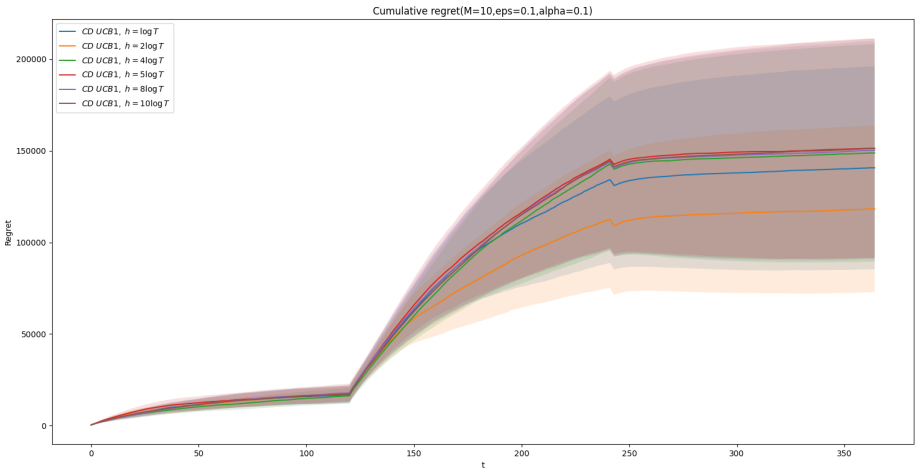


Figure 11: Cumulative regret for different values of h

From the plot it seems that smaller values of the threshold achieve smaller regret, in particular the preferred value is $2 \log T \approx 12$. This could suggest that, in our specific problem setting, employing a smaller threshold facilitates quicker adaptation for detecting changes. The algorithm, in this way, leans towards more frequent exploration of arms, enabling rapid assimilation of valuable information about their reward distributions.

6.2.4 Testing on α

For α, since it represents a probability, we tried these six possible values: $0.05, 0.1, 0.2, 0.3, 0.5, 0.7$

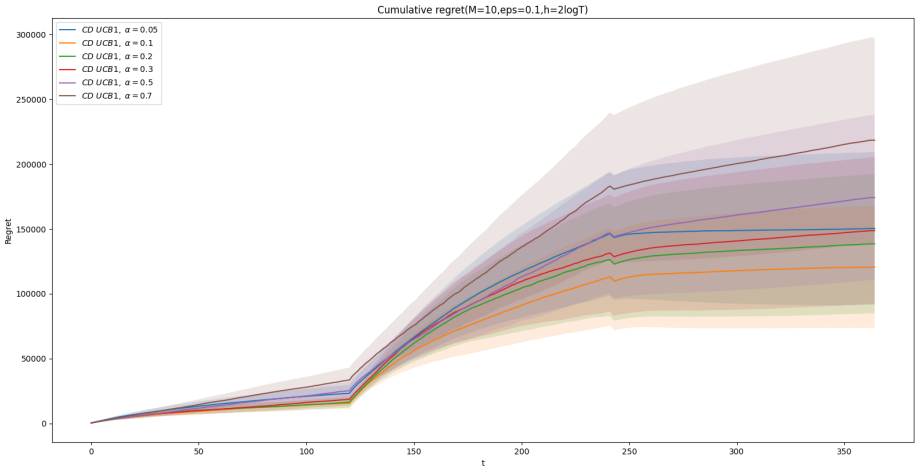


Figure 12: Cumulative regret for different values of α

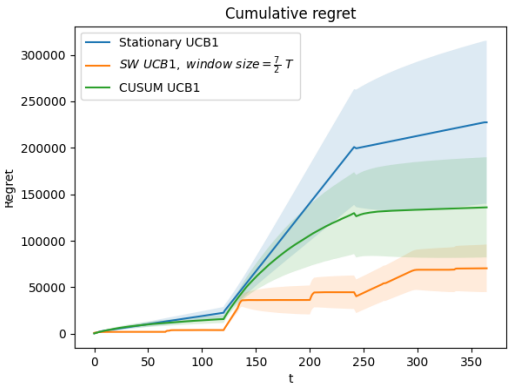
As somewhat expected, values of α above 0.5 provide larger regret, while values around $0.1 - 0.2$ are a better fit. In this way, we still keep a small probability of exploring random arms at each round, while with a higher probability we exploit the current best ones.

