POLITECNICO
MILANO 1863

Data Intelligence Applications

Pricing and matching project
Technical report

Anna Giovannacci,
Yunus Ege Saygılı,
Alice Casali,
Francesco Amorosini

September 2021

Contents

1 Introduction

1.1 Purpose

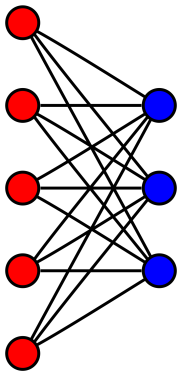
The purpose of this document is to explain how our team interpreted and implemented the assignment of the Pricing and Matching project, composed by eight steps.
The setting can be generally described as it follows: consider a shop whose objective is to maximize its revenue over a long period of time. The shop sells two kinds of **items**, and is affiliated to four different **customer classes**. The shop has also access to four different kinds of **coupons**, which can be used in order to obtain a discount on the second item after buying the first one. In order to reach his goal, the shop has to choose the optimal price for the items, as well as the assignment of promos to each one of its customers.

Along the eight steps of this project, the amount of known variables and their behaviour will be changed in an increasing difficulty.

1.2 Definitions

- *Conversion rates*: a conversion rate is the share of customers that are willing to buy a given item. According to the frequentist approach, a conversion rate is a good estimate of the probability that a customer will effectively buy an object.
- *Arms*: in Multi Armed Bandit context, the arms are the set of candidates for a learning algorithm.
- *Objective Function*: a function whose value needs to be maximized (or minimized) by means of an informed choice of its parameters.
- *Regret*: the cumulative regret for a given learning algorithm is the difference between the value of its Objective Function and the optimal value, aggregated over a time interval.
- *Round*: in bandit algorithms, a round is a function that returns a Bernoulli distributed variable that defines whether or not a certain event realized (in our case, if an item has been bought or not).
- *Bipartite Graph*: is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V.

Figure 1: An example of bipartite graph



1.3 Core algorithms: Thompson Sampling and Upper Confidence Bound

As they will be mentioned very often in this document, Thompson sampling (TS) and Upper Confidence Bound (UCB) are briefly explained in this section. These are two **bandit algorithms**. A bandit algorithm is a classic reinforcement learning problem, in which an agent is faced with a finite number of arms, each with a different reward distribution, and the agent is trying to maximise his objective function based on trials. The biggest challenge that the agent has to overcome is famously known as the exploration/exploitation dilemma.

In particular, TS is a **Bayesian approach** that identifies rewards of the arms by sampling a continuous probability distribution (prior).
Since, as we mentioned, in our case the rewards are Bernoulli distributed, a good choice for prior distributions is the β distribution (which is the conjugate prior probability distribution for the Bernoulli distribution). The β distribution is initialized with parameters $(\alpha, \beta) = (1, 1)$ which describe a uniform distribution, and which will be updated according to the obtained rewards.

Upper Confidence Bound is instead a **frequentist approach**, where arms are picked according to their empirical mean and confidence interval. After a new reward sample is collected, the empirical means are updated and the confidence intervals are shrunk accordingly.

2 Theoretical Steps

2.1 Step 1

"Provide a mathematical formulation of the problem for the case in which the daily optimization is performed using the average number of customers per class. Provide an algorithm to find the optimal solution in the offline case in which all the parameters are known. Then, during the day when customers arrive, the shop uses a randomized approach to assure that a fraction of the customers of a given class gets a specified promo according to the optimal solution. For instance, at the optimal solution, a specific fraction of the customers of the first class gets P0, another fraction P1, and so on. These fractions will be used as probabilities during the day."

The assignment is describing an offline weighted matching problem, considering a scenario in which all parameters are known. In general, a matching problem can be visualized using a **complete bipartite graph**, which in our particular case is defined as follows:

- On the right side: the nodes representing customers entering the shop during the day.
- On the left side: the nodes representing coupons that have to be assigned
- Edges: they match every customer with every possible coupon, and they are weighted by the expecting reward obtained by such assignment.

Once the bipartite graph has been built, a subset of its edges is chosen in such a way that each and every node on the left side is matched with exactly one node on the right side. The subset of edges is selected in order to optimize an **objective function**, which is described below by means of a **linear programming** problem.

