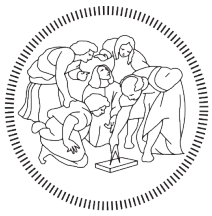
Online Learning Applications

Pricing and Advertising

Valentina Abbattista
Fiammetta Artioli
Ossama El Oukili
Fausto Lasca
Oswaldo Jesus Morales Lopez

September 22^{nd}, 2023



POLITECNICO
MILANO 1863

Contents

1 Chapter 0

1.1 Motivations and environment design

For the purpose of this project, we have focused on the real world scenario in which an e-commerce wants to increase its revenues by improving its pricing and advertising strategy.

The chosen e-commerce sells a backpack which can be used for different purposes and therefore can be bought by different classes of users.

Users are characterized by the following two binary features:

- F1: Age
  We can distinguish users which are below 18 years old from users which are above 18 years old
- F2: Profession
  We can distinguish students and workers

According to these features, users can be grouped in three different classes:

- C1: Students which are older than 18 years old
- C2: Students which are younger than 18 years old
- C3: Workers which are older than 18 years old

These three user classes can be targeted differently based on the intended usage of the backpack, which can be the following:

- C1: University backpack
- C2: School backpack
- C3: Laptop backpack

We assume that the production cost of the backpack is 40, and the margin is computed as $price - cost$.

1.2 Construction of the environment

For every user class we specified:

- A concave curve expressing the average dependence between the number of clicks and the bid which is the same for every class
- A concave curve expressing the average cumulative daily click cost for the bid which is the same for every class
- 5 different possible prices for the product. They are [50, 60, 70, 80, 90] and they are the same for every class.
- A function expressing how the conversion probability varies as the price varies:
  - For the class C1 we are assuming that students that are older than 18 years old are interested in buying the item if the price is low, thus there is a high conversion rate for prices 50 and 60 and a low conversion rate for the other prices.
  - For the class C2 we are assuming that students that are younger than 18 years old are interested in buying the item if the price is very low, thus there is a high conversion rate only for price 50.
  - For the class C3 we specified a function that initially increases, then has a peak on 70 and then decreases. This is because we are assuming that workers that are older than 18 years old are interested in buying a good quality product, but at the same time do not want to buy a too expensive product.

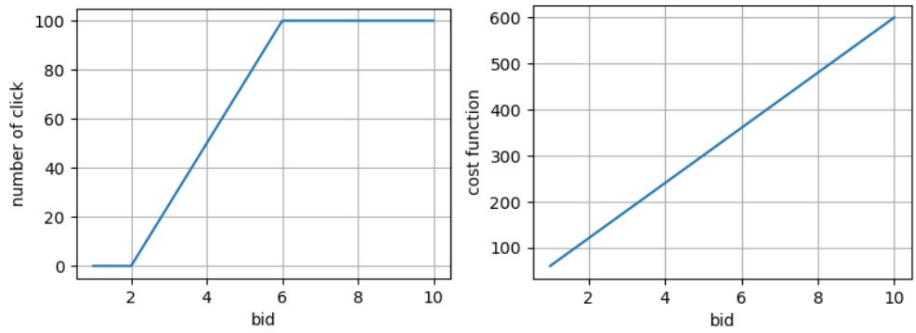


Figura 1: Number of clicks and cost functions

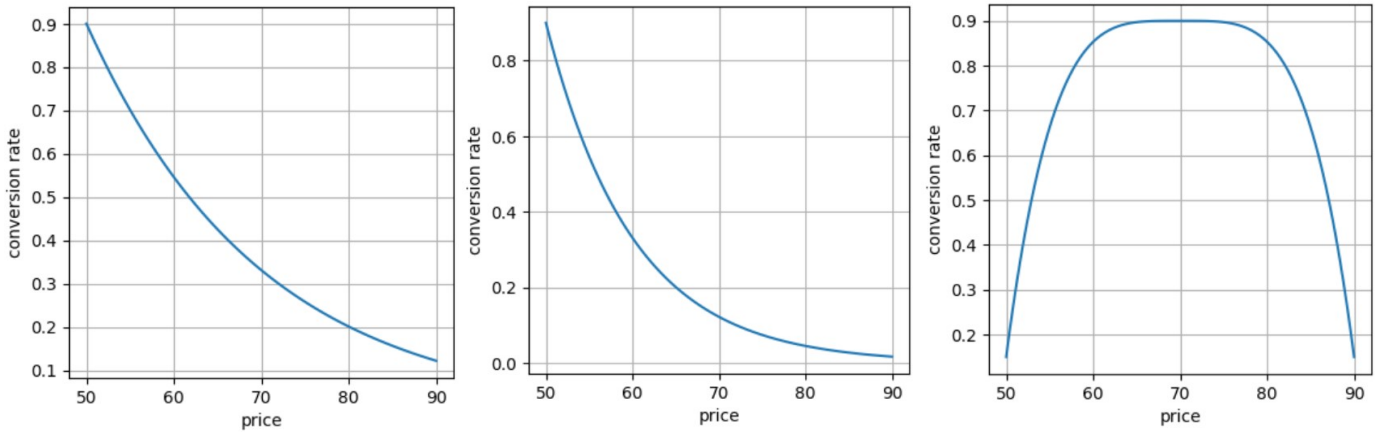


Figura 2: Conversion rate functions for classes C1, C2 and C3

2 Chapter 1: Learning for pricing

2.1 Environment

In this step we are considering the case in which all the users belong to class C1 and we are assuming that the curves related to the advertising part of the problem are known, while the curve related to the pricing problem is not. Thus we fix a bid (in this case BID=6) and we compute the number of daily clicks and the cumulative daily costs.

In this setting, given a pulled arm the reward given by the environment is the following:

$$B(number\_of\_daily\_clicks, conversion\_rate(pulled\_arm)) \cdot margin - cumulative\_daily\_costs$$

where $B(n, k)$ is the binomial distribution.

In our example, the best price for the backpack in terms of $conversion\_rate \cdot margin$ for class C1 is 60 and the optimal reward is given by the following:

$$n\_daily\_clicks(BID) \cdot conversion\_rate \cdot margin - cumulative\_daily\_costs(BID)$$

2.2 UCB algorithm

To find which is the best price for the backpack we can exploit the UCB algorithm where we learn the value $conversion\_rate \cdot margin$ for each arm and we find the arm that maximizes it.
We run this version of UCB algorithm with a time horizon $T = 365$ to simulate the learning process over a year. At each round:

1. The arm with the highest upper confidence bound is pulled

2. Given the pulled arm, the environment returns a reward
3. The upper confidence bounds are updated according to UCB algorithm

This is repeated for 1000 experiments and at the end the results of all the experiments are averaged. As we can see in the figure below, the achieved regret is sublinear and the algorithm learns the optimal price in a few days.

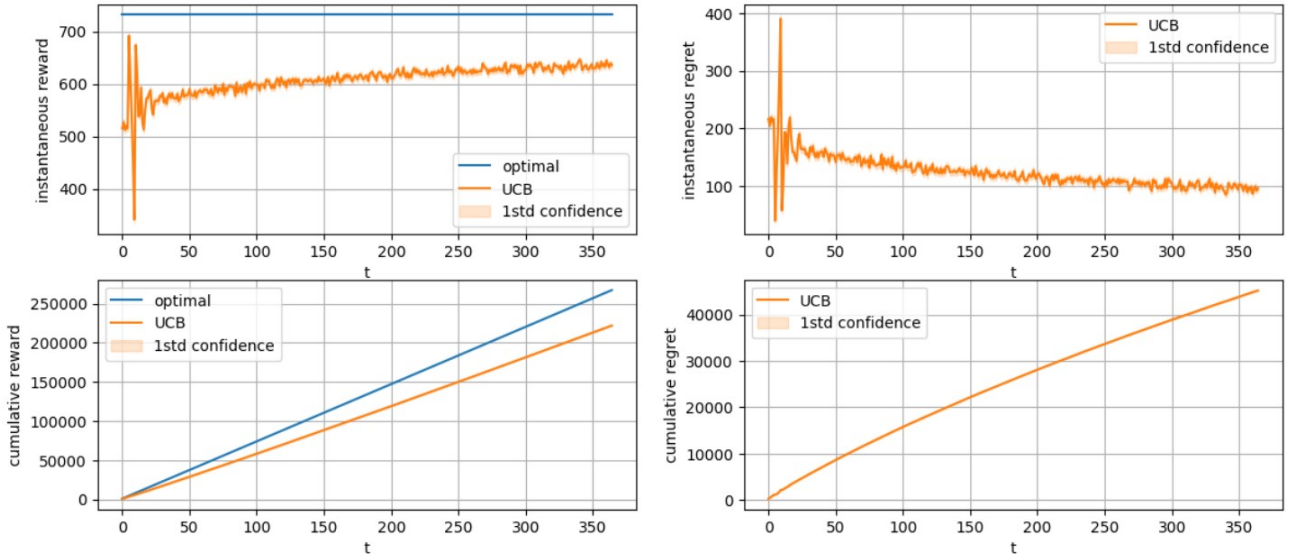


Figura 3: Instantaneous reward, instantaneous regret, cumulative reward and cumulative regret over a sufficient number of experiments.

