POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Online pricing and advertising for an E-commerce

Online Learning Applications
Computer Science and Engineering

Authors:

Fabrizio Carlo	**10747892**
Vargu Kevin	**10937465**
Targhini Enrico	**10632193**

Academic Year: 2023-24

Contents

1 | Step 0

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

1.1. Scenario Description

General description of the scenario

Considering an E-commerce of an high quality coffee blend, in particular the E-commerce sells online boxes of 1 kg of ground coffee. Coffee is a product that is requested by a wide range of customers, so, its E-commerce covers a great number of classes. The E-commerce can decide among 5 different selling price of the product [10$, 15$, 20$, 25$, 30$] and can take advantage of an advertising strategy playing on 100 different bids in a range from 100$ to 397$ (100$, 103$, 106$ ...). We consider our high-quality coffee blend to be a 'luxury' product, meaning it's relatively expensive and attracts only those customers who are genuinely interested. We hypothesize a correlation between a customer's stress level and their preference for lower-quality coffee – as stress increases, the desire for caffeine rises, and the willingness to pay a premium for high-quality coffee decreases. The pool of customers has been divided according to the latter motivation. Therefore, we assume that the most interested demographic in our high-quality coffee blend is the third category: individuals over 30 years old living in rural areas. Thanks to the advertising platform, the application is able to observe two binary features: F1 "Age range" and F2 "Geographic area".

$$F1 = \begin{cases} 1, & \text{if Age} \leq 30 \\ 0, & \text{otherwise} \end{cases}$$

$$F2 = \begin{cases} 1, & \text{if Geographic Area} = \text{Urban} \\ 0, & \text{if Geographic Area} = \text{Rural} \end{cases}$$

Using the above features we can identify 4 different classes of customers, but the analysis has been focused just on 3 classes excluding class $<$ F1=1,F2=0 $>$, customers with age less than 30 and living in a rural area, with the idea that the impact of that class on the sales of the coffee bland is much lower with respect to other classes.

Mathematical definitions

- $i \in \{1, 2, 3\}$ = class index
- P=$\{10, 15, 20, 25, 30\}$
- $p \in$ P = price
- $B = \{100, 103, ..., 397\}$
- $b \in B$ = bid
- $\mu_i \in [0, 1]^5$ = conversion probabilities of class i
- $\mu_{i,p}$ = conversion rate of class i at price p
- $\text{con}_i(p) \sim Bernoulli(\mu_{i,p})$ = conversion sample for a customer of class i at price p
- $n_i(b)$ = number of daily click of class i given the bid b
- $CON_i(p, n_i(b))$ = Total number of conversions of class i at price p out of $n_i(b)$ number of clicks
- $c_i(b)$ = cumulative daily click cost of class i given the bid b
- T = Time horizon (365 days)
- $t \in \{1, 2, 3, ..., 364\}$ = round indicator
- $\text{REV}_i(T)$ = Cumulative reward of class i over time horizon T
- $\text{rev}_i(t)$ = instantaneous reward of class i at round t
- REV(T)= Cumulative reward over time horizon T
- p_i^t price played at round t for class i
- b_i^t bid played at round t
- p_i^* optimal price for class i
- $b_i^*(p)$ optimal bid for class i given price p

Each of the above quantity, if it is indicated with t, means that is generated or sampled in round t. For simplicity, in the STEPS with only one class considered, the index $i - th$ has been omitted.

Generator functions for the simulation

Each class is characterized by a unique set of conversion probabilities, a distinct function that calculates the number of daily clicks based on the bid, and a separate function that describes the cumulative daily cost of clicks based on the bid.

Generation of number of daily clicks given the bid

To model the number of daily clicks given the bid of a class we have used the same "base"function:

$$f(x) = -(\frac{1}{a}x - b)^2 + c \quad \textit{where } x \textit{ is the bid}$$

Each class is characterized by a different choice of the parameters (a, b, c). The above function is strictly concave and going to $-\infty$ as the bid goes to ∞. The parameter a controls the width of the bell (as a grows the width reduce); the parameter b controls where is the position of the peak and the parameter c controls the height of the peak. The samples of the number of clicks have been done sampling values from:

$$n_i(b) = \begin{cases} y, & \text{if } \; y \sim N(\;\; f(b) \;\; , \;\; 2500 \;\;) > 0 \\ 0, & \text{otherwise} \end{cases}$$

Where $N(\mu, \sigma^2)$ represent a Normal distribution with mean equal to μ and variance equal to σ^2. Note that we have considered only samples that are greater or equal 0 (the number of daily clicks can not be less than 0). The choice of this specific concave function, that results in no daily clicks if the bid is too high, is motivated from the fact that if the bid is very high and the number of auctions in a single day is large, then the entire daily budget would expire in a single auction resulting in low daily clicks. Even if there is no specific constraint on the daily budget, the latter scenario has been considered as a better representation of the reality.

Generation of the conversions of a class given the price

Having $n_i(b)$ people of class i clicking on the ad of the coffee bland product shown on the website at price p , the total number of conversions is:

$$\text{CON}_i(p, n_i(b)) = \sum_1^n con_i(p) \sim Binomial(n, \mu_{i,p})$$

Generation of the cumulative daily click cost

An analogous construction for the samples of the cumulative daily cost has been done. The cumulative daily click cost given the bid is modeled using the following function:

$$h(x) = d\, log(x)$$

where d is the class parameter, defining a specific function for every class.

And the samples have been done as follows:

$$c_i(b) = \begin{cases} z, & \text{if } \ z \sim N(\ h(b)\ ,\ 625\ ) > 0 \\ 0, & \text{otherwise} \end{cases}$$

The latter function is instead strictly, that is reasonable as the cumulative daily cost grows as the bid grows even if no click have been produced.

Reward considerations

Considering a time horizon T and a time round indicator t, the objective of the project is to maximize the cumulative reward defined as follow:

$$REW(T) = \sum_{i=1}^{3} \ (\ \sum_{1}^{T} \mathrm{CON}_i(p^t, n_i(b^t)^t) \quad (p^t - C) - c_i(b^t))$$

Where C is the production cost.

That is basically the sum of the cumulative cumulative reward of all the classes. The cumulative reward of a class is the sum of the instantaneous (round) reward that is simply the difference between the number of conversions at time t multiplied by the margin minus, the cumulative daily cost at round t.

In our analysis we have not considered the constant C that for simplicity, since the results does not depend on the choice of C that is always less than the price and constant.

Classes description

Class 1 < F1=1, F2=1 >
Class 1 consists of customers aged 30 or younger, residing in urban areas. This group is generally more stressed and less inclined to pay a high price for a high-quality coffee blend. However, it is also the most sociable and accessible for advertising campaigns.

The following are the class 1 parameters:

- Conversion probabilities: $\{10\$: 0.4\ ,\ 15\$: 0.6\ ,\ 20\$: 0.3\ ,\ 25\$: 0.2\ ,\ 30\$: 0.1\}$

- Number of daily clicks parameters ($a = 3\ \ ,\ \ b = 85\ \ ,\ \ c = 10000$)

- Cumulative daily cost parameter : $d = 700$

The conversion probabilities of that class are shown in fig. 1.1 :

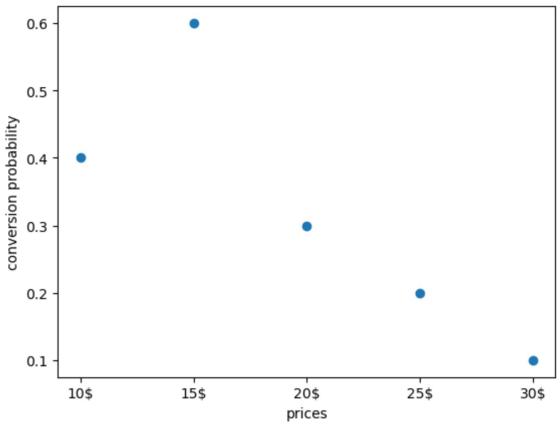


Figure 1.1: Conversion probabilities given the price of class 1

As shown above the highest conversion probability is the one associated to 15$, the second lowest price.

The number of daily clicks given the bid of that class is shown in fig. 1.2 and the cumulative daily click cost given the bid for the class 1 is shown in fig 1.3.

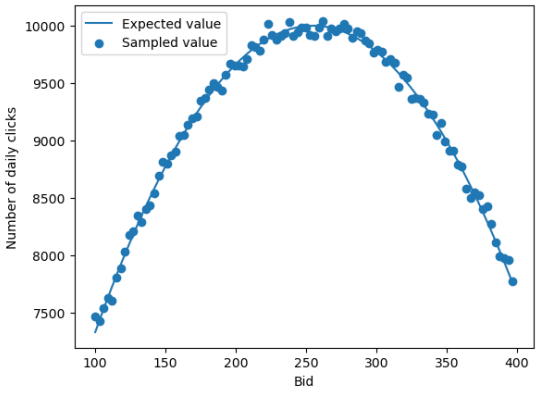


Figure 1.2: Number of daily clicks of class 1

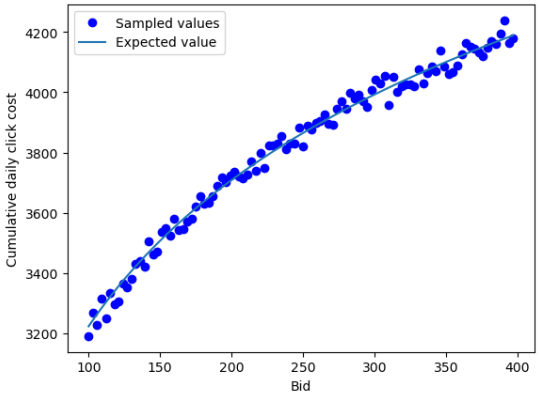


Figure 1.3: Cumulative daily click cost class 1

The point are an example of the samples of the number of daily clicks and the cumulative daily cost, in order to render the idea of the variability.

Class 2 < F1=0, F2=1 >

Class 2 comprises customers aged over 30, residing in urban areas. This group is moderately stressed and somewhat willing to pay a high price for a high-quality coffee blend. However, it is less socially active compared to the first class and, therefore, less reachable by advertising campaigns. Here are the parameters for Class 2:

- Conversion probabilities: $\{10\$: 0.1 \ , \ 15\$: 0.2 \ , \ 20\$: 0.4 \ , \ 25\$: 0.7 \ , \ 30\$: 0.4\}$

- Number of daily clicks parameters $(a = 3 \ , \ b = 95 \ , \ c = 9000)$

- Cumulative daily cost parameter : $d = 500$

The conversion probabilities of that class are shown in fig. 1.4 :

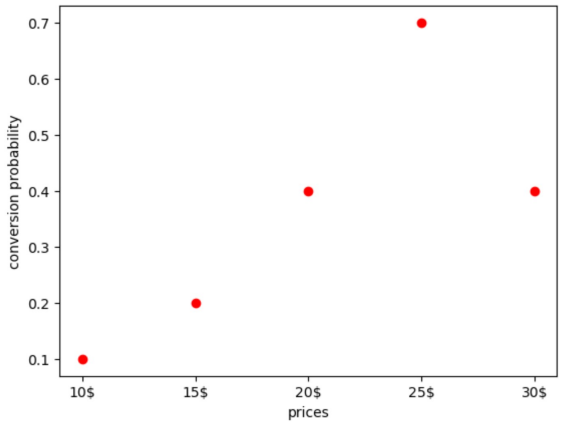


Figure 1.4: Conversion probabilities given the price of class 2

As shown above the highest conversion probability is the one associated to 25$, the second highest price. The number of daily clicks given the bid of that class is shown in fig. 1.5 and the cumulative daily click cost given the bid for the class 2 is shown in fig 1.6.