2.1.1 Sets

$$I = \{1, 2, 3, 4\} : \text{Customer Classes} \tag{1}$$

$$J = \{1, ..., N\} : \text{Customers} \tag{2}$$

$$T = \{0, 1, 2, 3\} : \text{Promo Types} \tag{3}$$

$$K = \{0, ..., m\} : \text{Prices for Item 1} \tag{4}$$

$$L = \{0, ..., n\} : \text{Prices for Item 2} \tag{5}$$

2.1.2 Parameters

$$N : \text{Number of customers:} \tag{6}$$

$$N_i : \text{Number of customers for class } i, \text{ for } i \in I \tag{7}$$

$$p1_k : \text{The value of price } k \text{ for item 1, for } k \in K \tag{8}$$

$$p2_l : \text{The value of price } l \text{ for item 2, for } l \in L \tag{9}$$

$$pr_t : \text{The discount ratio of promo } l, \text{ for } t \in T \tag{10}$$

$$cr1_{ik} : \text{Conversion rate of item 1 for class } i \text{ under price } k, \text{ for } i \in I,\ k \in K \tag{11}$$

$$cr2_{ilt} : \text{Conversion rate of item 2 for class } i \text{ under price } l \text{ with promo } t, \text{ for } i \in I,\ l \in L,\ t \in T$$

$$c_{ij} : 1 \text{ if customer } j \text{ is a member of class } i, \text{ 0 otherwise, for } i \in I,\ j \in J \tag{12}$$

$$NP_t : \text{Number of promos of type } t, \text{ for } t \in T \tag{13}$$

2.1.3 Variables

$$sp1_k : 1 \text{ if price } k \text{ is selected for item 1, 0 otherwise, for } k \in K \tag{14}$$

$$sp2_l : 1 \text{ if price } l \text{ is selected for item 2, 0 otherwise, for } l \in L \tag{15}$$

$$spr_{jt} : 1 \text{ if promo } t \text{ is selected for customer } j, \text{ 0 otherwise, for } j \in J,\ t \in T \tag{16}$$

2.1.4 Objective function

$$max \sum_{i,j} c_{ij} * ((\sum_k p1_k * sp1_k * cr1_{ik}) + (\sum_k sp1_k * cr1_{ik}) * (\sum_t spr_{jt} * (1 - pr_t) * (\sum_l p2_l * sp2_l * cr2_{ilt})))$$

2.1.5 Constraints

$$\sum_j c_{ji} = N_i \tag{17}$$

$$\sum_i c_{ji} = 1 \tag{18}$$

$$\sum_t NP_t = N \tag{19}$$

$$\sum_k sp1_k = 1 \tag{20}$$

$$\sum_{l} sp2_l = 1 \tag{21}$$

$$\sum_{j} spr_{jt} = NP_t \tag{22}$$

$$\sum_{t} spr_{jt} = 1 \tag{23}$$

$$0 \leq cr1_{ik} \leq 1 \tag{24}$$

$$0 \leq cr2_{itl} \leq 1 \tag{25}$$

By solving the optimization problem, we will obtain, for each kind of coupon, how many of them will be assigned to each class. For example, having 100 promos of type p_2, we can assign 50 of them will be distributed to customers of class 0, 12 of them to customers of class 1, 8 of them to customers of class 3, and then 30 of them to customers of class 4. During the day, a function that uses this fractions as probabilities will assign a promo to each customer.

2.2 Step 2

"Consider the online learning version of the above optimization problem, identify the random variables, and choose a model for them when each round corresponds to a single day. Consider a time horizon of one year."

For the online learning version of the problem, we have the following random variables with corresponding probability distribution

- Number of customers belonging to class i for a given day, N_iS: *Truncated Gaussian*
- Whether or not customer j buys item 1 for price k, $cr1_{ik}$: Bernoulli
- Whether or not customer j buys item 2 for price l with promo t, $cr2_{ilt}$: Bernoulli

The online version of the algorithm consider a round for each day, this means that for each day there is an unknown number of customers coming to the shop and these customers can buy or not the items based on the selected prices and promo fractions. The shop does not know the relative conversion rate so we need to improve our guessing one day at a time until the end of the year.

3 Stationary Environment

In the previous sections we have defined the theoretical basis of our problem. From now on, also practical results will be shown. For all the steps, we will fix:

- *Customers class distributions* as a truncated Gaussian: the number of customers per class that visit the shop during the day is generated by the environment as a sample of such distribution
- *Candidate prices* for each item.
- *True conversion rates*, the probabilities that a customer buys an item for a given price. These parameters will be chosen randomly and sorted in such a way that the lowest price is also the most popular for a certain class. Once chosen, the conversion rates will be fixed and -unless specified otherwise- they will be only known by the environment.