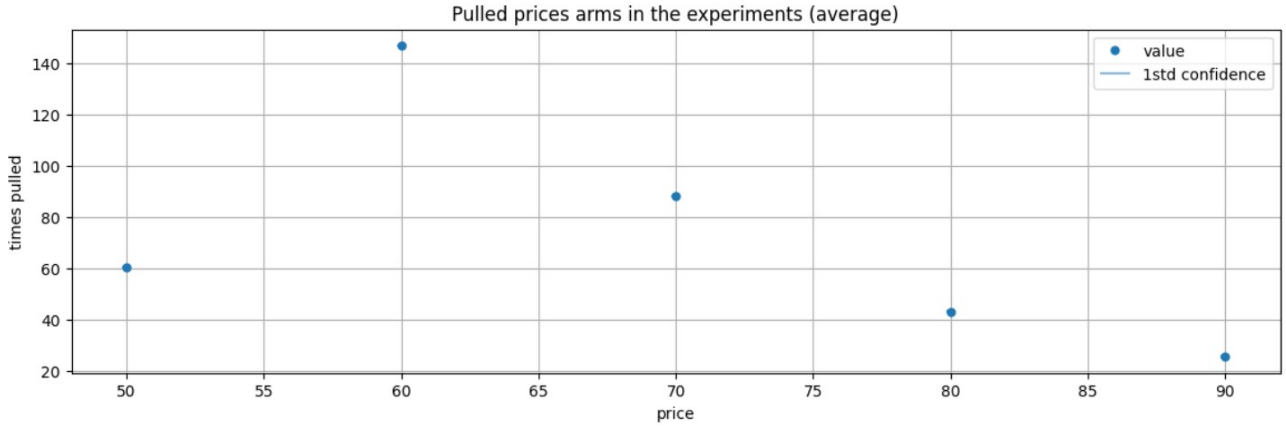


Figura 4: The optimal price is the most pulled arm

Support interval In UCB algorithm the choice of the support is significant. The support is an interval that determines the value of c that is used to update the confidences of the arms, where $c = \sqrt{2}(max(support) - min(support))$. From a theoretical point of view, the start and the end of this interval should be respectively the minimum and the maximum values for the value that the algorithm is learning (in our case is $conversion\_rate \cdot margin$), but from a more practical point of view the real interval is often smaller, thus it can be determined in an empirical way. We tried some different intervals and in our case the algorithm works well with a support $\sqrt{2}(0, 12)$.

2.3 Thompson Sampling algorithm

To find which is the best price for the backpack we can exploit the TS algorithm where we learn the conversion rate value for each arm and we try to find the arm that maximizes it.
We run this version of TS algorithm with a time horizon $T = 365$ to simulate the learning process over a year. At each round:

1. A sampled conversion rate based on beta distributions is drawn for each arm, then the arm with the maximum $sampled\_conversion\_rate \cdot margin$ is pulled
2. Given the pulled arm, the environment returns a reward
3. The beta distribution of the pulled arm is updated according to TS algorithm

This is repeated for 1000 experiments and at the end the results of all the experiments are averaged.

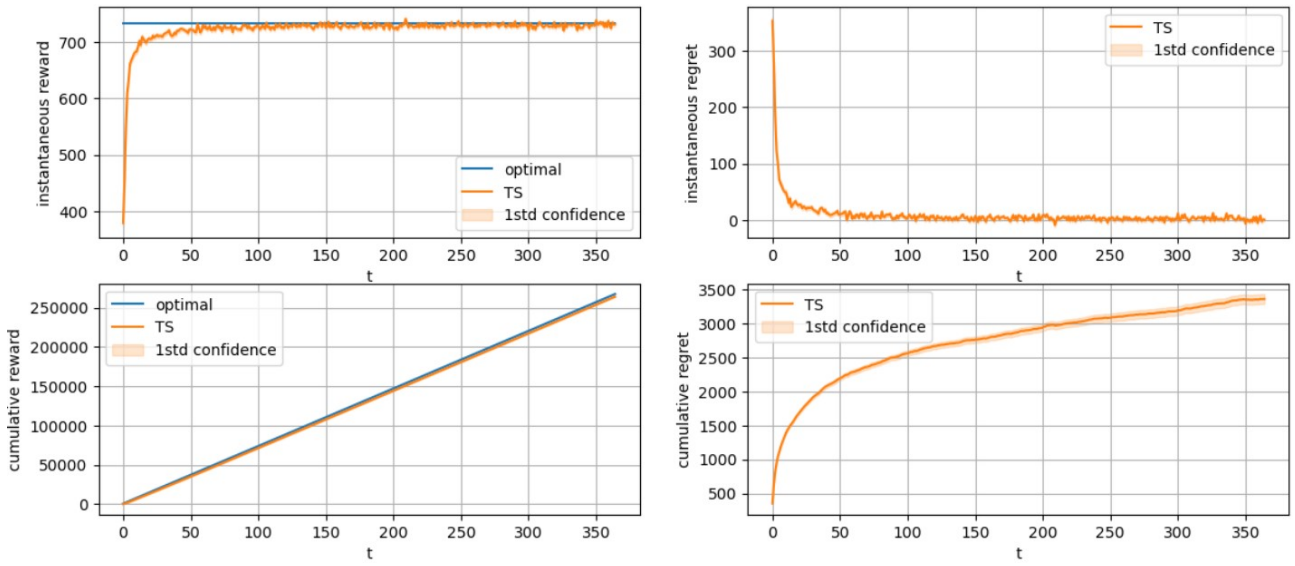


Figura 5: Instantaneous reward, instantaneous regret, cumulative reward and cumulative regret over a sufficient number of experiments

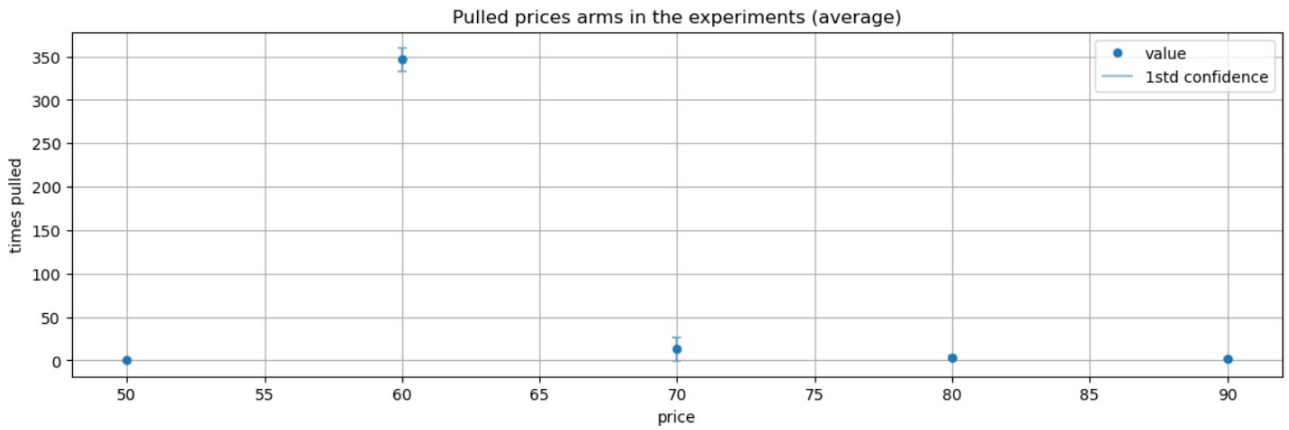


Figura 6: The optimal price is the most pulled arm

Both UCB and TS algorithms have a sublinear regret, but we can see that TS has better performance and this is because it uses a generative approach.

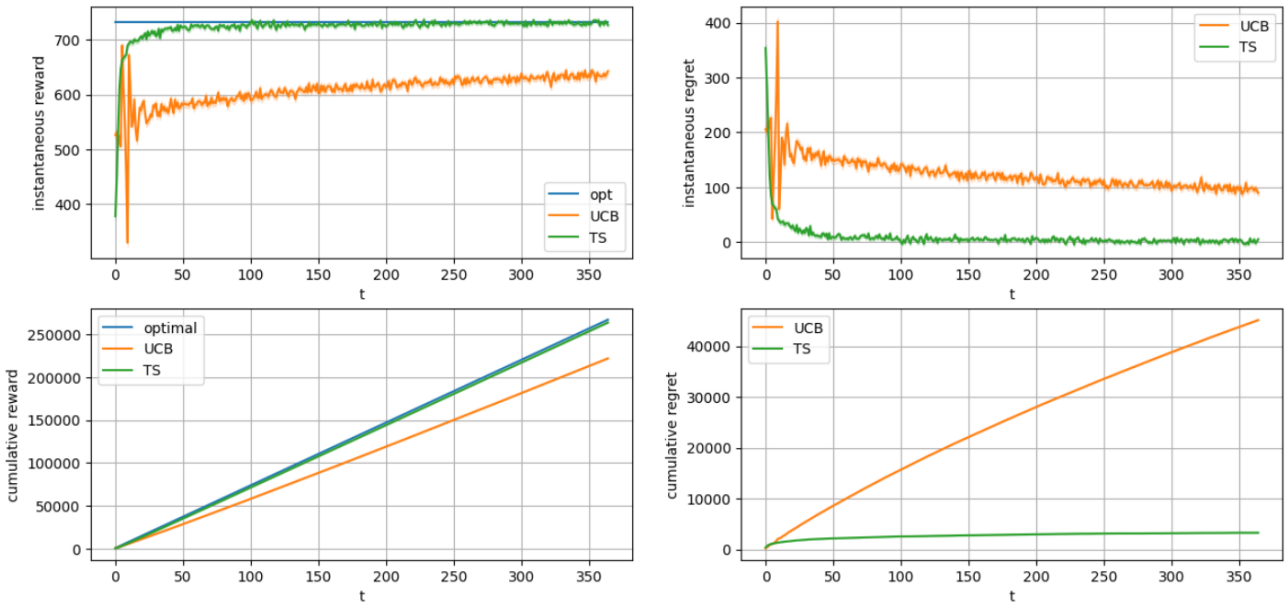


Figura 7: Results comparison

3 Chapter 2: Learning for advertising

3.1 Models and Methods

In this part, only one class of users is considered as in step 1, but the framework is different: the curve related to the pricing part is known, while the curves related to the advertising problem, which are the number of daily clicks and the cumulative daily cost, are not. We assume that, at each time t, when an arm is pulled, we are able to observe, other than the reward, the realization of the advertising curves on the specific bid.