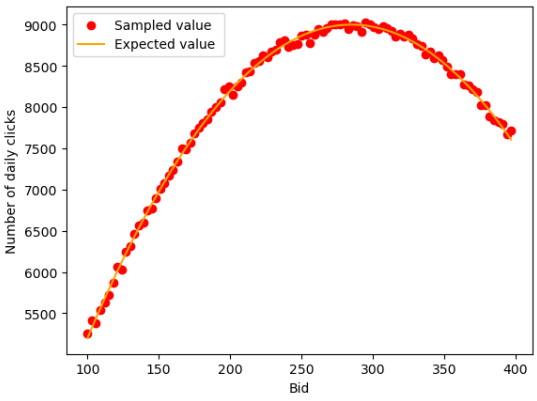


Figure 1.5: Number of daily clicks class 2

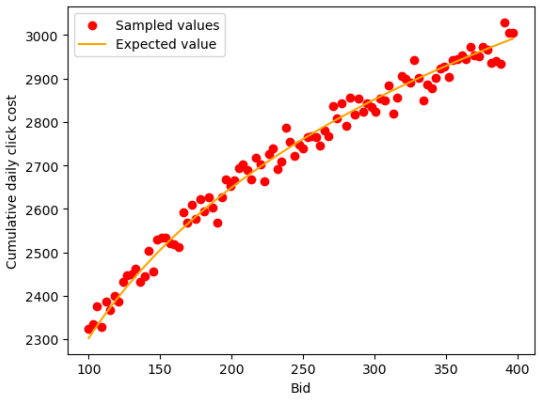


Figure 1.6: Cumulative daily click cost class 2

The point are an example of the samples of the number of daily clicks.

Class 3 < F1=0, F2=0 >
Class 3 consists of customers aged over 30, residing in rural areas. This group is the least stressed and highly willing to pay for a high-quality coffee blend. However, it is less socially active compared to the other classes and, consequently, less accessible for advertising campaigns.

The following are the class 3 parameters:

- Conversion probabilities: $\{10\$: 0.1 \ , \ 15\$: 0.1 \ , \ 20\$: 0.3 \ , \ 25\$: 0.5 \ , \ 30\$: 0.6\}$

- Number of daily clicks parameters $(a = 3 \ , \ b = 100 \ , \ c = 8000)$

- Cumulative daily cost parameter : $d = 400$

The conversion probabilities of that class are shown in fig. 1.7 :

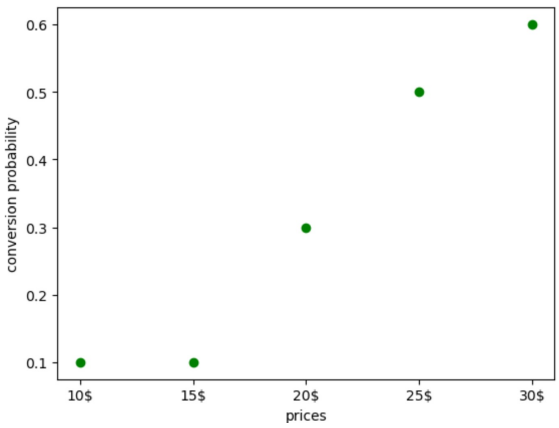


Figure 1.7: Conversion probabilities given the price of class 3

As shown above the highest conversion probability is the one associated to 30\$, the second highest price. The number of daily clicks given the bid of that class is shown in fig 1.8 and the cumulative daily click cost is shown in fig 1.6.

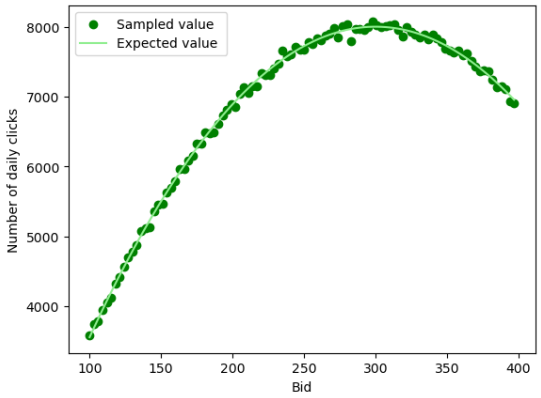


Figure 1.8: Number of daily clicks class 3

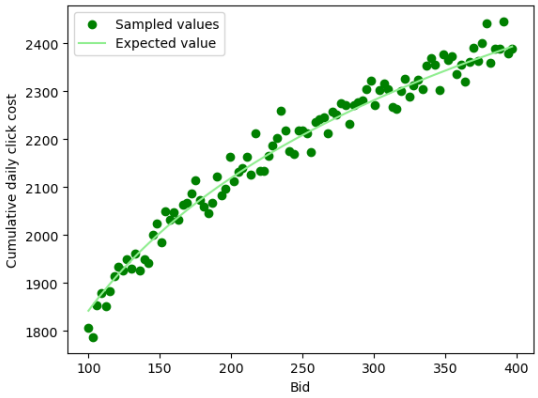


Figure 1.9: Cumulative daily cost class 3

Classes comparison

To highlight the differences in the classes, below the comparison plots are shown below :

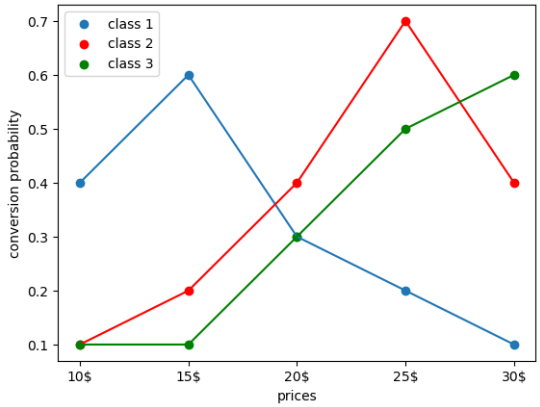


Figure 1.10: Comparison between conversion probabilities

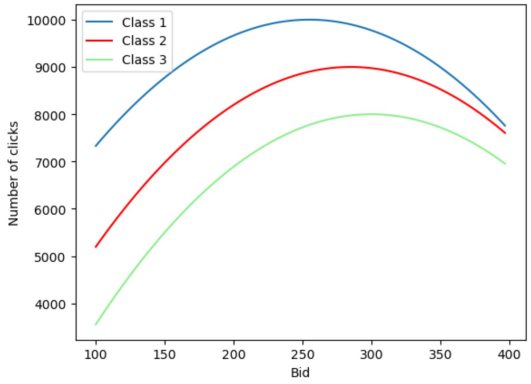


Figure 1.11: Comparison between the number of daily clicks

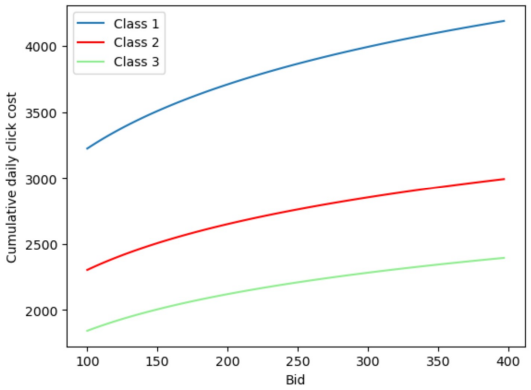


Figure 1.12: Comparison between the cumulative daily cost

2 | Step 1

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.

2.1. Design of the Algorithms

Clairvoyant algorithm

The General Clairvoyant optimization for the scenario has been done as follow:

Algorithm 1 Clairvoyant

for $t = 1, 2, \dots T$ **do**
  for $i = 1, 2, 3$ **do**
    $p_i^* \leftarrow$ Pricing Clairvoyant(μ_i)
    $b_i^*(p_i^*) \leftarrow$ Advertising Clairvoyant(p_i^*)
    play the joint arm $(p_i^* \ , \ b_i^*(p_i^*))$
  end for
end for

The subroutine Pricing Clairvoyant works as follow:

Algorithm 2 Pricing Clairvoyant

1: input : μ
2: **for** $t = 1, 2, \ldots T$ **do**
3: Play $p^* \leftarrow \underset{p}{\text{argmax}} \; p\mu_p$
4: **end for**

while the subroutine Advertising Clairvoyant works as follow:

Algorithm 3 Advertising Clairvoyant

1: input: p
2: **for** $t = 1, 2, \ldots T$ **do**
3: Play $b^*(p) \leftarrow \underset{b}{\text{argmax}} \; (\mu_p)(p)n(b) - c(b)$
4: **end for**

Basically, the optimization has been done in two subsequent steps: firstly compute the optimal price for every class and then, knowing the optimal price, compute the optimal bid for every class.

Learning Procedure

UCB1 Algorithm

UCB1 algorithm in general works with upper confidence bounds, so let UP_p^t be the upper confidence bound associated with arm p a round t. The UCB1 Algorithm works as follow:

Algorithm 4 UCB1

1: **for** $t = 1, 2, \dots T$ **do**
2: Play $p^t \leftarrow \underset{p}{\text{argmax}} \ (\widehat{E[con_p]}^t + UP_p^t)p$
3: Collect $rew^t_{p^t}$ and observe $CON(p^t, n(b^t))^t$
4: Update $\widehat{E[con_{p^t}]}^t$ and update for each arm p ,UP_p^t
5: **end for**

Where $\widehat{E[con_{p^t}]}^t$ is the empirical mean up to time t of the pricing reward obtained playing arm p=p^t. It estimate updates as follow:
Being: $rew^t = \frac{CON(p^t, n(b^t)^t)^t}{n(b^t)^t}$

$$\widehat{E[con_{p^t}]}^{t+1} = \frac{\widehat{E[con_{p^t}]}^t (t-1) + rew^t}{t}$$

That is the total number of conversions thanks to price p up to time t, over the total number of clicks happened when price p was played.

While UP_p^t is defined as follow:

$$UP_p^t \ \propto \ \sqrt{\frac{2log(t)}{num_p^{t-1}}}$$

where num_p^{t-1} is the number of times arm p has been played up to time $t-1$. Specifically, it has been used:

$$UP_p^t \ = 5 \ \sqrt{\frac{2log(t)}{num_p^{t-1}}}$$

a proportion factor equal to the number of elements in the support of P.

TS Algorithm

TS Algorithm is a Sampling based Bandit Algorithm, in the considered scenario the Bernoulli TS has been used. It has been assumed the following Beta-Bernoulli Bayesian model over the conversion of a single customer of class i at price p $con_i(p)$:

$$con_i(p) \mid \mu_{i,p} \sim Bernoulli(\mu_{i,p})$$

$$\mu_{i,p} \sim Beta(\alpha_{i,p}, \beta_{i,p})$$

therefore, considering $n(b)$ the total number of conversions of a single day as independent realizations of the Bernoulli described above, the model becomes:

$$CON(p, n(b)) \mid \mu_p \sim Binomial(\mu_p, n(b))$$

$$\mu_p \sim Beta(\alpha_p, \beta_p)$$

The Algorithm works as follow:

Algorithm 5 TS

for $t = 1, 2, \dots T$ **do**
  $\forall p$ sample $\widehat{\mu_p}^t \sim Beta(\alpha_p^t, \beta_p^t)$
  Play $p^t \leftarrow \underset{p}{\text{argmax}}\ p\ \widehat{\mu_p}^t$
  Receive $rew_{p^t}^t$ and observe CON^t and $!CON^t$
  Update $(\alpha_{p^t}^{t+1}, \beta_{p^t}^{t+1}) \leftarrow (\alpha_{p^t}^t + CON^t\ ,\ \beta_{p^t}^t + !CON^t)$
end for

Both α^0 and β^0 have been initialized as 1, recalling that a $Beta(1,1)$ distribution is equivalent in probability to a $U[0,1]$ distribution.

CON^t is the number of conversions at round t and $!CON^t$ is the number of non-conversions at time t.

2.2. Theoretical guarantees

UCB1 Algorithm

The theoretical guarantees for the UCB1 Algorithm states an upper bound to the cumu-

lative expected regret of the form:

$$R_T(UCB1) \leq \sum_{a:\mu_a<\mu_a^*} (\frac{4log(T)}{\triangle_a} + 8\triangle_a)$$

Where $\triangle_a = \mu_a^* - \mu_a$.