3.1 Shop

- *Customer classes*: as a customer walks into the shop, we assume to know to which class he/she belongs to.
- The *best candidate price* for the second item.
- The *distribution of promos* among customers of a given class.
- The *true conversion rates*, parametrized on classes, promos and prices.

3.2 Online Experiments

During the day, the algorithm performs the following steps:

1. Sample the total number of customers.
2. Print a number of coupons relative to the number of sampled customers.
3. Pull an arm for each learner (TS and UCB, independently).
4. For each customer:
   (a) Assign a promo to the customer. The promo can be assigned randomly according to fixed promo fractions, or it can be improved using some matching algorithm (see Step 5)
   (b) Calculate partial reward:
      - If the true conversion rates are unknown to the shop, perform a round for item 1 and 2. The result of the round is a Boolean variable that describes whether or not the customer bought an item.

- Calculate the revenue relative to this customer. If the conversion rates were known, skip the previous point and just calculate the expected reward.
- Normalize this result by the maximum reward possible

(c) Update the internal parameters of each learner:

- UCB update: update the empirical means and shrink the confidence bound accordingly.
- TS update: update the parameters of the beta distribution.

5. Save the total reward in a daily list of rewards.

At the end of the simulation, we plot the reward and the regret for each learner, analyzing the results and drawing some conclusions.

3.3 Step 3

"Consider the case in which the assignment of promos is fixed and the price of the second item is fixed and the goal is to learn the optimal price of the first item. Assume that the number of users per class is known as well as the conversion rate associated with the second item. Also assume that the prices are the same for all the classes (assume the same in the following) and that the conversion rates do not change unless specified differently below. Adopt both an upper-confidence bound approach and a Thompson-sampling approach and compare their performance."

Our first assignment is to learn the price of the first item, knowing, respectively:

- Price of the second item.
- Promos assignment.
- Conversion rate of the second item.
- Number of customers for each class.

In this scenario, we assume that the customers can buy the second item if they bought the first. With this reasoning the second item can be considered as an extension of the first one. It's like deciding the price of a laptop and propose to the customer also a mouse which has a fixed price. Also, we have a limited amount of promos, so the price is chosen in order to have a good discount for the maximum number of customers possible. In this setting we have two shops that round over the first price and just add the expected revenue of the second item:

- TS: which learns the best possible price using Thompson Sampling.
- UCB: which learns the best possible price using Upper Confidence Bound.

On the other hand, the third shop is supposed to represent the clairvoyant and to know all the parameters. So, for each customer, we calculate an expected reward using this formula:

$$\text{conversion\_rate}_1 * \text{clairvoyant\_price}_1 + \\ \text{conversion\_rate}_2 * \text{conversion\_rate}_1 * \text{price}_2 * (1 - \text{coupon})$$

The TS and UCB shops are assumed to know everything except for the price of the first item.

3.3.1 Clairvoyant calculation of the reward

Shop clairvoyant is supposed to know also the price of the first item. To obtain it, we do an offline simulation.
Reward is calculated as follows:

1. For each candidate of price1, the mean for each class of customers is taken, and then we multiply it by the true conversion rate and by the price of item 1.

2. After that, we consider each possible percentage of discount and we do a similar calculation for item 2. The main difference is that we calculate the discounts, the promo fractions, and also that price of item 2 is known.