We modelled both advertising curves as **Gaussian Processes**. We denote with N_b^t and C_b^t the random variables of the number of daily clicks and the cumulative daily costs $\forall b \in BIDS$ and $t \in T$.

GPUCB algorithm For the UCB algorithm we use the confidence intervals given by the modelled GP of each curve. In particular, for every bid, we have an estimate of the mean and the standard deviation of both curves. So, at each round t, we pull the bid that returns the maximum upperbound reward R_b, defined as:
$UB(R_b) = UB(N_b) \cdot conversion\_rate \cdot margin - UB(C_b)$

GPTS algorithm At each time t, we sample from both GP models of the curves and choose the arm that maximizes the sampled reward, defined as: $R_b = n_b \cdot conversion\_rate \cdot margin - c_b$ where n_b and c_b are the sampled values of the number of daily clicks and the cumulative daily costs, respectively.

To manage the GP curves, we relied on the library `sklearn` and we used as kernels a constant and a RBF kernel, whose parameters are found by an internal algorithm of sklearn (given an initialization for the search).

3.2 Numerical Results

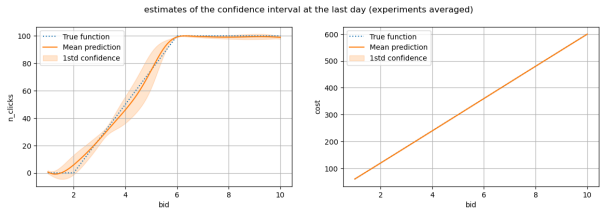


Figura 8: UCB: advertising curves at t=365

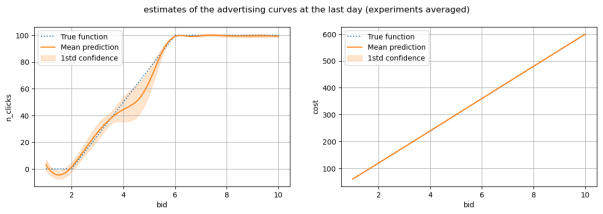


Figura 9: TS: advertising curves at t=365

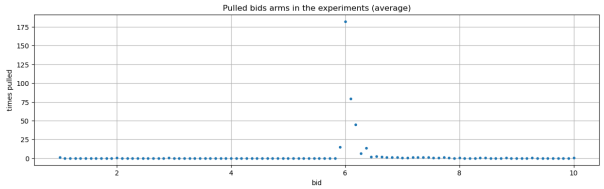


Figura 10: UCB: bids pulled at t=365

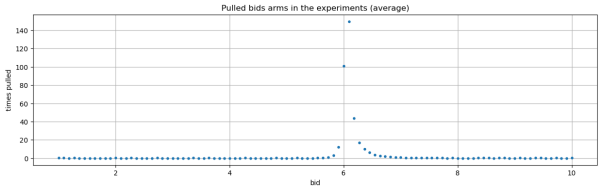


Figura 11: TS: bids pulled at t=365

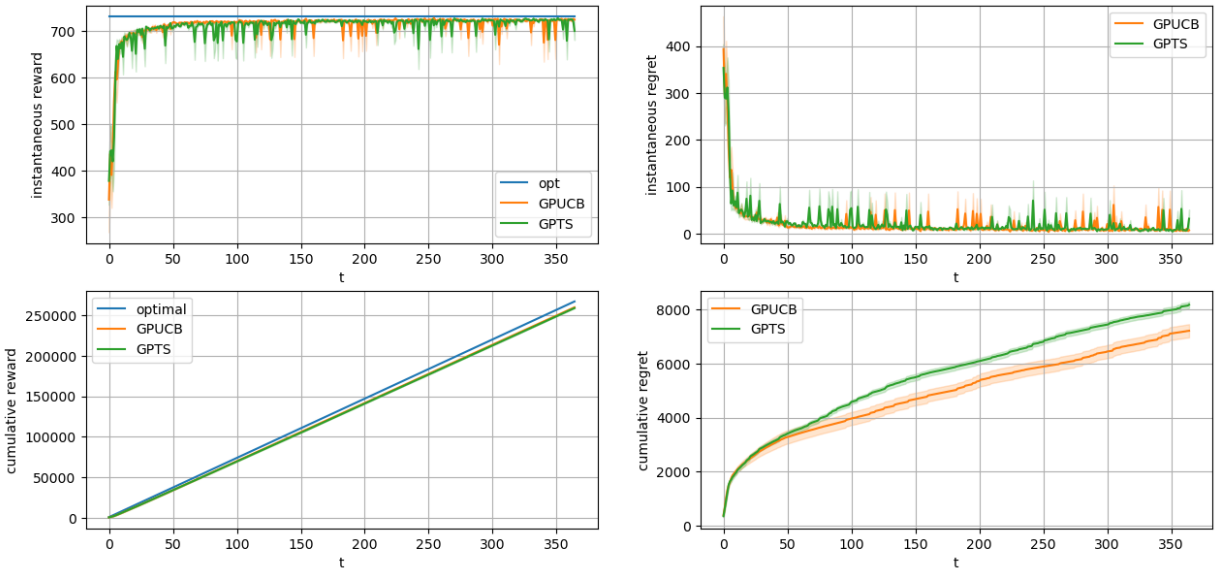


Figura 12: UCB vs TS

3.3 Conclusions

Both algorithms, GPUCB and GPTS, learn pretty well the optimal arm and also the underlying unknown advertising curves. From the pulled arms plot, we can see that both algorithms pull mostly the optimal arm.

4 Chapter 3: Learning for joint pricing and advertising

4.1 Environment

In this step, we are considering the case in which all the users belong to class C1 and we are assuming that the curves related to both the pricing and the advertising problem are unknown. Thus, to resolve this step, we can combine the techniques that we proposed in step 1 and 2 to learn both the pricing and the advertising parts.
In our case, the optimal price is 60 as in step 1 and the optimal bid is 6 as in step 2.

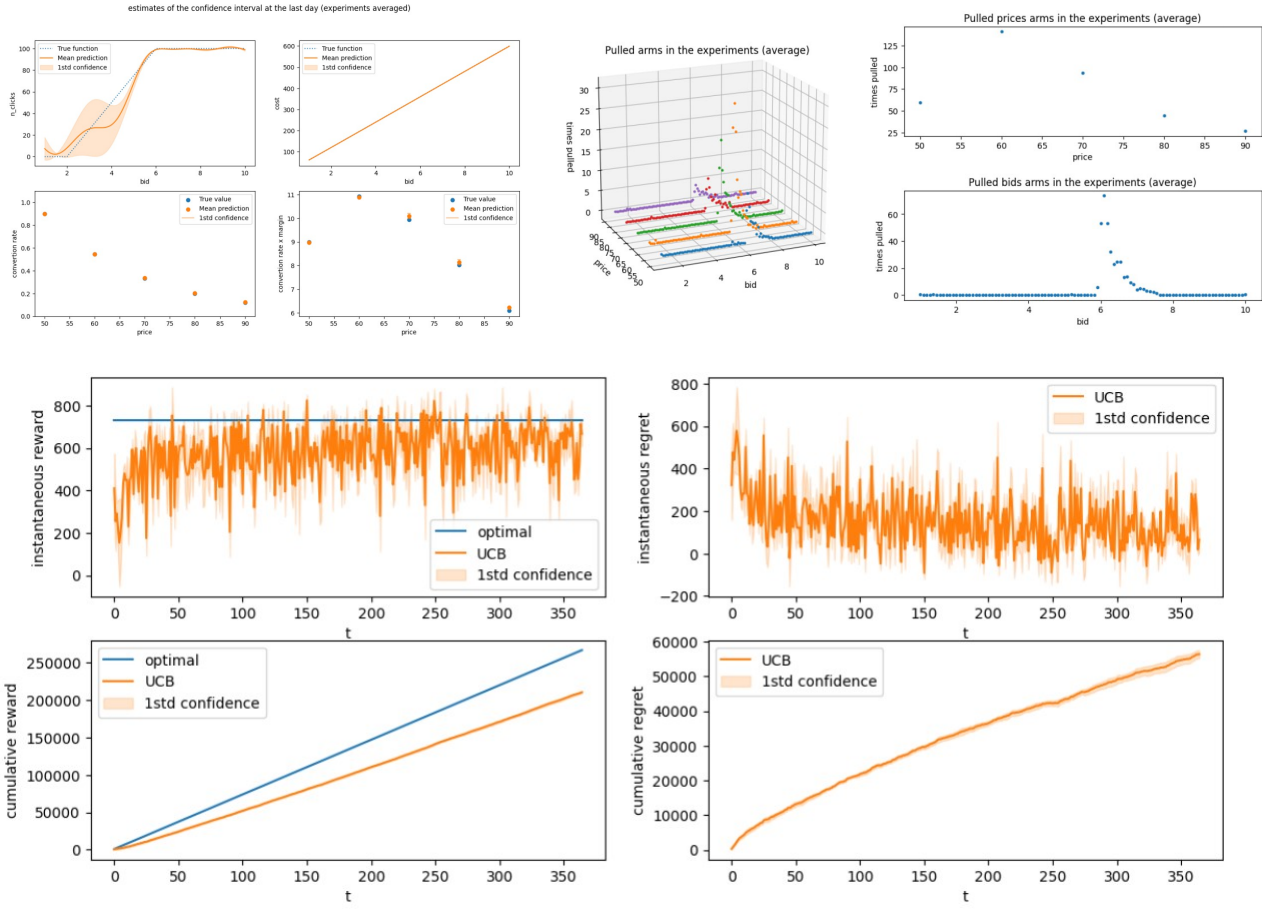
4.2 Algorithms

UCB algorithm We use the UCB algorithm to learn the value $conversion\_rate \cdot margin$ as in the step 1, and we combine it with two GPUCB learners that respectively learn the number of daily clicks and the cumulative daily costs related to the advertising part as in the step 2.