So, the cumulative expected regret should be logarithmic in function of T.

TS Algorithm

The theoretical guarantees for the TS Algorithm fixes an upper bound to the cumulative expected regret of the form:

$$R_T(TS) \leq O(\sum_{a:\mu_a<\mu_a^*} (\frac{\triangle_a(log(T) + log(log(T)))}{KL(\mu_a^*, \mu_a)})$$

Where (μ_a^*, μ_a) is the Kullback-Leibler dissimilarity measure between the Bernoulli distributions $Be(\mu_a)$ and $Be(\mu_a^*)$.

Therefore, the cumulative expected regret should be logarithmic in function of T as the case of the UCB1 Algorithm. It is expected the TS Algorithm to perform better than the UCB1, in the considered "simple" scenario of step 1, thanks to a lower degree of exploration and an higher degree of exploitation.

2.3. Results Obtained

As it emerges from the figures below, the theoretical guarantees are respected.

Here the results are reported:

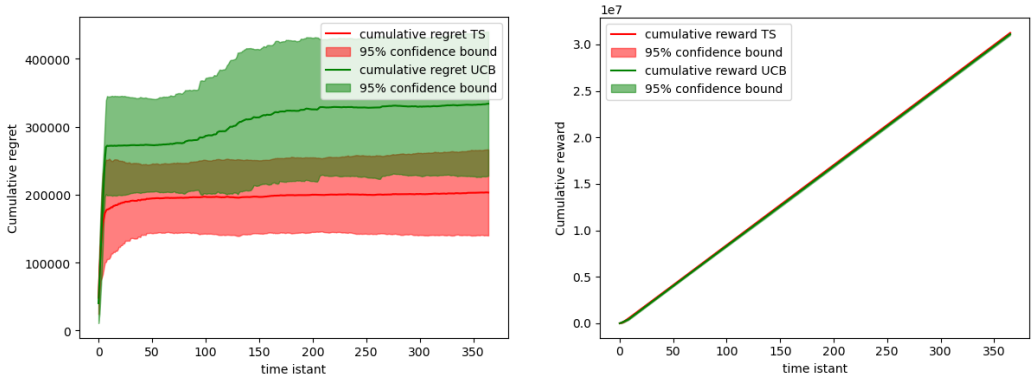


Figure 2.1: Cumulative quantities

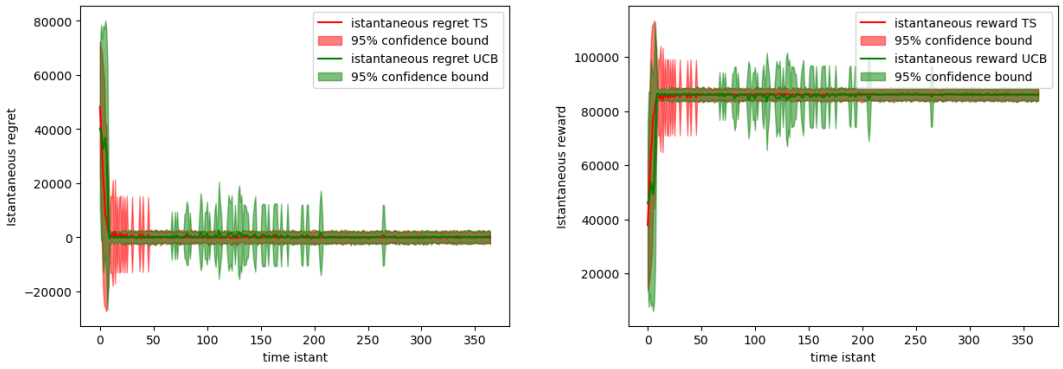


Figure 2.2: Instantaneous quantities

As we can see from the figure 2.2 of the cumulative regret, the UCB1 algorithm presents a "weird" behaviour around time instant 200, probably because a non-optimal arm has not been sufficiently explored yet and, therefore, its upper bound gets temporarily higher than the one of the optimal price. Nevertheless, in few rounds it stabilizes again over the optimum.

The estimation of the parameters by UCB1 and TS are shown in fig 2.3. On the left, the estimation of the conversion probabilities, done by the UCB1, associated to each arm (points in blu) and their upper confidence bound (in red). On the right, the estimation of the conversion probabilities, done by the TS, and their 95% confidence interval. The optimal clairvoyant price was 15, and we can see that the confidence bounds both of the TS and the UCB1 for price 15 are pretty accurate, that means that price 15 has been exploited a lot.

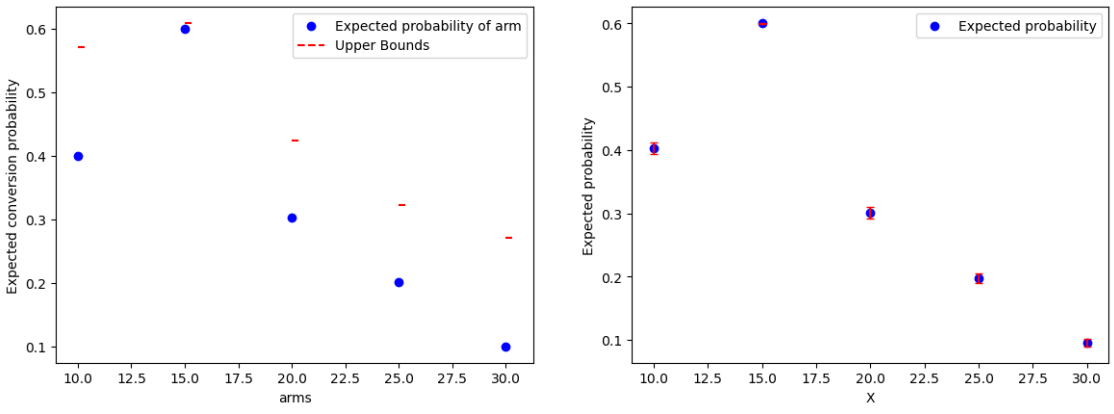


Figure 2.3: estimations of the conversion probabilities by the two algorithms

3 | Step 2

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.

3.1. Description of the environment

In this scenario we are still considering only the first class. The curve related to the pricing problem is known beforehand, while the curves related to the advertising problems are not. In this step, we apply the GP-UCB and GP-TS algorithms in order to model the two advertising curves, reporting the plots of the average value and standard deviation of the cumulative regret, cumulative reward, instantaneous regret, and instantaneous reward.

3.2. Learning Algorithm

Since the pricing curve is known, the clairvoyant algorithm is used to pull, at each iteration, always the best arm for the pricing part. For the advertising problem, the algorithm has to estimate the curves. To do so, two different algorithm are created, and then compared to watch which one provides better performances.

3.2.1. GP

Gaussian processes are useful tools to estimate functions from just few observations. The idea behind them is that, the information obtained by observing a value $f(x)$ associated to x, can be used to infer information for the neighbors of x. Therefore, they are powerful objects that are very useful in the case of MAB problems with a large number of arms. In the latter case of study, since the regret of a classic MAB in a stationary environment, is linear in function of the number of arms, applying a classic MAB algorithm is inefficient.

The combination of GP and MAB allows to reach a sub-linear regret.

In our analysis, the support of the two functions to estimate $(f(b), h(b))$ is pretty large $(B = 100, 103, 106, ..., 397)$, and the values of it are far one from the other (distance=3 between every two consecutive bids). Therefore, a great sensitivity analysis of the parameters of the GP has been done. In particular we had better results considering four different GPs, one to estimate $f(b)$ for the UCB1, one to estimate $f(b)$ for the TS, one to estimate $h(b)$ for the UCB1 and the last to estimate $h(b)$ for the TS.

All the GPs have been calibrated in order to deal with the lengthscale of the data. More precisely, the hyper-parameters of the kernel have been tuned for the latter objective,

3.2.2. GP-TS

This version of the algorithm mix up the Thompson Sampling with a Gaussian Process procedure. Basically, for each arm, the algorithm draws a value from a normal distribution with estimated mean and variance, and the arm associated with the highest value is pulled. When the reward is observed, the model is updated by performing a train using a Gaussian Process, where the variables are the arms and the targets are the corresponding obtained rewards.

Let:

- α: represent the standard deviation used by the Gaussian Process. For simplicity, and to have accurate results, we decided to use $\alpha = 100$.
- σ be an estimation of the variance of each arm. The algorithm starts with $\sigma = 100$ for each arm

The pseudo-code is defined as follows.

Algorithm 6 Gaussian Process TS

for $t = 1, 2, \dots T$ **do**
  $\forall p$ sample $\widehat{\mu_p}^t \sim Normal(mean_p^t, \sigma_p^t)$
  Play $p^t \leftarrow \underset{p}{\text{argmax}} \ \widehat{\mu_p}^t$
  Receive $rew_{p^t}^t$
  Append $(p^t, rew_{p^t}^t)$ to the gaussian process datas
  Train the model
  Update means and variances of the arms with their estimation using the gp
end for

Where:

- $rew^t_{p^t}$ is the observed reward at round t for arm p. In our case: $rew^t_{p^t}$ = conversion_probability * price - cumulative_cost

3.2.3. GP-UCB1

This version of the algorithm works similarly to the GP-TS version, but this time uses the UCB1 approach to pull arms. In fact, at each iteration, it computes for each arm the sum between the estimated value and the upper bound of a confidence value, and it plays the arm associated with the highest value. The algorithm then collect the reward and update the mean and confidences similarly to GP-TS.

The alpha and sigma parameter works as previously, the only new parameter used is the confidence, initialized at infinite for each arm. This is the pseudo-code of the algorithm:

Algorithm 7 Gaussian Process UCB1

1: **for** $t = 1, 2, \dots T$ **do**
2: $\forall p$ compute $\widehat{\mu_p}^t = mean_p + UP_p^t$
3: Play $p^t \leftarrow \underset{p}{\text{argmax}} \ \widehat{\mu_p}^t$
4: Receive $rew_{p^t}^t$
5: Append $(p^t, rew_{p^t}^t)$ to the gaussian process datas
6: Train the model
7: Update means and variances of the arms with their estimation using the gp
8: $\forall p$ Update $confidence_p = \sigma_p * 10$
9: **end for**

Where:

- UP_p^t is the upper bound of the confidence of arm p at round t
- $rew_{p^t}^t$ is the observed reward at round t for arm p. In our case:

  $rew_{p^t}^t$ = conversion_probability * price - cumulative_cost

As you can see, the algorithm is very similar to the GP-TS one, and the confidence update is similar to the UCB1 one.

3.2.4. GP-Handler

The goal of the second step is to estimate both the cumulative cost curve and the number of clicks given the bid curve. In this case, different GPs must be created for each curve. The problem that arise in this case is that different GPs could pull different arms and the same round and, in addition, the optimal arm for the click GP can differ from the optimal arm of the cost GP. In order to work around this problem, the algorithm creates an handler structure. The handler creates both the GP (one per curve), update them both using the respective rewards and pull the best joint arm at each iteration. This is the pseudo-code of the pull arm method:

Algorithm 8 GP-Handler TS

1: **for** arms in advertising arms **do**
2: Compute $n\_clicks_p \sim Normal(mean_p^t, \sigma_p^t)$ from GP_clicks
3: Compute $cum\_cost_p \sim Normal(mean_p^t, \sigma_p^t)$ from GP_cost
4: Estimate $reward_p = n\_clicks_p * price * prob - cum\_cost_p$
5: Play $p^t \leftarrow argmax(reward_p)$
6: **end for**

Algorithm 9 GP-Handler UCB1

1: **for** arms in advertising arms **do**
2: Compute $n\_clicks_p = mean_p + UP_p^t$ from GP_clicks
3: Compute $cum\_cost_p = mean_p + UP_p^t$ from GP_cost
4: Estimate $reward_p = n\_clicks_p * price * prob - cum\_cost_p$
5: Play $p^t \leftarrow argmax(reward_p)$
6: **end for**

The handler so pull the best arm basing on an estimation over the best reward it could obtain by playing that arm, and update both GP_clicks and GP_cost using p. For the price, the handler uses the price associated to the best arm pulled by the clairvoyant algorithm.