6.3 Algorithms Comparison

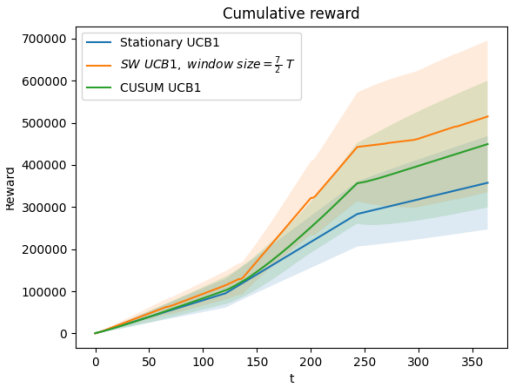
Now we compare the standard stationary UCB1 algorithm with its sliding window and CUSUM variants, choosing the versions achieving the smallest regret based on the sensitivity analysis performed in the previous subsection. A priori, we expect the stationary UCB1 algorithm to perform poorly in a non-stationary environment, since it doesn't possess any detection mechanism that makes it react to a change of the optimal arm; on the other hand, we expect the performances of the sliding window and the CUSUM algorithms to be closer to each other, even though, an active approach should potentially achieve higher rewards over the whole time horizon.

Referring to the theoretical results, if the number of change points is $C = O(T^\alpha)$, $\alpha \in [0, 1)$, then we have an upper bound on the expected regret for the SWUCB algorithm of order $O(T^{\frac{1+\alpha}{2}}\sqrt{\log T})$, provided that the size of the sliding window is approximately equal to $2\sqrt{\frac{T \log T}{C}}$(in our case, using a window size of ≈ 66 samples we satisfy said assumption). Similarly, we have an upper bound on the expected regret of the CUSUM-UCB algorithm of order $O(CT \log \frac{T}{C})$, if the values for the threshold and α are chosen properly as shown in the article Liu et al. [2017] *"A Change-Detection based Framework for Piecewise-stationary Multi-Armed Bandit Problem"*.

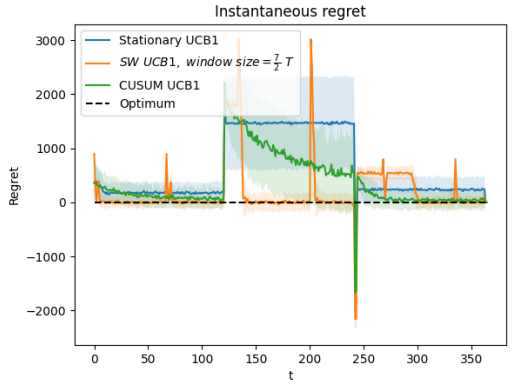
Interestingly, the sliding window UCB1 algorithm exhibits superior performance when compared to its CUSUM counterpart, specifically in the rapid identification of the new optimal arm after an abrupt change. One plausible explanation for this outcome could be the challenge of simultaneously setting four parameters in our version of the CUSUM algorithm, likely hindering its ability to reach its full potential, rendering the sliding window UCB1 algorithm the more favorable choice. As expected, nonetheless, the three algorithms showcase a sublinear growth in the regret, with the stationary UCB1 being the clear worst-performing one.



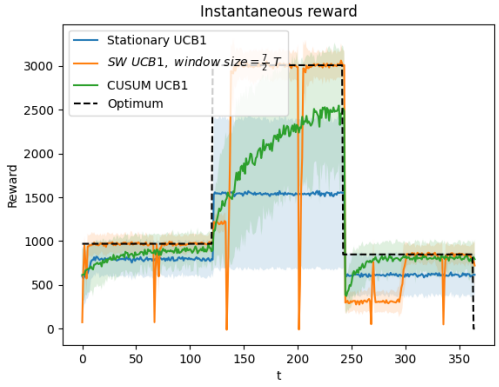
(a) Cumulative regret.



(b) Cumulative reward.



(c) Instantaneous regret.



(d) Instantaneous reward.

Figure 13: Results of algorithms' comparison.

7 Step 6

In this last section, we are asked to deal with *non-stationary environments* with many abrupt changes and **fixed bid**. In particular, to do that, we must exploit the **EXP3** algorithm, which is devoted to deal with adversarial settings.
We can define an **adversarial environment** as an environment in which we have an opponent (adversary) selecting the probability distribution of the arms in order to minimize the utility of our player.
If the non-stationary environment is *unrestricted*, in the sense that we don't have any information on how the probability distribution changes during time, then the setting is equivalent to an adversarial environment.