Figure 2: Cumulative Revenue plot for Step 3

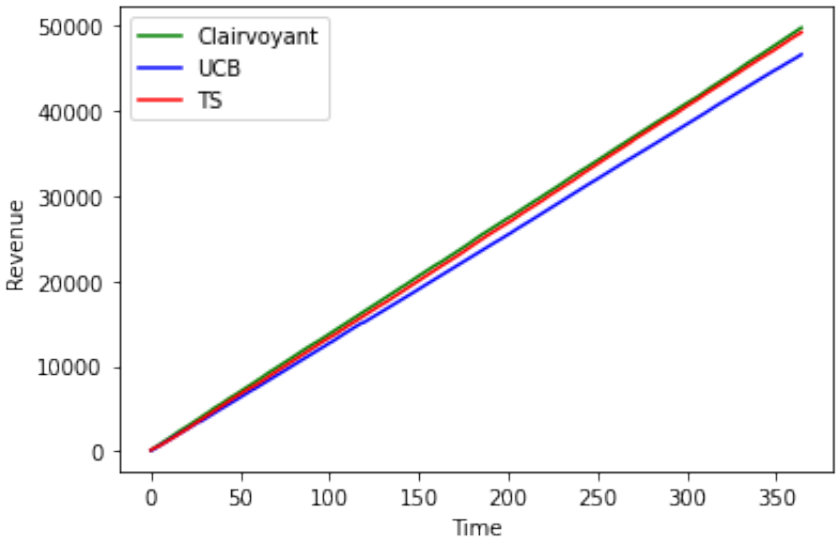
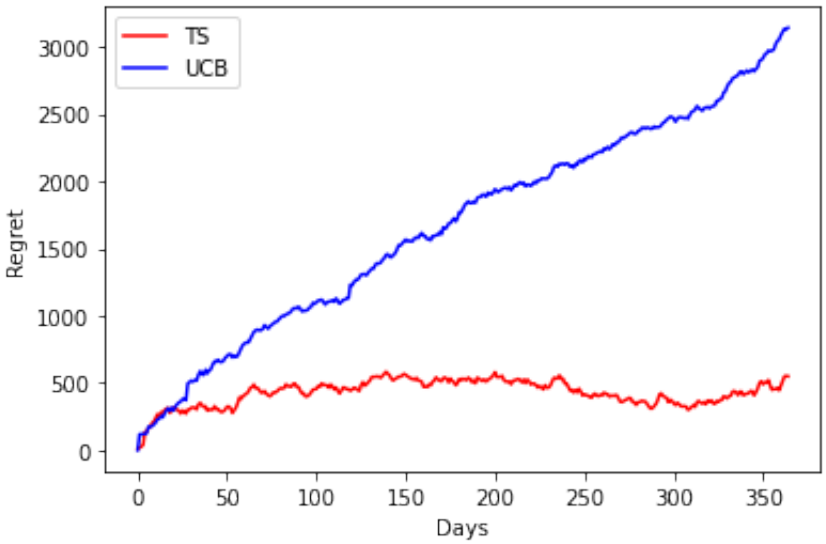


Figure 3: Cumulative Regret plot for Step 3



The gathered data are plotted in the images above. We can see that both shops are quite close to the optimal theoretical revenue set by the clairvoyant, but TS always shows slightly better performances that add up through the year.

3.4 Step 4

"Do the same as Step 3 when instead the conversion rate associated with the second item is not known. Also assume that the number of customers per class is not known."

In this scenario we are following exactly the same steps of the previous point, but this time we have to estimate the amount of customers that are willing to buy at a given price, as well as the number of customers that will enter the shop daily. This step, in a sense, is more similar to a realistic setting.

3.4.1 Environment and algorithms

Environment and algorithms remain almost unchanged, save for the following additions:

- **Estimation of customers**:
  Everyday we estimate the average number of customers by an iterative formula:
  $$\mu_{t+1} = (\mu_{t-1} \times (day - 1) + N_t)/day \tag{26}$$
- **Round for second item**:
  Since the conversion rate for the second item in unknown, we need to use again the round function and sample a Bernoulli variable.