3.3. Theoretical guarantees

For the GP-TS algorithm, the theoretical guarantees are that the regret of the algorithm can be bounded with probability 1-δ. The upper bound of the regret of GP-TS is:

$$R_T(GP-TS) \leq \sqrt{\frac{8}{\log(1+\frac{1}{\sigma^2})} * TB\gamma_T} \tag{3.1}$$

Where:

- σ^2 is the square of the maximum variance
- T is the time we compute the regret
- γ_T is the information gained at time T
- $B = 8 * \log(\frac{T^4 * M}{6*\sigma})$
- M is the number of arms

For the GP-UCB, we performed some researches over some theoretical guarantees. We discovered from: "Proceedings of Machine Learning Research" by [Paria, K. Kandasamy, and B. Póczos] that the regret is:

$$R_T(GP-UCB1) = O(\sqrt{T * \beta_T * \gamma_T}) \tag{3.2}$$

Where:

- It implies sub-linearity because $\gamma_T = \text{o}(\frac{T}{\log(T)})$
- $\beta \sim \log(t)$

In the end, both the upper bounds suggest that the regret must be sub-linear.

3.4. Results

The results obtained are in line with the theoretical guarantees provided. Here are the graphs of the algorithms' performances after 20 runs and a time period of one year (T=365).

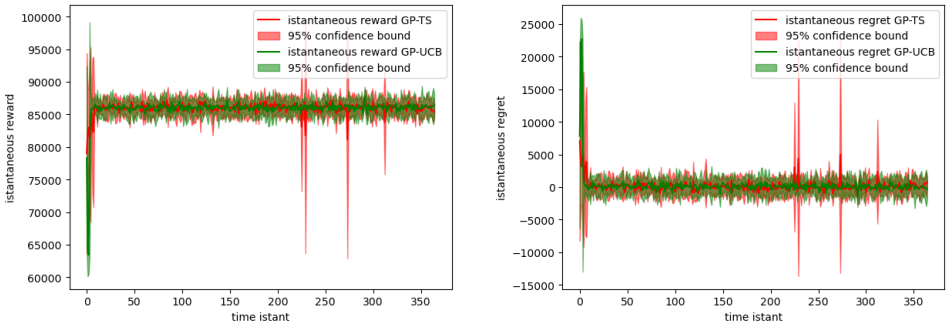


Figure 3.1: Comparison GP-TS vs GP-UCB1, instantaneous

These are the graphics of the instantaneous reward and the instantaneous regret of GP-TS and GP-UCB1. It's clear that, after little time instances, the instantaneous reward reaches immediately the best expected reward, while the instantaneous regret quickly decreases near zero. It can be noticed that the GP-UCB1 is pretty stable, pulling always arms close to the optimal at each round after few iterations, while GP-TS sometimes shows a larger confidence, even after the algorithm become stable. This could be cause by the fact that the arm is pulled using a normal distribution, so, even when the algorithm seems stable, it could explore other possible arms in the neighborhood of the optimal one.

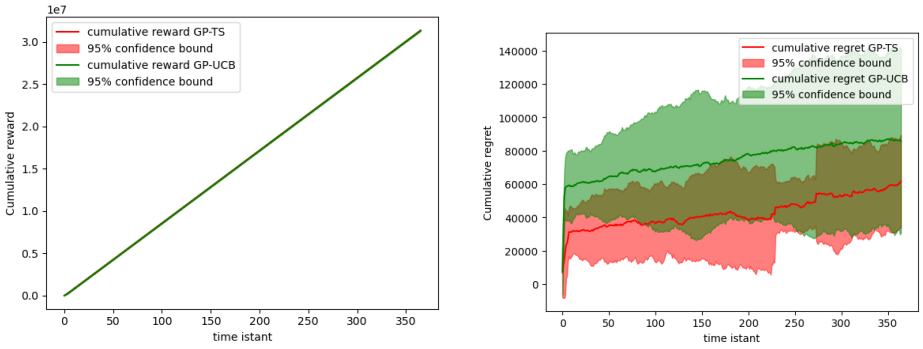


Figure 3.2: Comparison GP-TS vs GP-UCB1, cumulative

These are the graphics of the cumulative regret and reward. The cumulative reward is almost the same for both the algorithms, underlining that they almost reach the stability over the same arm. The cumulative regret is, as expected, sub-linear for both the algorithms. In addition, it's clear that the cumulative regret of the GP-TS is lower than the

one of the GP-UCB1, as expected, but it's also less stable (seems that the UCB1 has a lower slope), probably due to a continuous exploration of arms close to the optimal.

GP fitting

Here below the fitting of the GPs used in the GP-UCB1 and into the GP-TS, the pulled arms and the observed samples. The results are obtained after 365 rounds.

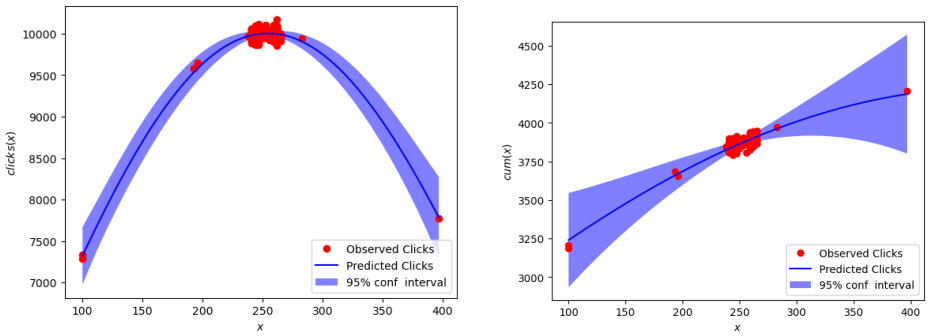


Figure 3.3: GP-UCB1. On the left the estimation of the curve $f(b)$ on the right the estimation of the curve $h(b)$

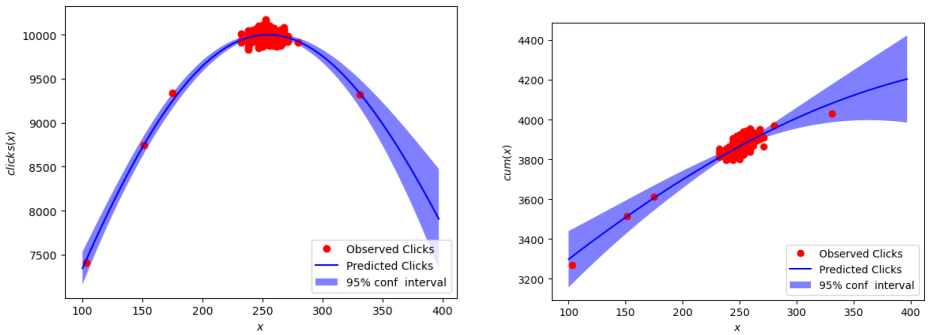


Figure 3.4: GP-TS. On the left the estimation of the curve $f(b)$ on the right the estimation of the curve $h(b)$

As we can see the GPs correctly estimate the curves.

4 | Step 3

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.

4.1. Description of the environment

In this step, we still consider only one class, but no curves are known. For the pricing problem, since TS algorithm provides better performances then UCB1, the algorithm estimate the pricing curve using TS and the advertising curves using GP-UCB1 and GP-TS.

4.2. Learning Algorithm

The algorithm is basically the same of the step two: it creates an handler (one for GP-UCB1 and one for GP-TS) that handle the GPs and, unlike the previous step, the handler also creates a TS learner for the pricing problem. At each round, The handler draw an arm from the TS algorithm and uses the corresponding price to pull the best joint arm like before. After the rewards are collected, this handler also updates the parameter of the TS learner.

4.3. Results

The results obviously are worse than the previous step, since no clairvoyant algorithm is used in the scenario. Here are the graphs of the algorithms' performances after 20 runs and a time period of one year (T=365).

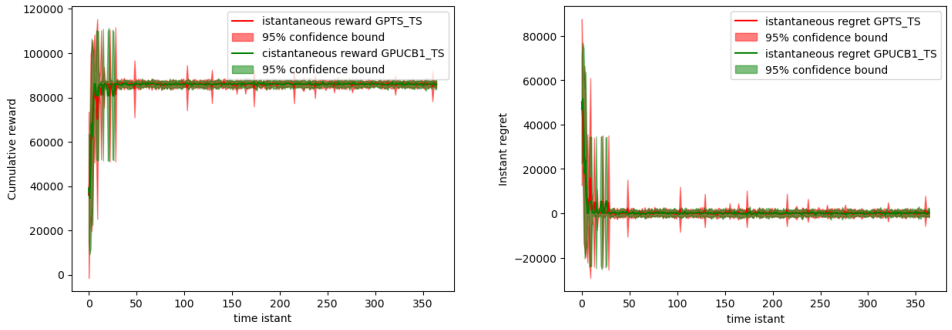


Figure 4.1: Comparison GP-TS_TS vs GP-UCB1_TS, instantaneous

These are the graphics of the instantaneous reward and regret. In this case, the GP-UCB1_TS algorithm takes longer than before to stabilize, having a large variance for the first 40 iterations. The GP-TS_TS, instead, is pretty much stable, pulling like before a different arm even after T=200 due to the variance of the normal distribution that is larger for not pulled arms.

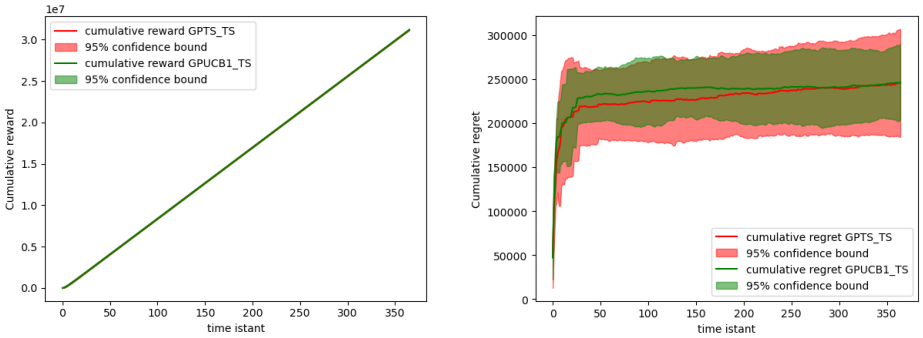


Figure 4.2: Comparison GP-TS_TS vs GP-UCB1_TS, cumulative

These are the graphics of the cumulative reward and regret. As the previous step, the cumulative reward is similar for both algorithms.After 20 iteration, GP-UCB1_TS become stable, without any big increases after that instant. The GP-TS_TS, instead, has a lower cumulative regret than the GP-UCB1_TS, but it keeps increasing, reaching the regret of the GP-UCB1_TS. As before, this behaviour probably because the GP-TS_TS is exploring the neighborhood of the optimum.

5 | Step 4

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-UCB1_ TS and GP-TS_ 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. Apply the GP-UCB1_ TS and GP-TS_ 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-UCB1_ TS 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.

5.1. Scenario Description

In this scenario, customers belong to three different classes depending on their features. The different features consider are the one described in the step 0. In particular, three different scenarios have to be considered:

1. In the first scenario, the structure of the context is known beforehand.
2. In the second scenario, there is no knowledge over the context.
3. In the third scenario, like before, there is no knowledge over the scenario, but the context can be generated using a context generation algorithm.

5.1.1. First scenario

In the first scenario, the structure of the context is known. Since it knows that three different classes exist, three different GP-UCB1_TS and three GP-TS_TS must be created, one per class. Each GP works like no other GP or classes exist. It pulls the best arm corresponding to its own class and update its model watching the reward corresponding to its class. After each iteration, in order to compute the cumulative reward, cumulative regret, instantaneous reward and instantaneous regret, the rewards obtained from each class must be summed, and compared with the sum of the optimal rewards for each class.