7.1 EXP3 algorithm

The **EXP3 algorithm** aims to strike a balance between exploiting arms with high estimated rewards and exploring arms to discover potentially better rewards.
Furthermore EXP3 is based on *importance-weighted sampling*, whereas the other considered algorithms are based on unweighted rewards.
Importance-weighted sampling provides certain advantages when moving to more complex problems, such as stochastic multi-armed bandits with side information.
This technique encourages the exploration of promising arms even when facing adversarial conditions. Arms with higher rewards receive a higher importance weight, thereby increasing their probability of being chosen in the future.
This algorithm in the adversarial setting provides a $O(\sqrt{KT})$ in expectation regret bound, as proved by Auer et al. [2003] *"The nonstochastic multiarmed bandit problem. SIAM Journal of Computing, 32(1), 2002b"*.

By adjusting the exploration parameter γ and using adaptive reward updates, the algorithm can handle non-stationary environments and improve its performance over time. The parameter $\gamma \in [0, 1]$ has the following meaning:

- close to 0: gives more probability according to the weights and therefore to the rewards obtained by the game
- close to 1: the probability distribution over the arms tends to be uniform

The algorithm works in the following way:

- Fix the parameter $\gamma_t = \gamma$
- For each round t:
  - **Update** the **trusts probabilities** according to EXP3 and the parameter γ_t in the following way:
$$trusts'_k(t+1) = (1-\gamma_t) \cdot w_k(t) + \gamma_t \cdot \frac{1}{K}$$
$$\text{trusts}(t+1) = \frac{trusts'_k(t+1)}{\sum_{k=1}^{K} trusts'_k(t+1)}$$
    where K is the number of arms.
  - Make a **random arm choice**, with $probabilities = trusts$
  - **Get a reward**: accumulate rewards of the chosen arm k, then update the weights $w_k(t)$ and normalize the weights:
$$\widetilde{r}_k(t) = r_k(t)$$
$$w'_k(t+1) = w_k(t) \times \exp\left(\frac{\tilde{r}_k(t)}{\gamma_t N_k(t)}\right)$$
$$w(t+1) = \frac{w'_k(t+1)}{\sum_{k=1}^{K} w'_k(t+1)}$$

After some value exploration, we decided to set constant $\gamma_t = 0.05$ as can be seen in Figure 14. We tested also decreasing γ_t with time, firstly as in the following formula, as reported in Bubeck and Cesa-Bianchi [2012] *"Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems"*:

$$\gamma_t = \min\left(\frac{1}{K}, \sqrt{\frac{\log(K)}{tK}}\right)$$

and then as in this second one:

$$\gamma_t = \min\left(1, \sqrt{\frac{K\log(K)}{(e-1)t}}\right)$$

However, the obtained results with these last two formulas were worse with respect to the other tested values for the parameter γ_t.
For the sake of readability, we plotted in Figure 14 only the six best-performing ones. The sensitivity analysis of parameter γ_t has been carried out for both settings (i.e. the one with three phases and the one with more phases), yielding to the same results.