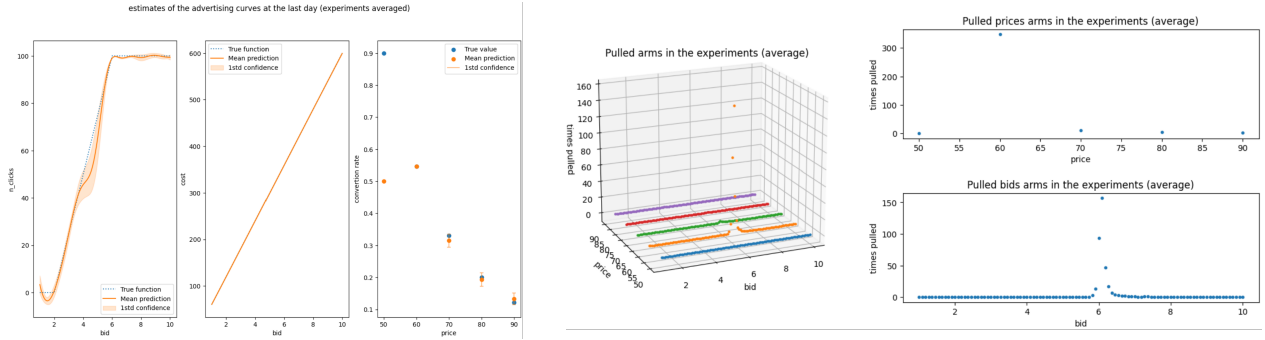
TS algorithm We use the TS algorithm to learn the value of the conversion rate and than pull the arm that maximizes the value $sampled\_conversion\_rate \cdot margin$ as in the step 1, and we combine it with two GPTS learners that respectively learn the number of daily clicks and the cumulative daily costs related to the advertising part as in step 2.

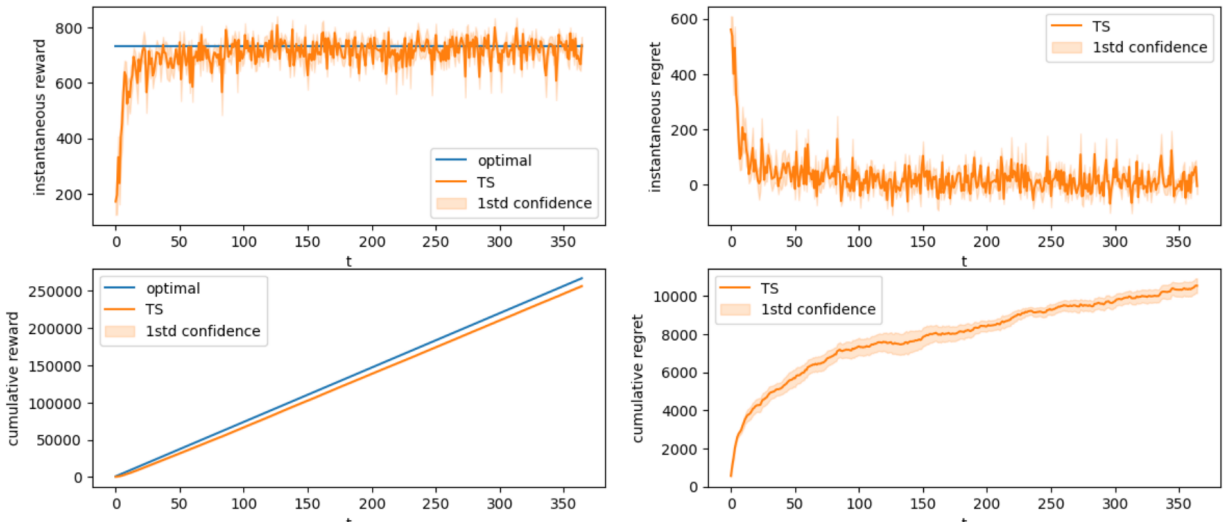
4.3 Numerical results

- UCB approach:



- TS approach:





4.4 Conclusions

In this part the TS approach performs way better than the UCB approach. Indeed, we can observe that the TS method pulls mostly the optimal rewards.

5 Chapter 4: Contexts and their generation

5.1 Models and Methods

In this case we consider a scenario with different types of users. The framework is the following:

- Each class can have its own pricing and advertising curves, which are unknown to us;
- We know the feature space, which in this case is $\{0, 1\}^2$;
- We can set different bids and prices for each subset of the feature space;
- We assume that, at each round, we interact with all types of users;
- We consider two scenarios:
  - We know a priori that there are only 3 types of users, each one characterized by different curves
  - We don't have any a priori information on the structure of contexts.

In this step, we have to compare how the learning behaves in 3 different cases:

- The context structure is known;
- the context structure is unknown and no context generation algorithm is used;
- the context structure is unknown and we use a context generation algorithm.

5.1.1 Environment

At each round, we choose a context structure (i.e., a partition of the feature space) and for each context (i.e., a set of the partition) we choose the arms (bid and price) to be pulled. Then, the environment returns, for each user class, the number of daily clicks, the cumulative daily costs, the number of converted clicks and the reward.
Provided an environment that manages each user class separately (as the one in step 3), the final environment is simply the collection of these environments for single user classes. With an appropriate algorithm, the environment determines which user class belongs to a certain context and pulls the arms specified for the context to which the user class belongs.

5.1.2 Learners

For both UCB and TS approaches, we have split the problem into 3 parts in an encapsulated way:

Single Context Learner This class is like the learner in the step "Learning for joint pricing and advertising". In particular it's used to learn the learners' parameters of what we think it's a Context.

Fixed Context Learner In this level, we define the context structure and, for each context in the context structure, we define a context learner (Single Context Learner). The pull method collects all the pulls that come from the context learners and the update method manages and collects all the data (that can belong to different classes of customers) and passes it to the corresponding context learner. However, this level does not change the context structure.

Contexts' Greedy Learner This level is an extension of the previous one; it adds the possibility of changing the context structure by implementing a greedy algorithm that changes, if needed, the context structure every 14 days:

- Starting from the current context structure, we create candidates of partitions of the feature space to be the new context structure. For example: if the context structure is $\{0,1\}^2$ then the candidates are the following two:
  - by splitting feature F1: $\{0\} \times \{0,1\}$, $\{1\} \times \{0,1\}$
  - by splitting feature F2: $\{0,1\} \times \{0\}$, $\{0,1\} \times \{1\}$
- We fit each of the candidates' contexts with the data we have collected
- We evaluate each of the candidates context structure by taking the lower bound of the reward of the best arm:
  - for the conversion rate, we take the Hoeffding bound: the empirical mean $- \sqrt{-\frac{log(0.95)}{2|Z|}}$, where Z is the set data;
  - for the number of daily costs and the cumulative daily sum, since they are modeled using a GP, we use the empirical mean $\pm$ 1.65 * the standard deviation given by the GP (+ or - accordingly to their sign in the reward formula)
- Once we have the lower bounds, we compare the lower bound of the current context structure with those of the candidates, and we keep the context structure with the highest lower bound.

5.2 Numerical results

- Final results in terms of reward and regret:

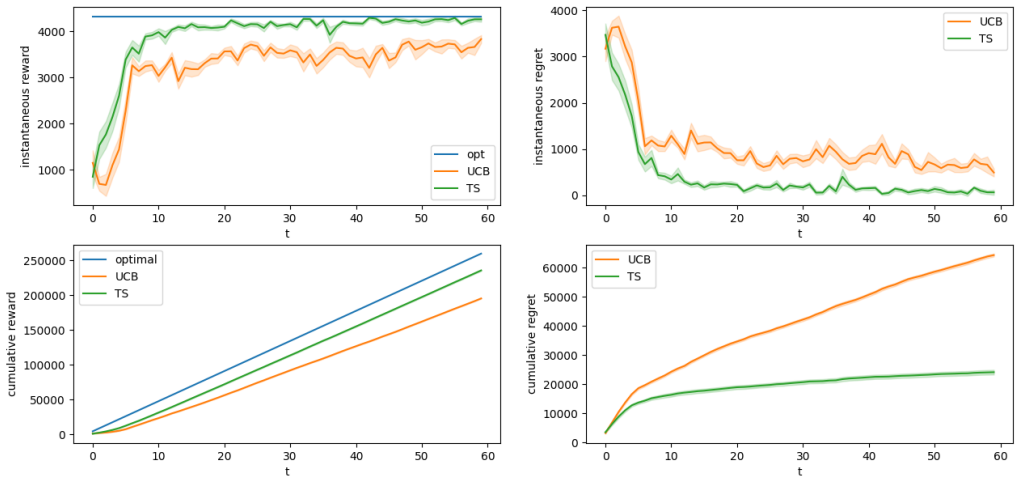


Figura 13: UCB vs TS in the case of known context structure

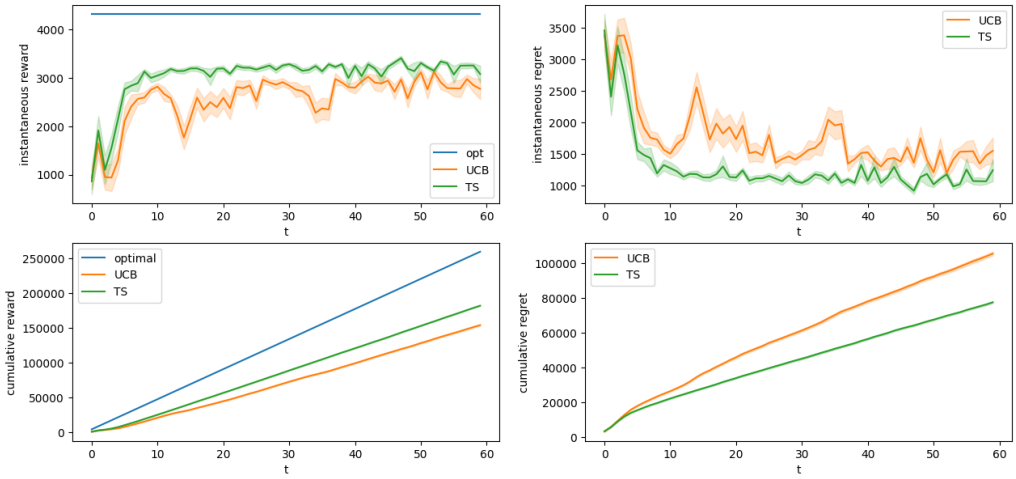


Figura 14: UCB vs TS in the case of unknown context structure and no context generation algorithm