5.1.2. Second scenario

In this scenario, it assumes that only one class exists: it creates one GP-UCB1_TS and one GP-TS_TS and works like previous step, but in this case, it updates the model using the sum of the rewards of each class. Like previous scenario where classes are known, the cumulative reward, cumulative regret, instantaneous reward and instantaneous regret are computed by comparing the sum of the rewards for each class with the sum of the optimal rewards.

5.1.3. Third scenario

At the beginning of this scenario, the structure of the context is unknown. Every two weeks, a context generator is activated. The duty of the context generator is to find if a different context can be created basing on the reward obtained so far.

Learning Algorithm

Every two weeks, the algorithm creates a new GP for each context known. In order to do not lose the progress and updates made so far, we store every reward obtained at each iteration in a DataFrame, stored in the context generator, and massively update the GP with every reward obtained corresponding to its contexts. If the context generator doesn't find new contexts, the GP is not modified. Each GP works like previous steps: it doesn't know anything about other classes and GPs, and it updates its model using the sum of the rewards of its context.

Context Generator

The context generator is composed by a big DataFrame that collect, at each iteration, every reward for each class with the corresponding pulled arms. Every two weeks, the context generator activates. Its duty is to watch at the obtained reward so far and find

if it's better to generate new context or keep the previous ones. The context generator algorithm works as follows:

Algorithm 11 Context Generator

μ^*_{root} = max(mean of rewards of each arm)
$LB_{root} = \mu^*_{root} - argmax(\mu_{root}) * \sqrt{-\frac{\log(\delta)}{2*|argmax(\mu_{root})|}}$
for leaf in feature_to_check **do**
    for $j = 0, 1$ **do**
        $\mu^*_{leaf_j}$ = max(mean rewards of each arm where feature leaf = j)
        $LB_{leaf_j} = \mu^*_{leaf_j} - argmax(\mu_{leaf_j}) * \sqrt{-\frac{\log(\delta)}{2*|argmax(\mu_{leaf_j})|}}$
        $P_{leaf_j} = \frac{|leaf_j|}{|root|} - \sqrt{-\frac{\log(\delta)}{2*|root|}}$
    end for
    $LB_{leaf} = LB_{leaf_0} P_{leaf_0} + LB_{leaf_1} P_{leaf_1}$
end for
$LB^*_{leaf} = max(LB_{leaf})$
Repeat with leaf as new root if $LB^*_{leaf} > LB_{root}$
Otherwise, return root

At the end, the context generator return the generated contexts and, if they are different from the considered contexts of the GPs, it reinitialize new GPs considering those contexts.

Parameter choice In this algorithm, we considered as confidence $\delta = 0{,}95$ since the algorithm is considering a confidence interval of δ. In addition, we decided to put a lower bound that depends only on the times an arm is played, since the value of the mean will become more precise each time the algorithm select that arm and collect the corresponding reward.

5.2. Results

5.2.1. First scenario

The result of the first scenario are similar to the results found at the step 3, since the algorithm works in the same ways. Here are the graphs of the algorithms' performances after 10 runs and a time period of one year (T=365).

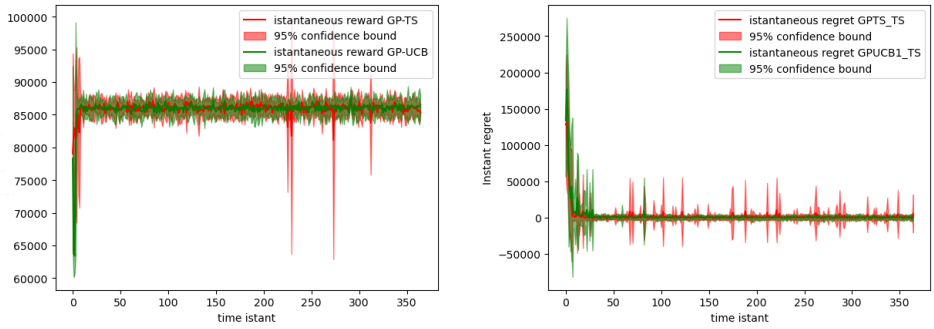


Figure 5.1: Comparison GP-TS_TS vs GP-UCB1_TS, instantaneous

These are the graphics of the instantaneous reward and regret. The behaviour is the same of the previous step. Obviously, since the reward is computed by summing three different rewards, the variance of the GP-TS_TS result to be larger, and this can be clearly seen from the instantaneous regret graph.

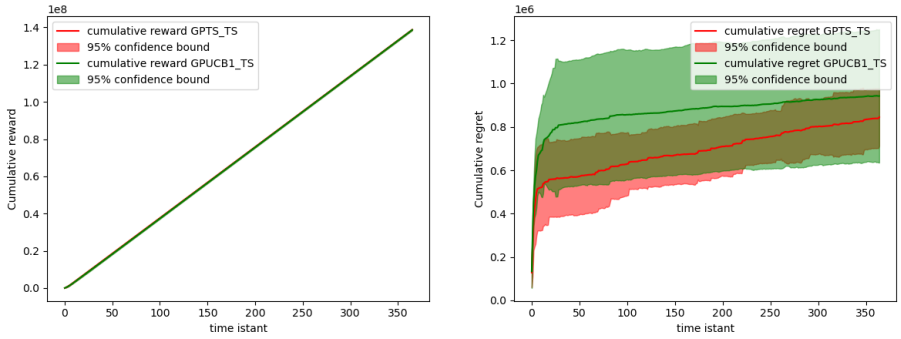


Figure 5.2: Comparison GP-TS_TS vs GP-UCB1_TS, cumulative

These are the graphics of the cumulative reward and regret. The cumulative reward is, as before, linear and equal for both algorithms. The cumulative regret is sub-linear, the GP-UCB1_TS become stable, as before, after 20 time instant and the GP-TS_TS one increase as before due to the variance of the arms.

5.2.2. Third scenario

In this graphs are compared the results of all the previous scenarios, highlighting which curve belongs to which scenario. Due to time reasons, the algorithm has performed 7 runs over a time period of one year (T=365).

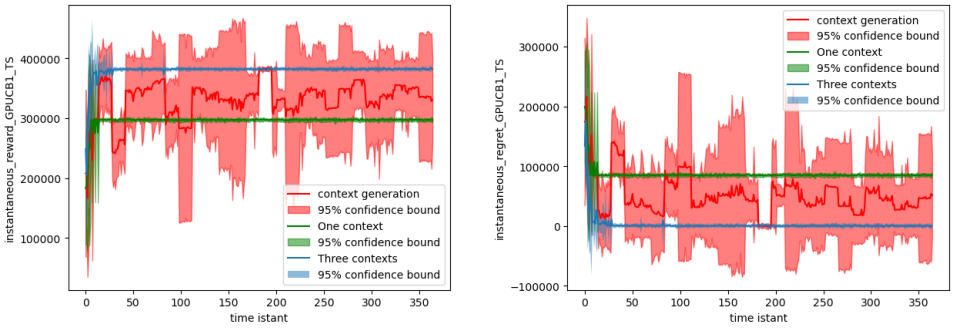


Figure 5.3: Comparison GP-UCB1_TS, instantaneous

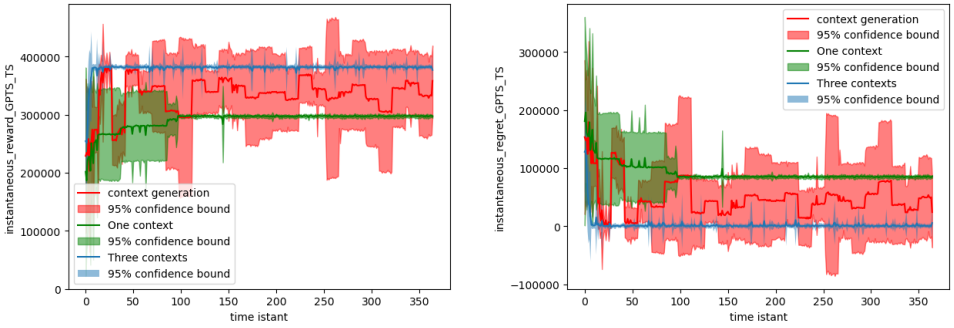


Figure 5.4: Comparison GP-TS_TS, instantaneous

The algorithm of the first scenario provides the best performances; it quickly stabilizes to the correct reward and a zero regret for both GP-UCB1_TS and GP-TS_TS. The algorithm of the second scenario, since it as no knowledge, will pull the same arm for all the classes, obtaining a reward that is worse than the optimal one and, therefore, it's regret will never reach zero. The curve related to the third scenario is placed between the two curves. It can be easily seen that the context change every 14 time instances, stabilizing always to a different value. The variance of the curve remains very wide since the context is changing and less data are provided for some contexts. In addition, these graphs highlight that the two curves provide an upper bound (knowledge of contexts) and a lower bound (no knowledge) to the curve, setting a minimum and a maximum to the regret (or reward) we can possibly obtain at each time instant.

The cumulative rewards of the algorithms works as expected, the algorithm that as the complete knowledge of the contexts has a better slope, while the worst function is clearly

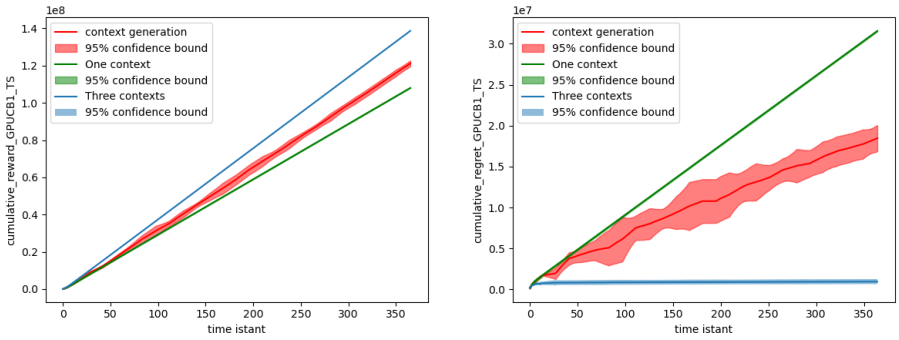


Figure 5.5: Comparison GP-UCB1_TS, cumulative

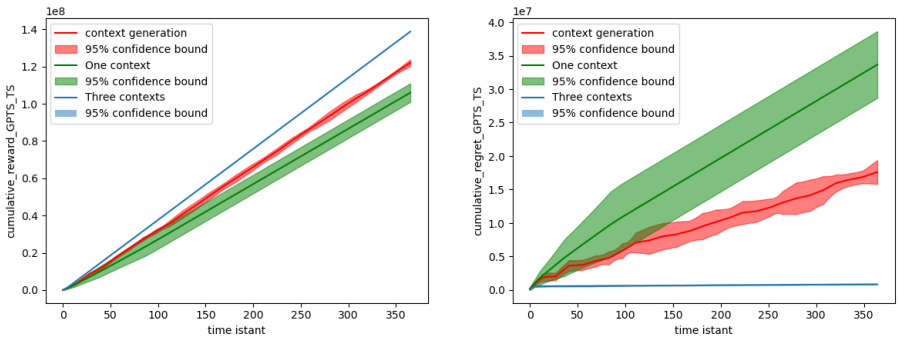


Figure 5.6: Comparison GP-TS_TS, cumulative

the second algorithm's one. The cumulative regret is much lower for the first scenario that's hardly presentable, since it rapidly finds the best rewards for each classes. The curves of the third scenario, instead, are sub-linear. Due to the context change, the confidence of the curve doesn't shrink immediately like others, but it remains wider for a longer time. In any case, the performances of the algorithm without any knowledge are, again, an upper bound to the previous curves.

6 | Step 5

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.

6.1. Description of the environment

A Non-Stationary Environment is the case of study of this section. In particular, the Non-Stationarity regards the unknown pricing curve, resulting in three seasonal phases.