Figure 4: Cumulative Revenue plot step 4

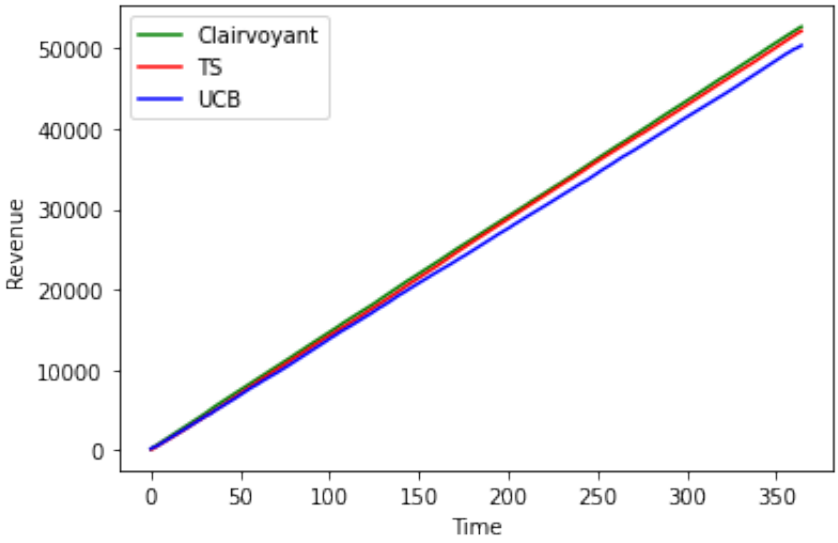
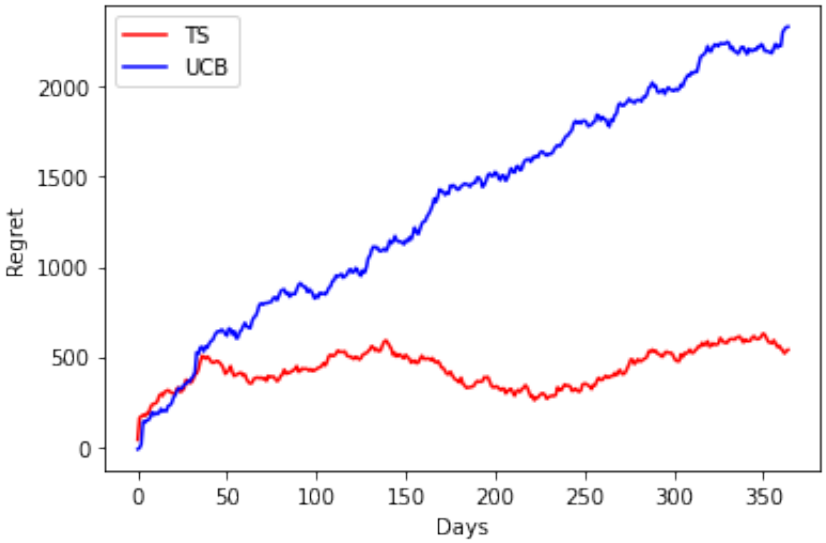


Figure 5: Cumulative Regret plot step 4



The above results are quite similar to the previous ones, but we noticed that in this scenario TS takes a little more time to converge to the optimal solution, as we can tell from the Regret spike in the first 50 days.

3.5 Step 5

"Consider the case in which prices are fixed, but the assignment of promos to users need to be optimized by using an assignment algorithm. All the parameters need to be learnt."

In this scenario we know the fixed prices for both items, but the promo assignment needs to be learnt optimized using a matching algorithm. To address this problem, at the beginning of each day we make use of the Hungarian algorithm in order to solve the promo assignment.
First of all, we define a **bipartite graph** as follows:

- On the right side, we have the expected number of customers
- On the left side, we have the printed coupons

Since we do not know how many customer will come each day, we make an estimate as we have seen in Step 4. The fraction of promos we should assign is fixed so the amount of printed coupons will be proportional to the number of customers.

3.5.1 Clairvoyant Weights

The clairvoyant reward works easily; knowing the prices and the conversion rates and, the weights of the edges are calculated using the following formula (which returns the expected reward resulting from matching promo i with customer j):

$$\begin{aligned} weight_{ij} = price\_item\_1 * conversion\_rate\_1 + \\ conv\_rate\_1 * price\_item\_2 * conversion\_rate\_2 * promo \end{aligned} \tag{27}$$

Then, we apply Hungarian Algorithm to the matrix of the weights. Since Hungarian returns a minimal cost matching, we consider the negative value of the weights in order to maximize the matching expected reward.
Hungarian Algorithm works as follows: for each row, the smallest entry is found, and then it is subtracted to all the other entries. The same is done in each column. So we have a matrix composed by 0 and other numbers. We try to cover all the 0s. Trying to use the less number of lines possible, if we obtain, N (=maximum matrix dimension) lines, then we are finished and we proceed with the assignment, otherwise we try to compensate by choosing the minimum number that is not marked and adding that weight to a marked column.

3.5.2 Thompson Sampling and UCB weights

For TS an UCB weights, we pull the arms doing a different action. In fact, according to the selected candidate we choose respectively:

- a sample from the beta distribution
- the updated upper confidence bound

The weights represent the reward of assigning to a customer a given promo.Then, we apply Hungarian as explained previously. As for the conversion rates and the amount of daily customers, these are estimated in the same way described in Step 4.