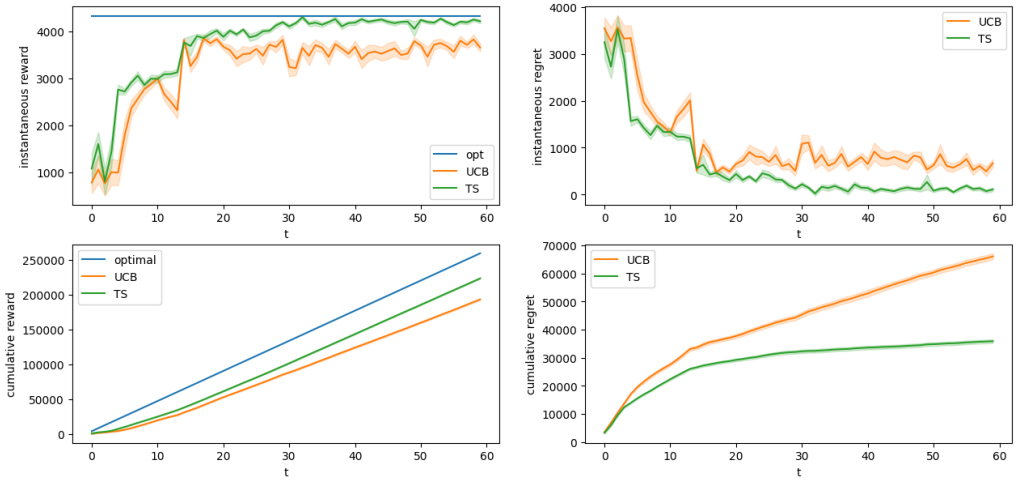
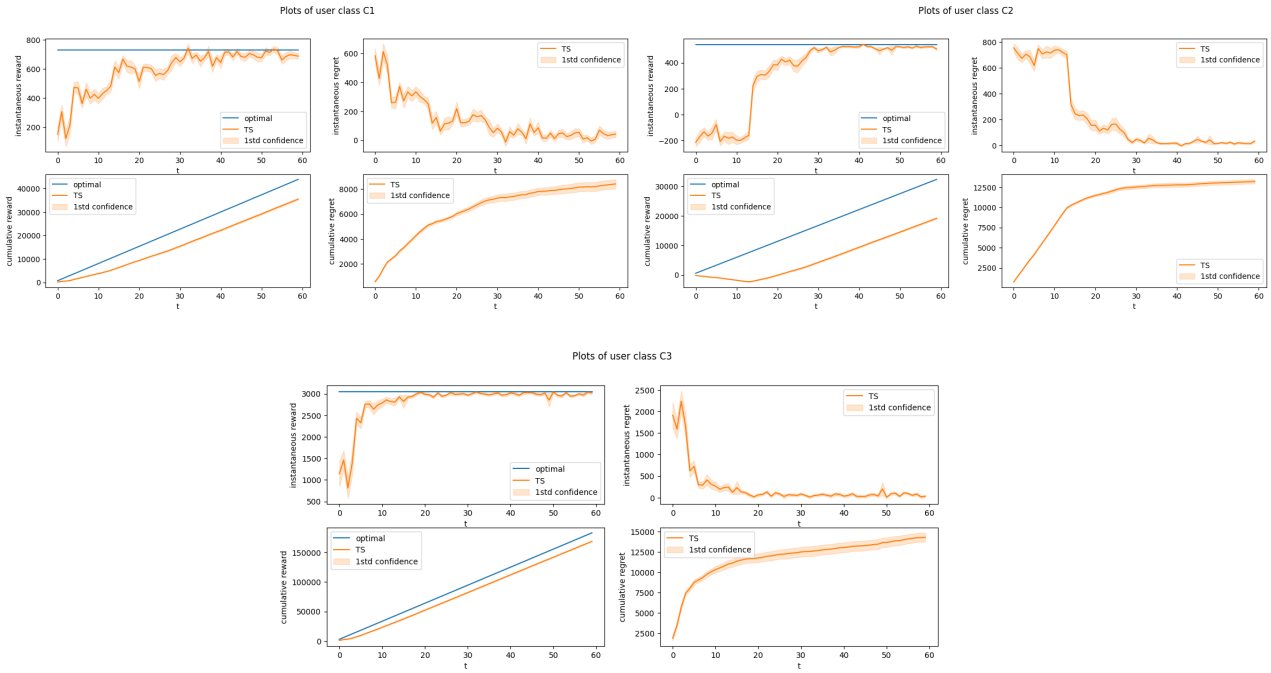


Figura 15: UCB vs TS in the case of unknown context structure and context generation algorithm

- Example of the reward and regret for each user class when using the context generation algorithm in TS:



5.3 Conclusions

- **Known context structure:** we can observe that the instantaneous reward is very close to the optimal one and in the TS this happens very fast;
- **Unknown context structure and no context generation algorithm** In our experiments, we observe that the optimal reward is never reached. This happens because the learner treats all the customer classes as if they were in the same context, i.e., as if they had identical pricing and advertising curves, which is false. Furthermore, we observe that, at a certain point, the instantaneous reward follows a horizontal asymptote. This behavior arises because, in this particular example, the learner learns the optimal arm of the customer class that yields the highest reward.
- **Unknown context structure and context generation algorithm** We observe that the context generation algorithm improves the learning process. This improvement arises from the learner's capacity to understand that by changing the context structure, it can get higher rewards.
  In the plot of the total instantaneous reward, we can notice a jump at t=14. This is due to the fact that, in our example, usually a new context structure is introduced. We can also notice a higher variability on days 28 and 42 for the same reason.
  In the different classes reward and regret plot, we notice that initially, the learner mostly learns the arms that maximize the reward of the class C3 since it has the highest optimal reward, even if this has a cost for the class C2, because the eventual trade-off is favorable. However, when the context structure changes twice (treating each customer class as in different contexts), it learns the optimal arm of the 3 customer classes. A similar behavior can be observed also in the UCB approach.

6 Chapter 5: Dealing with non-stationary environments with two abrupt changes

6.1 Non-stationary environment

In this step, we are considering the case in which all the users belong to class C1 and we are assuming that the curves related to the advertising part of the problem are known while the curves related to the pricing problem are not, with the addition that the curves related to the pricing part are non-stationary, being subject to seasonal phases. The phases are 3 and are spread over the time horizon.
These are the conversion functions for the non-stationary class C1, where every phase has the same length:

- Phase 1 starts in January and lasts until the end of March. We are assuming that in this period some university students buy the backpack because of the beginning of the second semester, thus the conversion rate is high for prices 50 and 60 and low for the other prices. In this phase the optimal price in terms of $conversion\_rate \cdot margin$ is 60.
- Phase 2 starts in April and lasts until the end of August. We are assuming that in this period students are less likely to buy a backpack for the university due to the summer break, thus conversion rates are generally low. In this phase the optimal price in terms of $conversion\_rate \cdot margin$ is 50.
- Phase 3 starts on September and lasts until the end of the year. This is the period in which university students are more likely to buy the backpack for the beginning of the new academic year, thus conversion rates are generally high for prices from 50 to 70 and are low for higher prices. In this phase the optimal price in term of $conversion\_rate \cdot margin$ is 70.

In this non stationary setting, given a pulled arm the reward is given by the following formula as in Step 1. The only difference is that for the conversion rate we need to consider the current phase:

$$B(number\_of\_daily\_clicks, conversion\_rate(pulled\_arm, current\_phase)) \cdot margin - cumulative\_daily\_costs$$

where $B(n, k)$ is the binomial distribution.

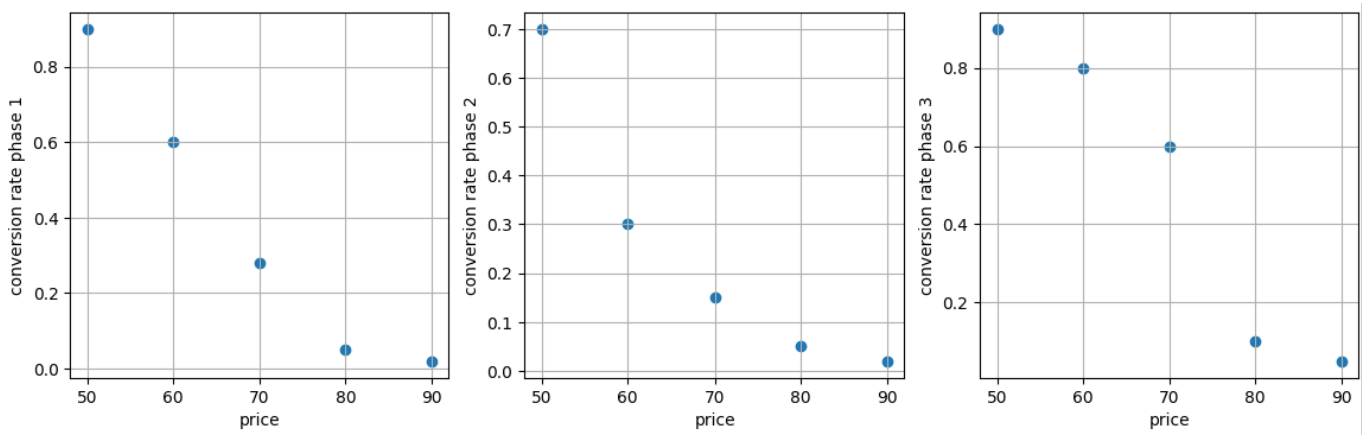


Figura 16: Conversion functions of class C1 for the 3 phases

6.2 UCB algorithm

First, we apply the basic UCB algorithm in this non stationary environment. As expected, since the optimal arm changes during the phases, the performance of UCB is very poor and the regret is linear.