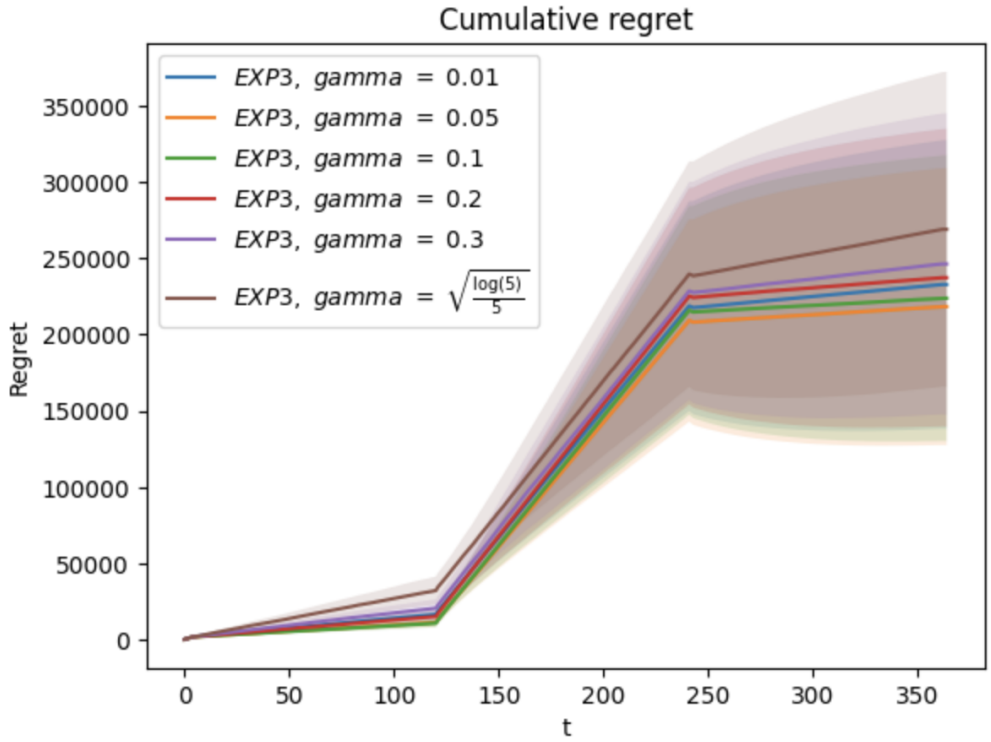
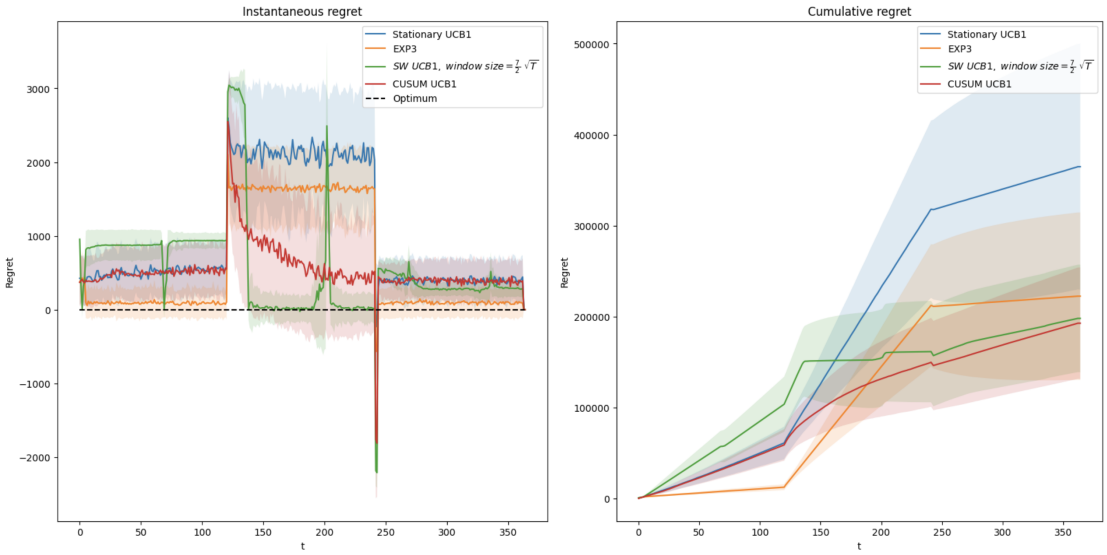


Figure 14: Sensitivity analysis of parameter γ.