The seasons are spread over the year, i.e. the one covers 122 days, as well as the second one, while the third 121 days. These three phases could be due to the ”warm” season, where people tend to drink less coffee daily (that is usually an hot beverage) because of the high temperature, so people could be more willing to pay for an high quality coffee blend. To represents the latter, in the second phase (warm season) the conversion probability associated to higher prices is higher with respect to the normality.

It has been assumed that the curves related with the advertising are known, therefore the learning will focus just on the pricing.

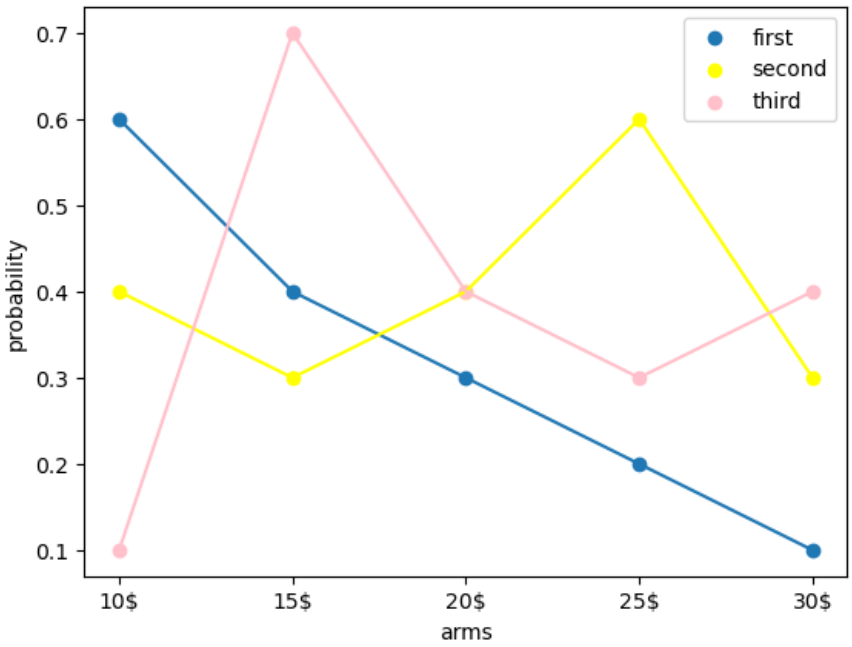


Figure 6.1: Seasons

6.2. Learning Algorithm

Since the curves related to the Advertising are known, playing on the bids is equivalent to use the Advertising Clairvoyant algorithm that receives as input the price played by the bandit on price. Therefore, the complete procedure has been done as follow.

Algorithm 6 Procedure Step 5

1: **for** $t = 1, 2, \dots T$ **do**
2: Compute $p^t \leftarrow Pricing\_Bandit()$
3: Compute $b^*(p^t) \leftarrow$Advertising Clairvoyant(p^t)
4: Play $(p^t, b^*(p^t))$ and collect the reward
5: Update $Pricing\_Bandit()$ given $reward^t$ and p^t
6: **end for**

A classic UCB1 algorithm has been applied and compared with two different types of $Pricing\_Bandit()$ for Non-Stationary environment, both being different extensions of an UCB1. The first one is a passive UCB1 bandit, which takes the advantage of a *sliding window* to tackle the Non-Stationarity; the second is an active UCB1 bandit that uses a change detector in order to detect the change of the stationarity of a time series.

6.2.1. Passive UCB1 (*sliding window*)

The passive UCB1 (*sliding window*) works taking into account only the last τ samples (the ones included in the *sliding window*) for its plays. Thanks to that mechanism, the passive UCB1 is able to avoid the problem of a Non-Stationary environment, since, inside of the *sliding window*, the time series is almost stationary.

However, a great sensitivity analysis must be done on the parameter τ, the size of the *sliding window*. It is a common practice to fix $\tau \propto \sqrt{T}$

The pseudo-code of the Algorithm is the following one.

Let:

- $n^t_{\tau,p}$ the number of times arm p has been played from time $t-\tau$ up to time t
- $\widehat{E[con_p]}^t_\tau = \frac{\sum_{k=t-\tau}^{t} CON(p,n(b^k))^k}{\sum_{k=t-\tau}^{t} n(b^k)^k}$ the empirical mean based on the last τ rounds of the conversion rate of price p
- $UP^t_p = 5\sqrt{\frac{2log(t)}{n^t_{\tau,p}}}$

Note that $n^t_{\tau,p}$ represents the number of clicks that have been performed when price p was shown, from time $t-\tau$ to time t.

The pseudo-code is defined as follows.

Algorithm 7 UCB1 sliding window

1: **for** $t = 1, 2, \dots T$ **do**
2: if $\exists$ arm p s.t. $n^t_{\tau,p}$=0, then play $p^t \leftarrow p$
3: Else play $p^t \leftarrow \underset{p}{\text{argmax}}\ (\widehat{E[con_p]}^t_\tau + UP^t_{p,\tau})p$
4: Collect $rew^t_{p^t}$ and observe $CON(p, n(b))^t$
5: Update for each p ,$\widehat{E[con_p]}^t$ and ,UP^t_p
6: **end for**

Instruction 2 aims to play first the arms that have not been played yet in the last tau rounds. The latter instruction is necessary for the Algorithm, in order to give a bit more weight to the exploration phase because the sliding window is not a fully representative sample of the time series.

6.2.2. Active UCB1 (*CUSUM detector*)

The active UCB1 takes advantage of a detector that monitors the estimated conversion probability to detect a change in the unknown distribution of every arm.

The detection is performed using the CUSUM change detection point Algorithm, whose functioning is described below.

Algorithm 8 CUSUM

1: input: M,ϵ,h
2: For each arm, with the first M samples compute the reference $\bar{X}_p^o$
3: From sample $M+1$ check the cumulative positive deviation $g_p^+(t)$ and the cumulative negative deviation $g_p^-(t)$.
4: if $g_p^-(t) > h$ or $g_p^+(t) > h$, report a change for arm p

Being:
$g_p^+(t) = max\{0, g_p^+(t-1) + s_p^+(t)\}$
$g_p^-(t) = max\{0, g_p^-(t-1) + s_p^-(t)\}$

where
$s_p^+(t) = (x_p(t) - \overline{X_p^o}(t)) - \epsilon$
$s_p^-(t) = -(x_p(t) - \overline{X_p^o}(t)) - \epsilon.$

The learning procedure of this active UCB1 is defined as follow:

Algorithm 9 Active UCB1

1: Input: α
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Play a random arm p^t w.p. α
4: Play $p^t \leftarrow \underset{p}{\text{argmax}}(E[\widehat{con}_p]^t_{\tau_p} + UP^t_{p,\tau_p})p$ w.p. 1-α
5: Receive reward and observe $CON(p^t, n(b))^t$
6: Perform CUSUM detection on $CON(p^t, n(b))^t$ for arm p^t
7: If detection set $\tau_{p^t} = t$
8: **end for**

Also the active version of the UCB1 requires an higher degree of exploration with respect to the standard UCB1, in order to deal with Non-Stationary environments. Therefore, the hyper-parameter α is used to increase the exploration of the algorithm.

As for the passive UCB1, the active one requires an accurate sensitivity analysis in order to tune the parameters M, ϵ and h of the CUSUM detector and parameter α of the active UCB1. In particular α and h the parameters should be:

- $\alpha \in O(\sqrt{\frac{log(T)}{T}})$
- $h \in O(log(T))$

6.3. Theoretical guarantees

In general, Non-Stationary bandits like the one above discussed,assure a regret, except for logarithmic terms, of the order:

$$R(T) \in O(\mid A \mid T^{\frac{1+\alpha}{2}})$$

Where $\beta \in [0, 1[$ is defined such as:

$$\mid Break\ Points \mid \in O(T^{\beta})$$

6.4. Results

The active version of the UCB1 and the passive version of it have been compared to the classical UCB1 on the base of the cumulative reward, cumulative regret, instantaneous

reward and instantaneous regret. The results are shown below.

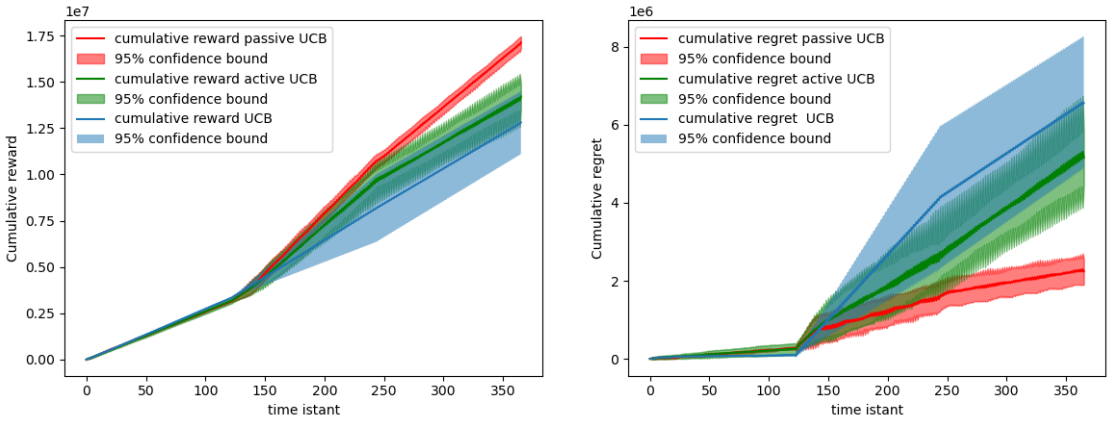


Figure 6.2: Comparison Active UCB1 vs passive UCB1, cumulative

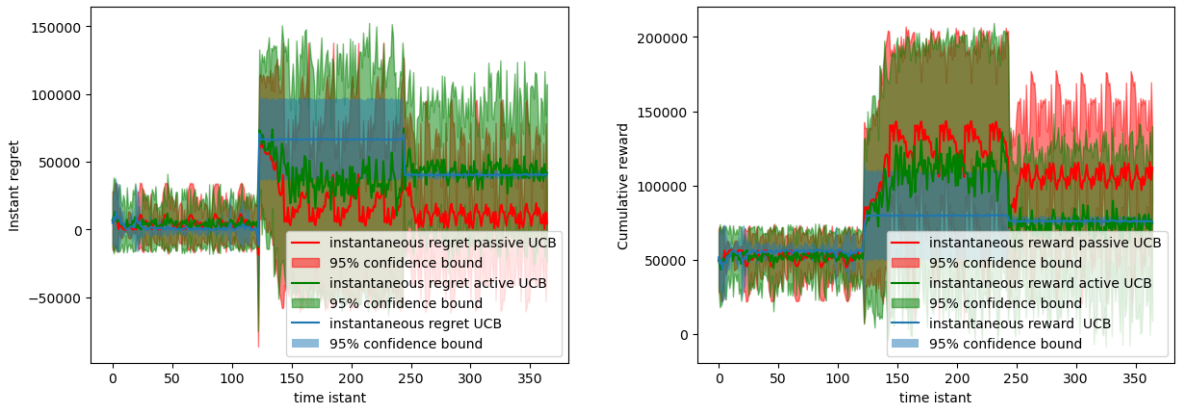


Figure 6.3: Comparison Active UCB1 vs passive UCB1, instantaneous

As expected both the active UCB1 and passive UCB1 perform better than the classic UCB1, in a Non-Stationary environment. We can observe that the passive UCB1 outperforms also the active one. Probably because the change in the conversion probabilities of the prices is not strong enough to be detected by the CUSUM. Indeed, in the last phase, the regret of the CUSUM is aligned with the one of the UCB1, that means that they are playing the same non-optimal arm. Probably the CUSUM UCB1 is designed for Non-Stationary Environments with strong abrupt changes that are far one from the other, so that the CUSUM is able to detect them and the active UCB1 is able to stabilize.