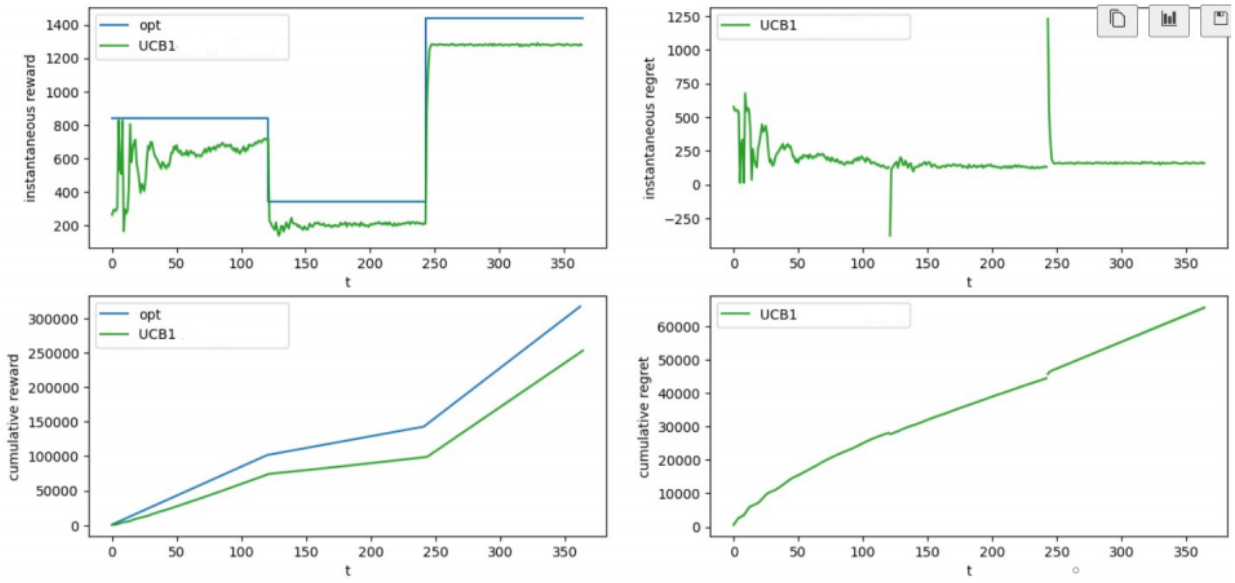


Figura 13: Basic UCB performance in non stationary environment

6.3 Sliding Window UCB algorithm

Given the results obtained above, we require an algorithm that is able to deal with non stationary environments. We initially implement a sliding window version of the UCB1 algorithm.
Despite vanilla Thomphson Sampling algorithm is proven to work well in its sliding window version (rely on "Sliding-Window Thompson Sampling for Non-Stationary Settings" by Francesco Trovò, Stefano Paladino, Marcello Restelli, Nicola Gatti, 2020), we saw that this doesn't hold for UCB1.
Indeed, since UCB1 relies on the upper confidences and since these tend to be very close one another, the introduction of a sliding window makes the learning very unstable. In particular, when the sliding window starts to discard samples, these ones probably belong to some non optimal arm that UCB1 pulled mostly in the first phase. Discarding these samples makes the upper confidences of these arms higher and since all the upper confidences tend to be very similar this is sufficient for the algorithm to start to push the non optimal arms.

To obtain better results we implemented a discounted version of the sliding window UCB algorithm. The only difference with the previous algorithm is that the number of times a given arm has been pulled is computed in the

following way:

$$N_t(\gamma, i) = \sum_{s=max(1,t-window\_size)}^{t} \gamma^{t-s} \cdot \mathbb{1}_{I_s=i}$$

where γ is the discount factor.
In this way when the window starts to discard samples, these are multiplied by a discount factor that, reducing their importance, does not push the algorithm to pull non optimal arms, since the upper confidence of better arms tends to be sufficiently higher, thanks to the discount factor. For all the details rely on the paper "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems" by Aurélien Garivier, Eric Moulines, 2008.
We compared the results of the two versions of Sliding Window UCB, applied to the non-stationary environment described above, in the following plots:

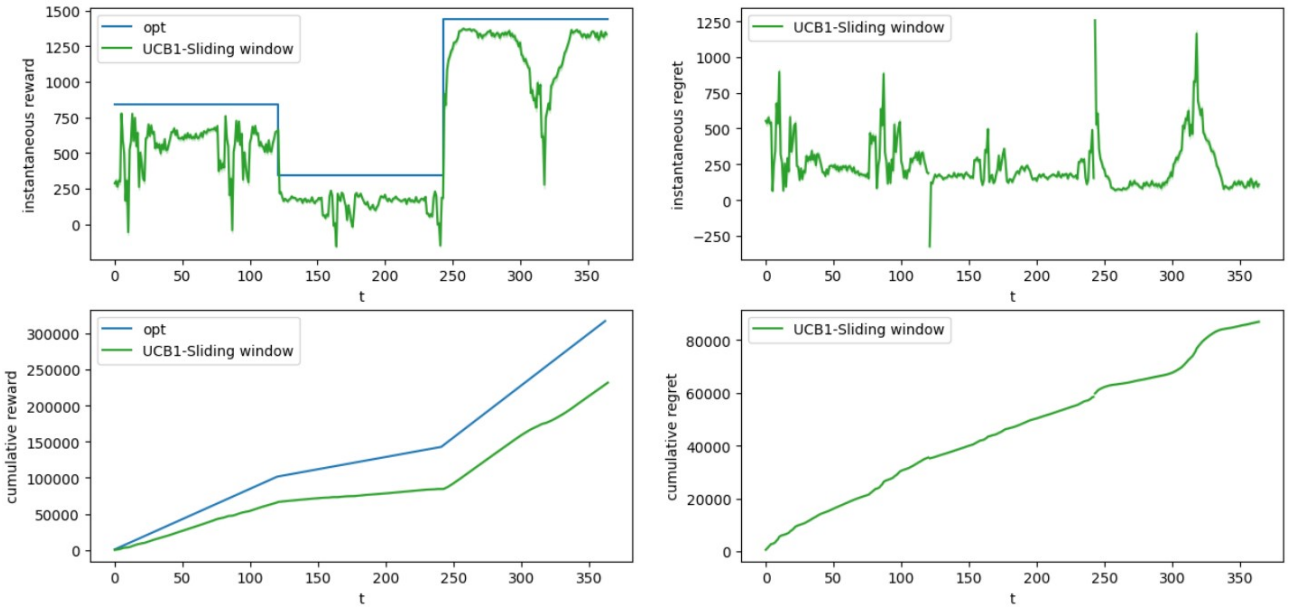


Figura 17: Sliding Window UCB performance in non stationary environment

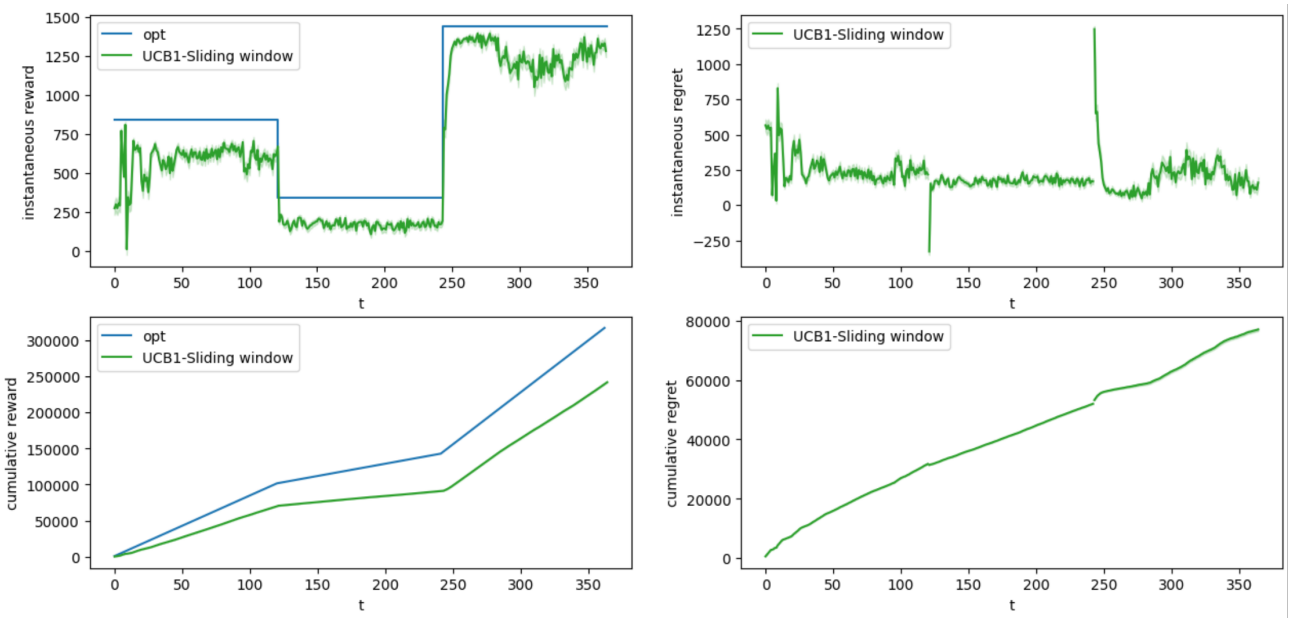


Figura 18: Sliding Window UCB with discount factor performance in non-stationary environment