7.1.1 3 phases setting

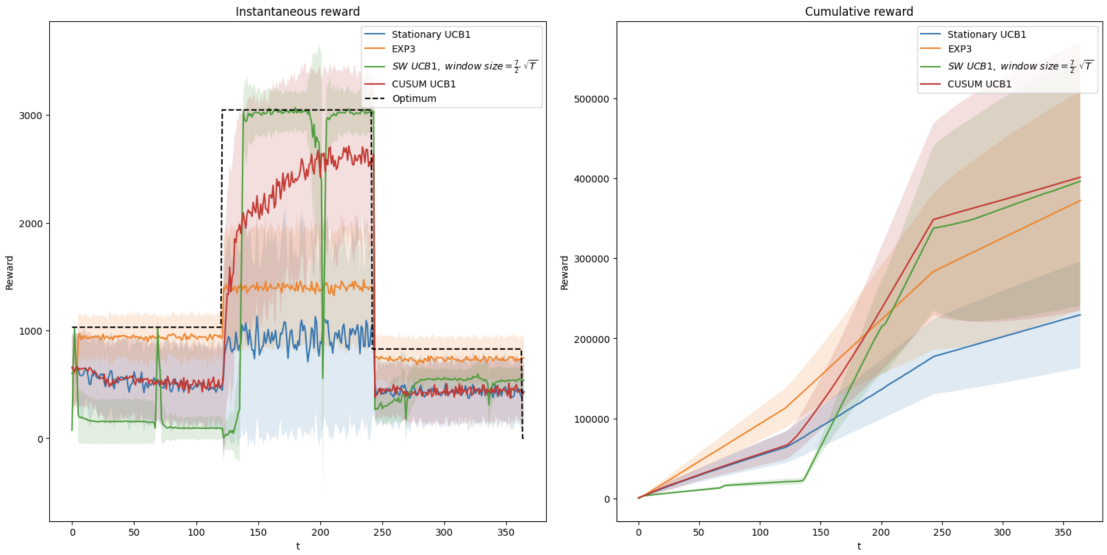
The first setting we are requested to consider consists of a simplified version of step 5.
We expect that **EXP3** performs worse than the two non-stationary versions of **UCB1**.

We analyzed the three phases setting of step 5, by exploiting the same probabilities of that step and running the EXP3 algorithm as described before over 100 experiments and comparing it with UCB1, SW-UCB1, and CUSUM-UCB1.
The configuration of the sliding-window UCB1 and CUSUM is coherent with the sensitivity analysis conducted in step 5. Therefore, $window\ size = \frac{7}{2}\sqrt{T}$ for the SW-UCB1 and $M = 10,\ \epsilon = 0.1,\ h = 2\log T,\ \alpha = 0.1$ for what concern the CUSUM-UCB1.
We achieved the expected outcomes, in particular, the two non-stationary flavors of UCB1 outperform the EXP3 algorithm, which suffers from a linear regret due to its constant exploration until the end of the game.
On the other hand, the latter provides a lower cumulative regret than the stationary UCB1. These results can be observed in the subplots of Figure 15.



(a) 3 phases: instantaneous regret.

(b) 3 phases: cumulative regret.



(c) 3 phases: instantaneous reward.

(d) 3 phases: cumulative reward.

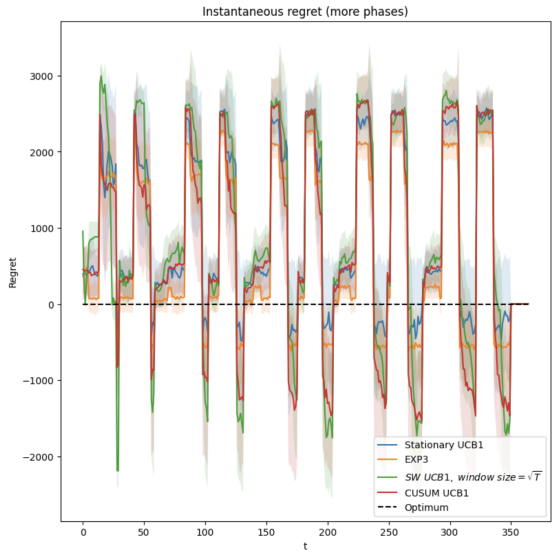
Figure 15: Results obtained in the 3 phases scenario.

7.1.2 More phases setting

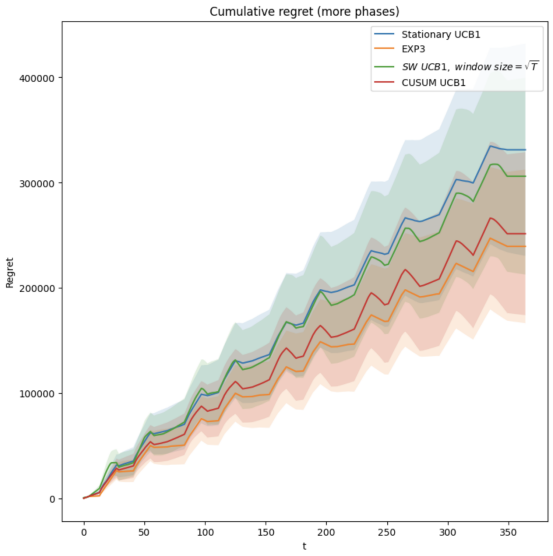
Lastly, we are requested to consider a different non-stationary setting with a higher non-stationary degree. In particular, we must consider 5 different phases, each one associated with a different optimal price.
These phases cyclically change with high frequency.
Here, we expect **EXP3** to outperform the non-stationary version of **UCB1**.