6.5. Sensitivity Analysis

6.5.1. Sliding window

As said before the sliding window should be of $\sqrt{T}$, which in our case is almost 19. The sensitivity analysis has been done comparing the cumulative regret of UCB1 sliding window, using different values of the window size. The results are in the figures below.

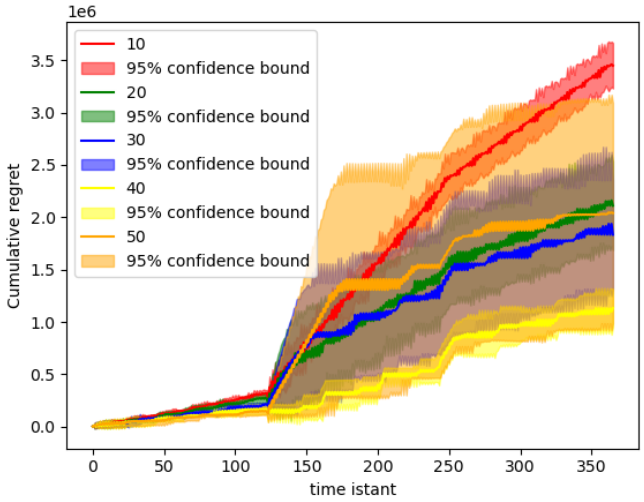


Figure 6.4: Sensitivity analysis, window size

In practice, τ should be proportional to $\sqrt{T}$ in this case almost 19. A sensitivity analysis has been done in order to tune that parameter. The optimal value for the parameter seems to be 40 which is $2\sqrt{T}$.

6.5.2. *M*

M is the first tuned parameter of the active UCB1. The optimal value for M seems to be around 20, that is $\sqrt{T}$. Not incidentally, its tuning follows the empirical rules for the tuning of τ, size of the sliding window. Indeed, they roughly shoudl respect th same theoretical rules. Here the plot:

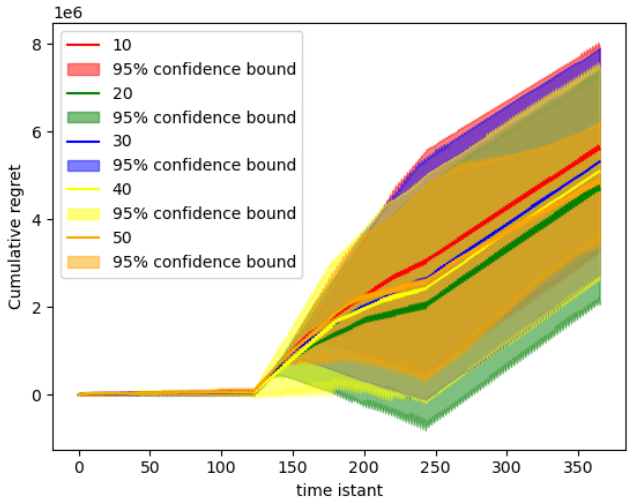


Figure 6.5: Sensitivity analysis, M

6.5.3. ϵ

ϵ is a parameter that controls the sensitivity toward a single sample for the CUSUM detection. High value for ϵ means low sensitivity.

The optimal value for ϵ has been elicited out of the following ones [0.08 , 0.1 , 0.12 , 0.14 , 0.16]. From the results the optimal value seems to be to be around 0.08. Therefore, a sensitive CUSUM is required in the considered scenario.

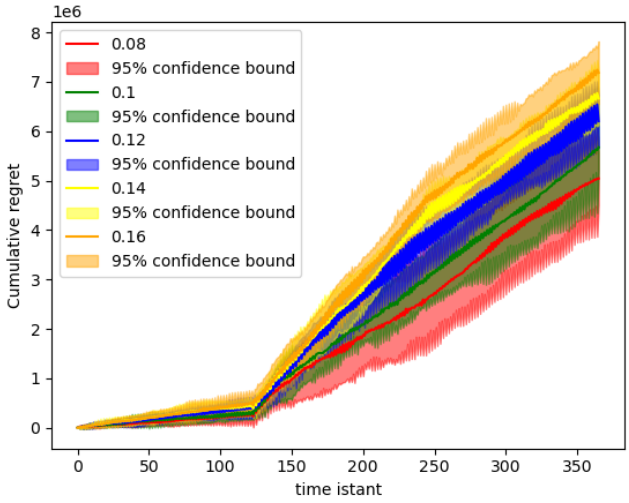


Figure 6.6: Sensitivity analysis, ϵ

6.5.4. h

h is the threshold for the detection in the CUSUM.

The optimal value for h has been explored among [0.8 , 1 , 1.2 , 1.5 , 1.8]. The optimal vaule for h seems to be $\in [0.08, .., 1.2]$.

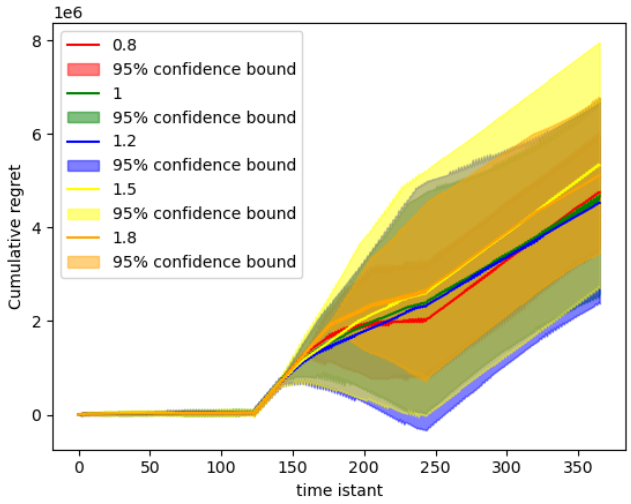
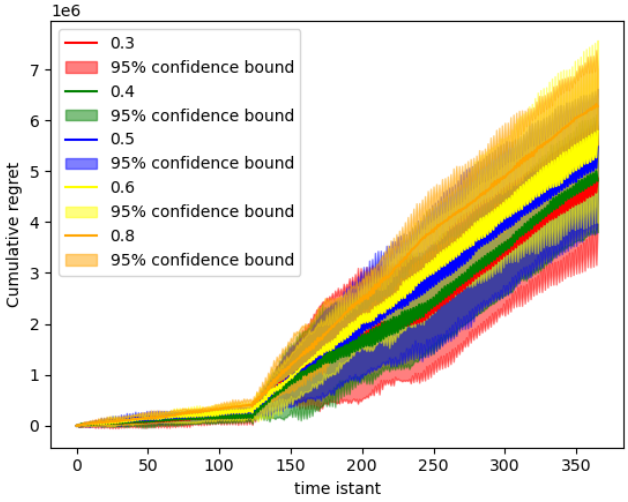


Figure 6.7: Sensitivity analysis, h

6.5.5. α

Tuning of α, exploration parameter for the active UCB1.

The optimal value for α has been explored among [0.3 , 0.4 , 0.5 , 0.6 , 0.8]. The optimal vaule for α seems to be 0.3.



7 | Step 6

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

7.1. Description of the environment

In this chapter, we address the challenge posed by adversarial settings within two distinct environments, employing the EXP3 algorithm alongside our established counterparts, UCB1 Cusum and UCB1 SW. Our focus is exclusively on the first class of customers. These environments are differentiated based on the quantity and timing of changes, which are elucidated in the subsequent sections:

7.1.1. 3 Phases

The environment remains consistent with that utilized in Step 5, with the noteworthy distinction being the fixation of the bid to an arbitrary value, specifically set at 300. It is important to note that in each configuration, the value associated with the optimal arm varies, and these changes occur abruptly. This setup is designed to provide insights into the responsiveness and adaptability of our algorithms under these specific conditions, shedding light on their behavior in scenarios marked by sudden shifts in optimal arm values.

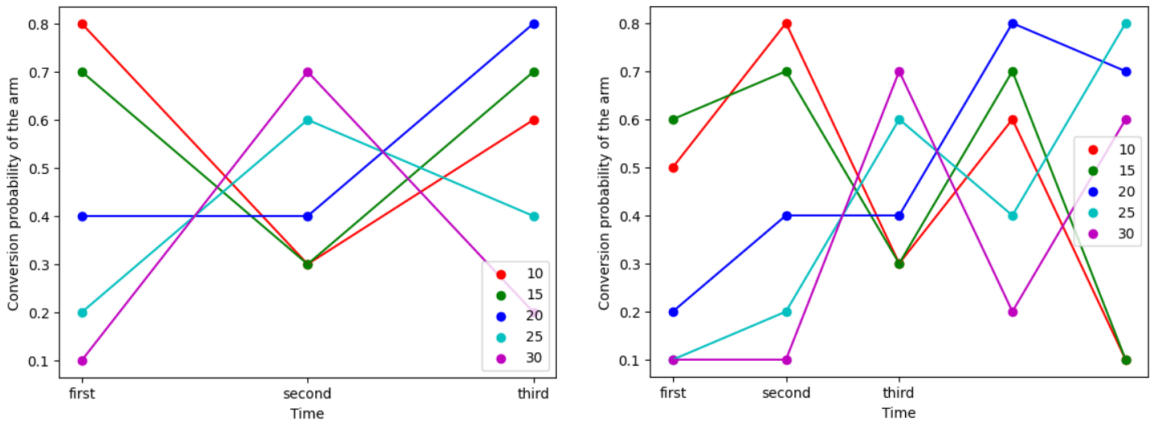


Figure 7.1: On the left trend of arms for 3 phases, on the right for 5

7.1.2. 5 Phases

Similar to the previous environment, the bid remains fixed at 300. However, in this case, we introduce five distinct phases characterized by frequent changes, occurring approximately every 15 time steps. The second, third, and fourth phases align with those encountered in the preceding environment. Importantly, each phase is associated with a unique optimal arm, thereby introducing dynamic shifts that challenge the adaptability of our algorithms.

7.2. Design of the Algorithm

EXP3

"Exp3" is an acronym for "Exponential-weight algorithm for Exploration and Exploitation." This algorithm operates by managing a set of weights associated with each available action. These weights play a pivotal role in the algorithm's decision-making process, influencing the random selection of the next action to be taken. Furthermore, the algorithm adjusts these weights accordingly, augmenting them when favorable outcomes are observed and diminishing them in response to less favorable outcomes.

The algorithm utilized for both scenarios has been implemented as shown in figure 7.2:

In the formulation, the symbol η represents a regularization parameter that serves the purpose of fine-tuning the value of the estimated reward. Without this regularization, in situations where an exceptionally high reward is observed, it could disproportionately favor one action, causing other actions to be rarely chosen.

Additionally, we introduce an egalitarianism factor $\lambda \in [0, 1]$. This factor allows us to

Algorithm 1 Exp3 Algorithm

Given $\lambda \in [0,1]$ initialize the weight vector $w_i = 1$ for all arms i
for $t = 1, 2, \ldots, T$ **do**
  $\forall i$, calculate $p_i = (1-\gamma) \cdot \frac{w_i}{\sum_j w_j} + \frac{\gamma}{K}$, where K is the number of arms
  Sample an arm I_t from the probability distribution p_i
  Observe the reward r_t for arm I_t
  Calculate the estimated reward $\hat{r}_t = \eta \cdot \frac{r_t}{p_{I_t}}$
  Update $w_{I_{t+1}} \leftarrow w_{I_t} \cdot e^{\frac{\hat{r}_t}{K}}$
  $\forall j$, j $\neq I_t$, $w_{j_{t+1}} \leftarrow w_{j_t}$
end for

Figure 7.2: EXP3 Pseudo-code

modulate the inclination to select actions in a uniform random manner. In essence, when $\lambda = 1$, the weights exert no influence on the action choices at any given step, resulting in equal probability for all actions to be selected.