6.4 Change Detection algorithm

A way to improve the performances is to switch from a passive non stationary method to an active one, that is able to actively deal with the samples when a change occurs.
The most used active algorithms are based on a change detection method and the one that we have implemented is CUSUM. For the details about the use of CUSUM paired with UCB1 algorithm rely on the following paper "A Change-Detection based Framework for Piecewise-stationary Multi-Armed Bandit Problem" by Fang Liu, Joohyun Lee and Ness Shroff, 2017.
The algorithm is a UCB1 that resets when a change is detected for the reward of one of the arms. In the original version only the arms that detect a change reset, but since a change occurs when the conversion rate function changes and since the reward function of every arm depends on the same conversion rate function, we can modify the simple algorithm to reset every arm when a change is detected.
The algorithm snippet is the following:

Algorithm 1 CD-UCB

Require: T, α and an algorithm CD$(\cdot,\cdot)$
  Initialize $\tau_i = 1, \forall i$.
  for t **from** 1 **to** T **do**
    Update according to equations (3-5).
    Play arm I_t and observe $X_t(I_t)$.
    if CD$(I_t, X_t(I_t)) == 1$ **then**
      $\tau_{I_t} = t + 1$; reset CD$(I_t, \cdot)$.
    end if
  end for

Algorithm 2 Two-sided CUSUM

Require: parameters ϵ, M, h and $\{y_k\}_{k\geq 1}$
  Initialize $g_0^+ = 0$ and $g_0^- = 0$.
  for each k **do**
    Calculate s_k^- and s_k^+ according to (6).
    Update g_k^+ and g_k^- according to (7).
    if $g_k^+ \geq h$ or $g_k^- \geq h$ **then**
      Return 1
    end if
  end for

Figura 19: CUSUM UCB1 algorithm pseudocode

In particular, CUSUM is two-sided since there are two random walks for the detection: $g+$ that detects positive changes and $g-$ that detects negative changes. For the tuning of the parameters we relied on this documents as a starting point, then we fined tuned plotting some chart to understand the magnitude of the drift of the random walks and to set the threshold to avoid false detections.
The obtained results are the following:

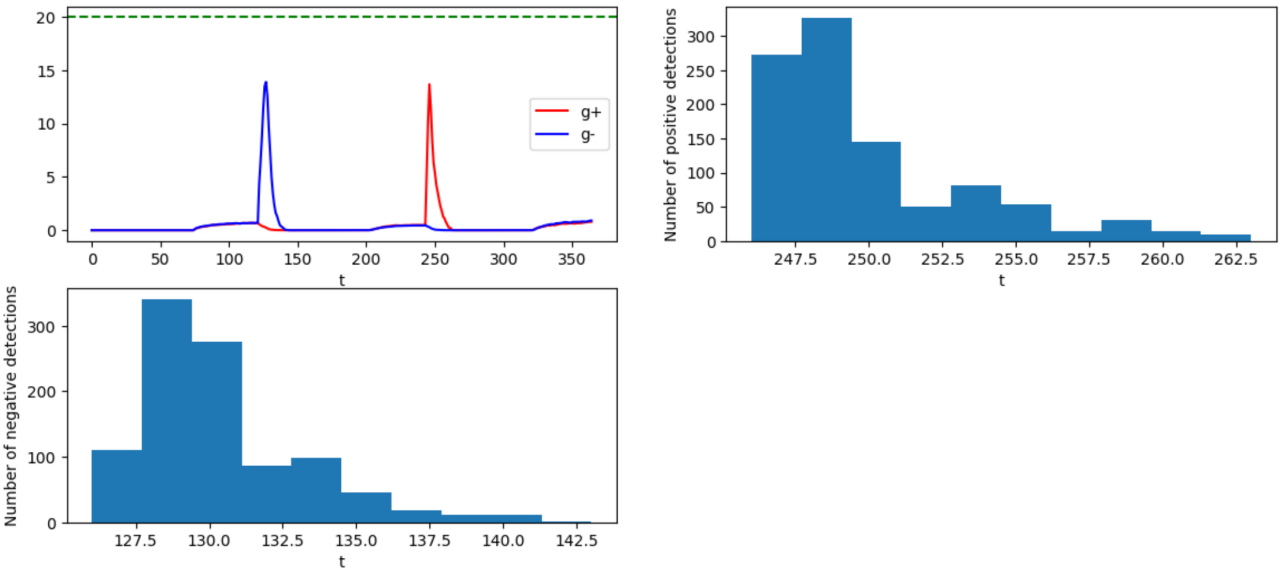


Figura 20: Plot of random walks, histogram of positive detections, histogram of negative detections

In our case the conversion rate function has a negative abrupt change at t = 122 and a positive abrupt change at 244. As we can see from the first plot, the negative random walk $g-$ drifts quite fast starting from t = 122, while the positive random walk $g+$ starts drifting fast just after t = 244. We can also see from the histograms that the detections are fast.

Finally, for 1000 experiments there are exactly 1000 positive detections and 1000 negative detections, thus the algorithm's detections are both fast and accurate. The reward and regret performances are the following:

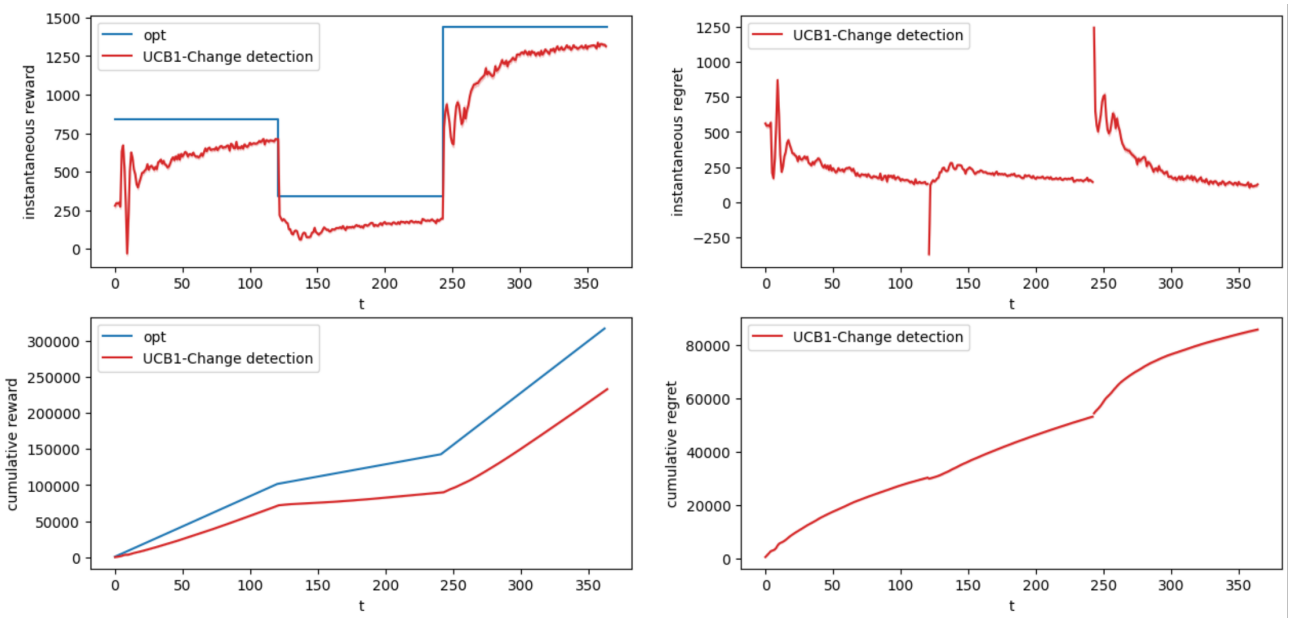


Figura 21: Performances of CUSUM UCB1 algorithm

7 Chapter 6: Dealing with non-stationary environments with many abrupt changes

7.1 Exp3 Algorithm

In this step we have implemented the Exp3 algorithm, where Exp3 stands for Exponential-weight algorithm for Exploration and Exploitation. This algorithm is primarily devoted to dealing with adversarial settings but it can also be used to deal with non-stationary settings when no information about the specific form of non-stationarity is known a priori.
The algorithm works as follows: it assigns weights to each possible arm and uses these weights to randomly decide which arm to pull next. At each step the weight of the pulled arm is either increased or decreased based on the reward of the arm itself.
The pseudocode for the algorithm is the following:

1. Given $\gamma \in [0, 1]$, initialize all weights $w_i = 1$ for $i = 1, ..., N$ where N is the number of arms
2. At each round t:
   (a) Set $p_i(t) = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{N} w_j(t)} + \frac{\gamma}{N}$ for each i
   (b) Draw the next action randomly according to the distribution of $p_i(t)$
   (c) Observe reward $x_{i_t}(t)$
   (d) Define the estimated reward $\hat{x}_{i_t}(t)$ as $\frac{x_{i_t}(t)}{p_{i_t}(t)}$
   (e) Set $w_{i_t}(t + 1) = w_{i_t}(t)e^{\gamma \hat{x}_{i_t}(t)/N}$
   (f) Set all other weights $w_j(t + 1) = w_j(t)$

7.2 Application of Exp3 to a non-stationary environment with two abrupt changes

In the first case we applied Exp3 algorithm to the setting of Chapter 5, thus considering the case in which all the users belong to class C1, the curves related to the advertising part of the problem are known while the curves related to the pricing problem are unknown and subject to the 3 seasonal phases described above.
On a time window of $T = 365$ the results we obtained are the following:

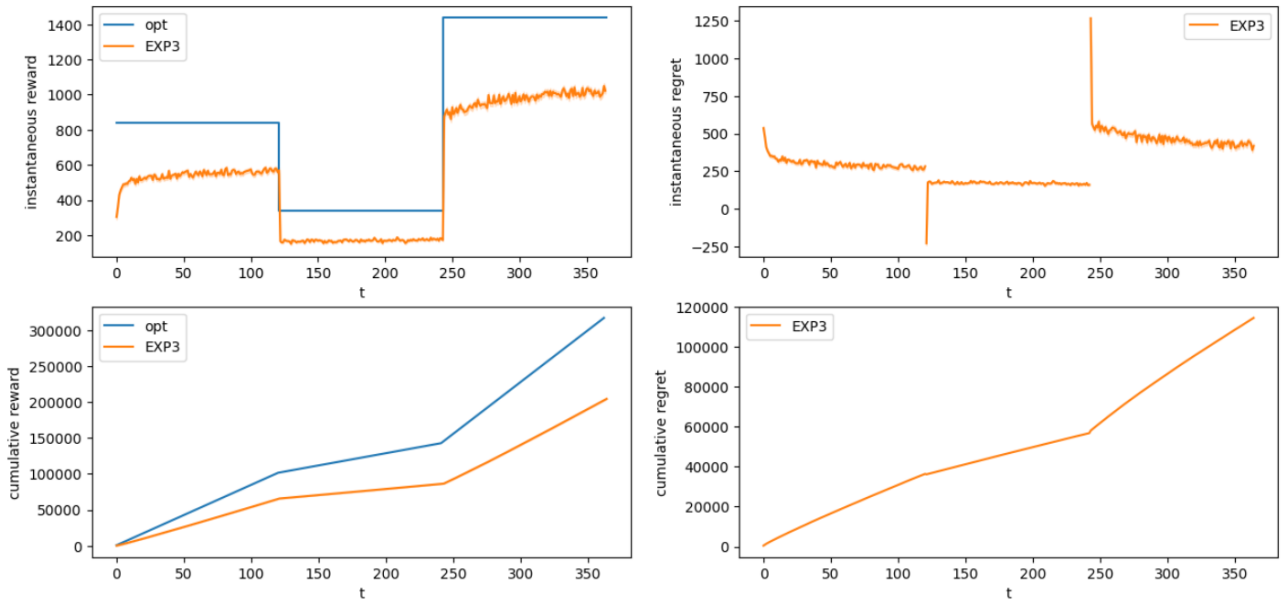


Figura 22: Performances of EXP3 algorithm

We compared these results to the ones obtained with the two non-stationary versions of UCB from Chapter 5. We can see, as expected, that Exp3 performs worse than the two algorithms, both in terms of cumulative reward and of cumulative regret.

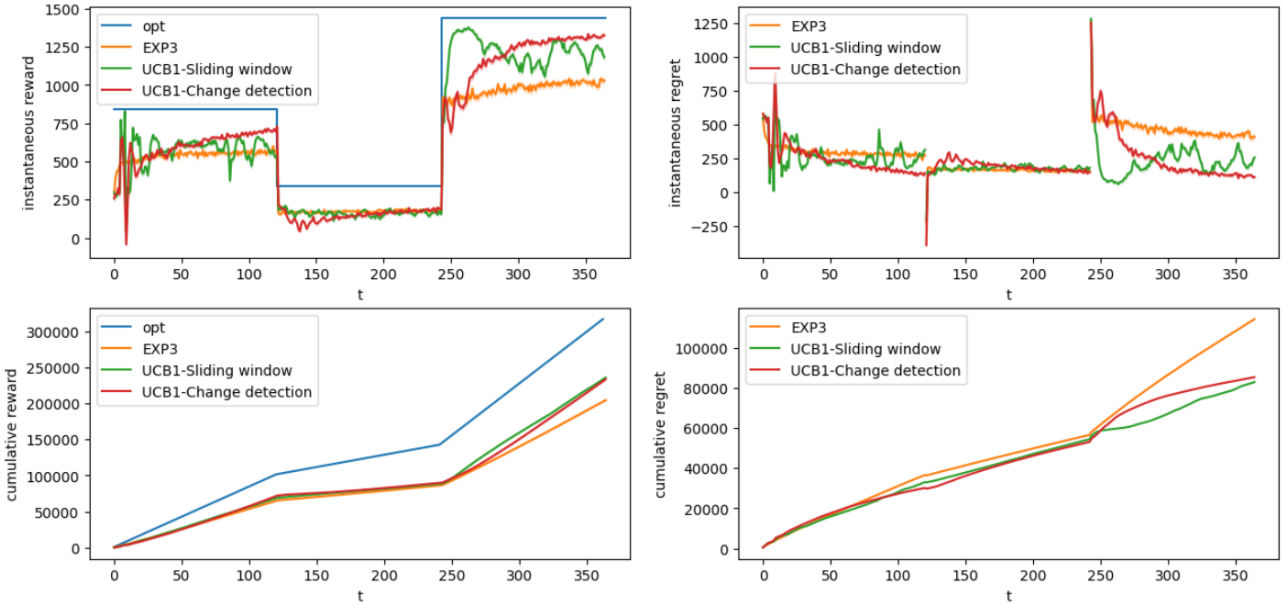


Figura 23: Performances of EXP3, UCB-Sliding Window and UCB-Change Detection

7.3 Application of Exp3 to a non-stationary environment with many abrupt changes

In the second case we considered a different non-stationary setting with a higher non-stationary degree, modelled by 5 phases which cyclically change with high-frequency.

We have considered the following 5 phases which last two days each and repeat themselves cyclically:

- Phase 1: The optimal price in terms of $conversion\_rate \cdot margin$ is 60
- Phase 2: The optimal price in terms of $conversion\_rate \cdot margin$ is 80
- Phase 3: The optimal price in terms of $conversion\_rate \cdot margin$ is 50
- Phase 4: The optimal price in terms of $conversion\_rate \cdot margin$ is 70
- Phase 5: The optimal price in terms of $conversion\_rate \cdot margin$ is 60

On a time window of $T = 365$ we have obtained the following results:

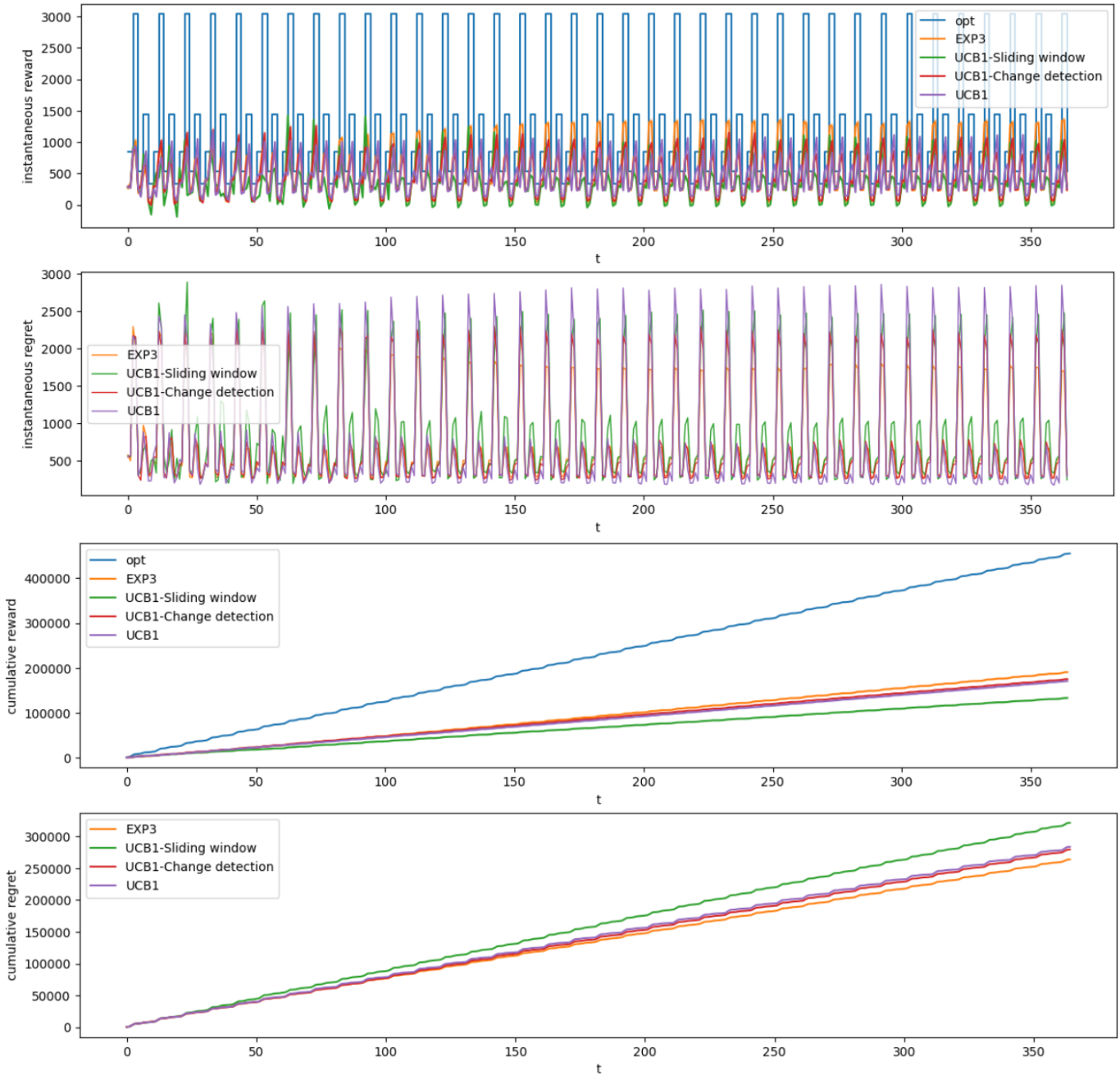


Figura 24: Performances of EXP3, UCB1, UCB-Sliding Window and UCB-Change Detection

As expected, in this highly non-stationary setting, both in terms of cumulative reward and cumulative regret the Exp3 algorithm performs better than the basic UCB1 algorithm and the two non-stationary versions of UCB1 of Chapter 5.