In this second setting, we decided to involve a total of 25 phases, by repeating cyclically 5 times the same phases. In each subset of 5 phases, the first 3 have the same probability distribution of the setting with 3 phases.
We repeated the sensitivity analysis for this new setting and we observed that we were able to obtain better results by modifying the window size of the SW-UCB1 from $\frac{7}{2}\sqrt{T}$ to $\sqrt{T}$ while maintaining all the other parameters with the same values as before.
As we can observe in figure 16, EXP3 outperforms all the flavors of UCB1 and, the second best performing one is the CUSUM-UCB1 algorithm.
The EXP3 algorithm outperforms CUSUM-UCB1 due to its robustness to adversarial changes, indeed, its adaptability allows it to continue exploring new options and adapt to changes.
Moreover, EXP3 is capable of adapting the learning rate, i.e., the amount of exploration and updating of action probabilities, based on the information collected over time. This allows it to

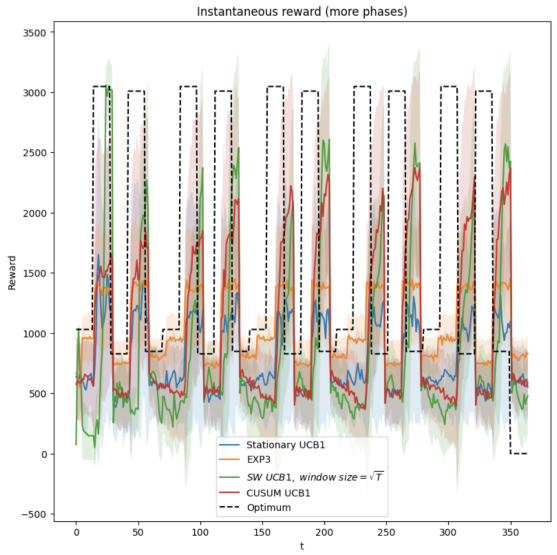
balance the risk of exploring new options and exploiting high-reward options in an adversarial environment.
CUSUM-UCB1 has a fixed methodology for calculating the threshold value and detecting out-of-control values, which may not be ideal for adapting to changes and challenges present in an adversarial environment.



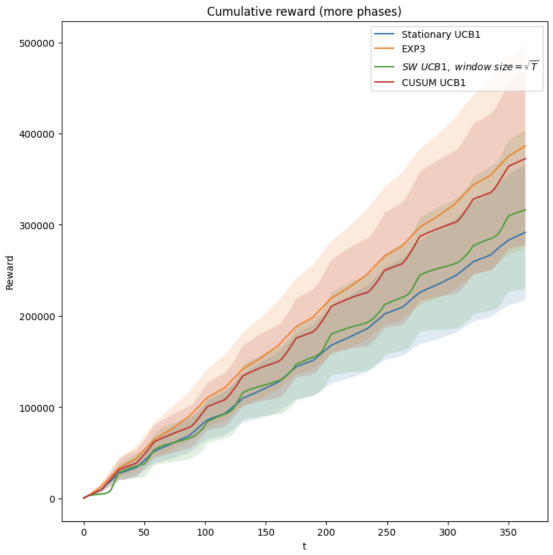
(a) 25 phases: instantaneous regret.



(b) 25 phases: cumulative regret.



(c) 25 phases: instantaneous reward.



(d) 25 phases: cumulative reward.

Figure 16: Results obtained in the 25 phases scenario.

References

Peter Auer, Nicolò Cesa-Bianchi, Y Freund, and RE Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32:48–77, 01 2003.

Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. In *Foundations and Trends® in Machine Learning*, pages 1–122, January 2012. URL `https://www.microsoft.com/en-us/research/publication/regret-analysis-stochastic-nonstochastic-multi-armed-bandit-problems/`.

Fang Liu, Joohyun Lee, and Ness B. Shroff. A change-detection based framework for piecewise-stationary multi-armed bandit problem. *CoRR*, abs/1711.03539, 2017. URL `http://arxiv.org/abs/1711.03539`.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process bandits without regret: An experimental design approach. *CoRR*, abs/0912.3995, 2009. URL `http://arxiv.org/abs/0912.3995`.