7.3. Theoretical guarantees

As we expect results similar to those produced by the UCB SW and UCB CUSUM algorithms, we also anticipate that the regret will follow a comparable trend. We foresee the regret to grow at a rate of approximately :

$$R(T) \in O(\mid A \mid T^{\frac{1+\alpha}{2}})$$

7.4. Results

We will now proceed to assess the performance of the EXP3 algorithm in both of these environments. This evaluation involves a comparative analysis with the performance of UCB CUSUM and UCB SLIDING WINDOW algorithms. It is our anticipation that, in the first environment, EXP3 is likely to exhibit a comparatively inferior performance, whereas in the second environment, it is expected to demonstrate improved performance.

7.4.1. 3 Phases

Observing the results, it becomes apparent that the performance of EXP3 surpasses our initial expectations. It exhibits a slightly superior performance compared to CUSUM,

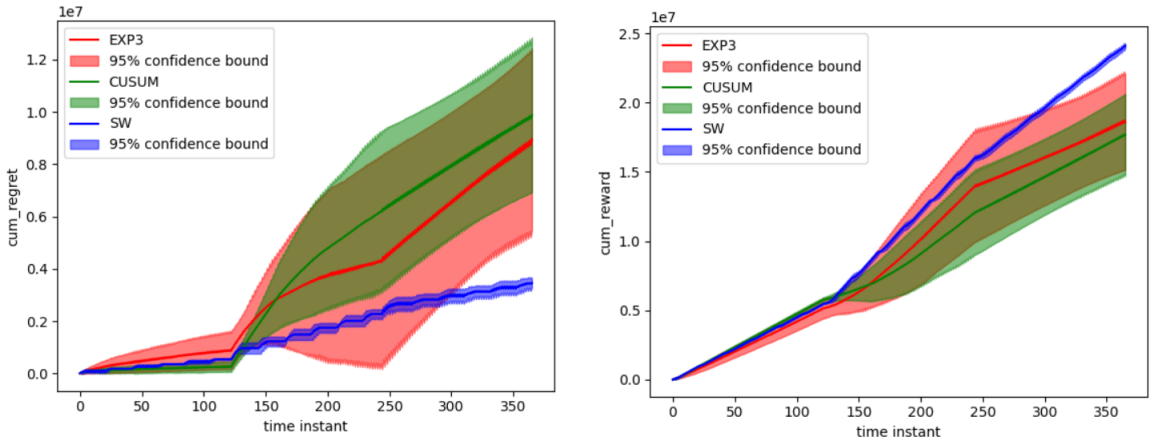


Figure 7.3: Comparison EXP3 vs CUSUM vs SW, cumulative

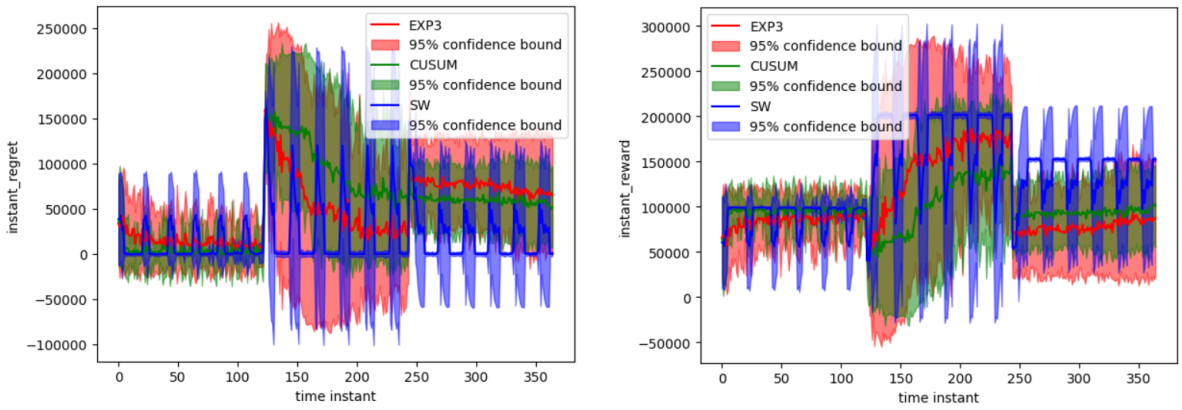


Figure 7.4: Comparison EXP3 vs CUSUM vs SW, instantaneous

particularly when first abrupt setting change occurs, but it suffers the last change, where it weights are now too high. EXP3 demonstrates a remarkable ability to adapt swiftly and effectively.

In contrast, SW stands out as the most proficient among the algorithms, as evidenced by the cumulative plots. It notably maintains only one-third of the cumulative regret and achieves almost double the cumulative reward in comparison to the other algorithms. However, it is worth noting that SW displays a degree of instability, requiring periodic recalibration approximately every 20 time steps, which corresponds to the length of its sliding window.

7.4.2. 5 Phases

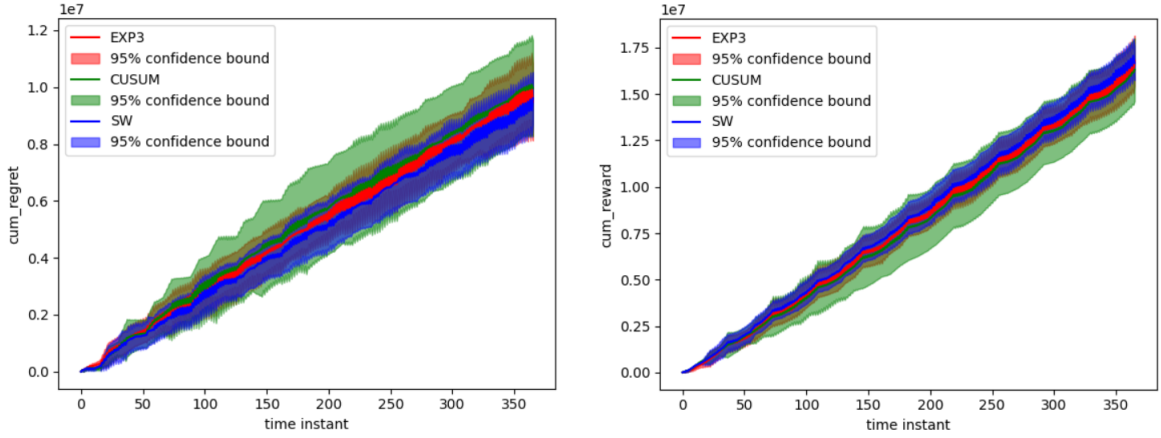


Figure 7.5: Comparison EXP3 vs CUSUM vs SW, cumulative

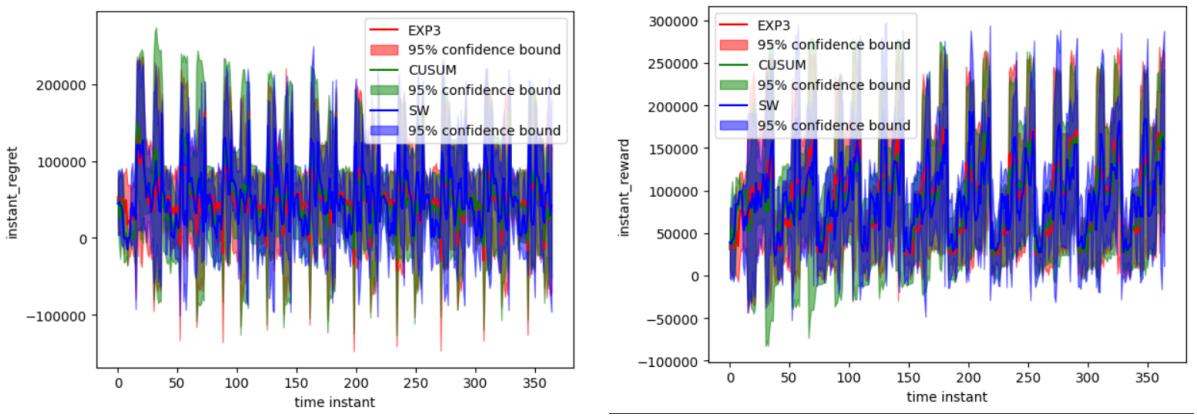


Figure 7.6: Comparison EXP3 vs CUSUM vs SW, instantaneous

In this particular instance, the performance of the EXP3 algorithm deviates from our initial expectations. While it still outperforms CUSUM, it falls short of surpassing SW, which continues to demonstrate its superiority as the best-performing algorithm. However, it's noteworthy that the gap between EXP3 and SW has narrowed significantly; their performance is now remarkably similar.

Despite SW's dominance in terms of cumulative plots, a closer examination reveals that its instantaneous plots exhibit considerable instability with fluctuating trends. In contrast, EXP3 exhibits a more consistent and linear behavior, akin to the patterns we observed in the previous environment.

The unexpected underperformance of EXP3 can likely be attributed to our utilization of the η parameter for adjusting expected rewards. This choice has the effect of slowing

down the growth of weights associated with different actions. Consequently, there is a slower rate of change in the probability distribution, which places a greater emphasis on exploration rather than exploitation.

7.5. Sensitive analysis

7.5.1. η

The selection of η as a regularization parameter serves a crucial purpose in our experiment. It effectively prevents the weights from growing unbounded, especially in cases where the rewards are exceptionally high. Without this regularization, the algorithm's weights would escalate rapidly to infinity within a minimal number of iterations, rendering the algorithm impractical and unusable.

In our experimentation, we found that an appropriate value for η was 0.0001, which we ultimately employed for this specific experiment. We also conducted experiments by considering the logarithm of the rewards with various bases, but these alternative approaches yielded inferior results. For simplicity and given their limited relevance to the ongoing project, we have omitted the detailed analysis of these alternative approaches.

7.5.2. γ

The γ parameter plays a pivotal role in the behavior of the algorithm. Its value significantly influences the trade-off between exploration and exploitation. When γ is set to a lower value, the algorithm places more emphasis on exploiting the arms with higher estimated rewards, making it less explorative. Conversely, as γ approaches 1, the probability distribution over arms in EXP3 becomes increasingly uniform, and the agent essentially selects arms randomly without relying on any prior information.

In Figure 7.7, we can discern a distinct trend: greater values of γ correspond to suboptimal results, while the best performance is achieved when γ is set to 0.1.

However, it's important to highlight that as we decrease γ to smaller values, we must also adjust the η parameter accordingly. This adjustment becomes necessary because lower values of γ imply reduced exploration. Consequently, the same arm will be selected repeatedly, causing its weight to grow exponentially. In extreme cases, this weight can reach infinity, leading to the probability associated with that arm becoming NaN (Not-a-Number). This highlights the delicate balance required when configuring both γ and η.

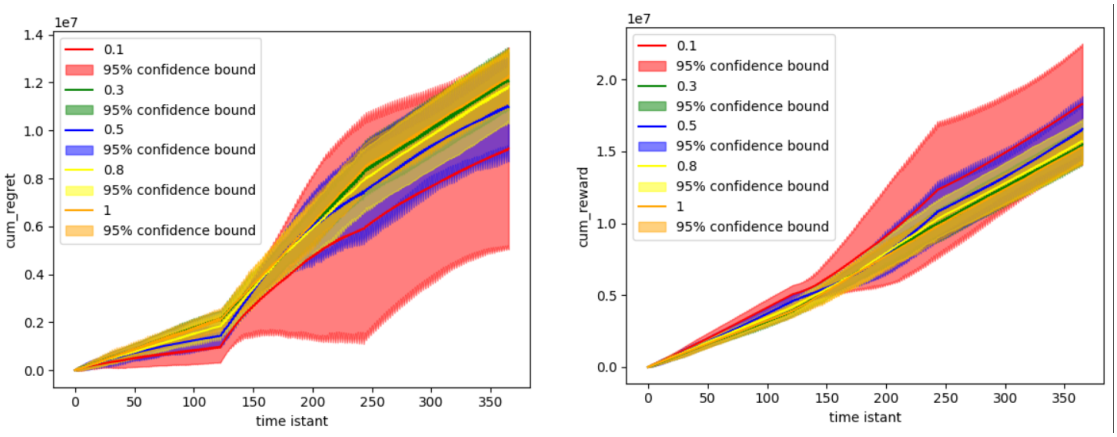


Figure 7.7: Analysis of γ value