Figure 6: Cumulative Revenue plot step 5

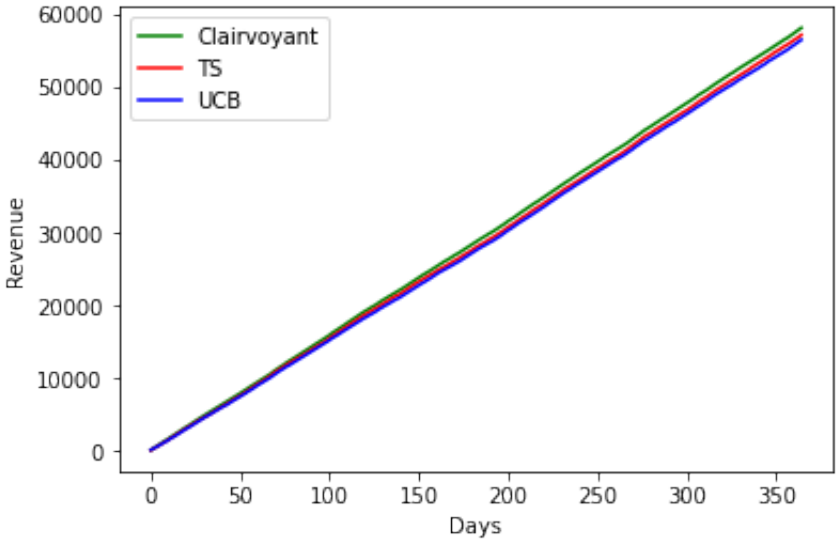
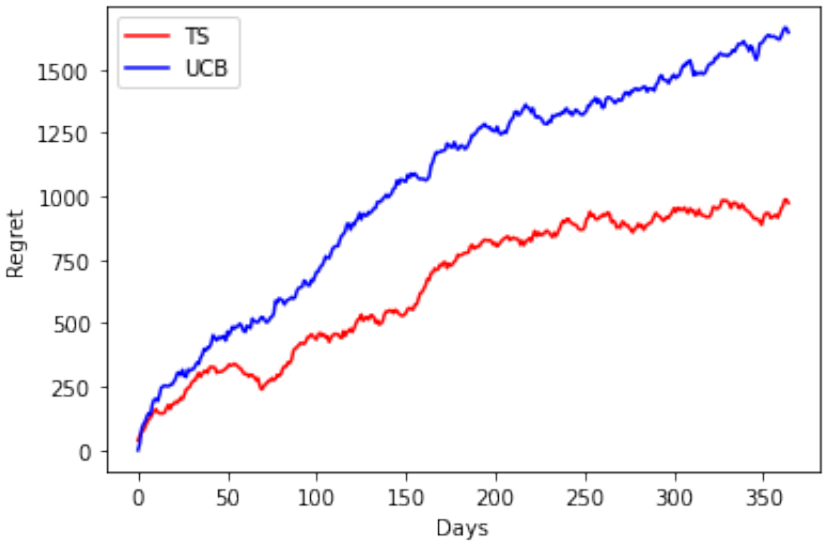


Figure 7: Cumulative Regret plot step 5



As we progress through the assignment and introduce more unknown variables, we are noticing that that the gap between UCB and TS is slowly narrowing.

3.6 Step 6

"Consider the general case in which the shop needs to optimize the prices and the assignment of promos to the customers in the case all the parameters need to be learnt."

In this step, we should learn not only the price of the two items, but also the assignment of the promos. As a first attempt, we tried using TS and UCB as we did in the previous steps, combining together all the different learning methods for the unknown variables, and as expected we concluded that Thompson Sampling works better. Since in this scenario there are quite a few unknown variables, we also explored many different methods to learn them all at once, and then compared all together:

- *3-step Thompson Sampling with independent price learners*: it makes use of three learners: two of them are used to learn the optimal prices for the two items, while the other learns the weights for solving the matching problem. This approach chooses price1 and price2 independently and then calculates the resulting matching.
- *3-step Thompson Sampling with dependent price learners*: just like the previous shop, this one is provided with 3 TS learners for learning the prices and the matching weights. The difference with the previous method is that the second price is chosen according to the first price choice.
- *UCB*: it works in the same way as the 3-step TS with independent price learners, but using UCB as learner.
- *2-step Thompson Sampling*: there is one learner that learns the prices, considering all the possible permutations of them; as in the other methods, the remaining learner that learns the matching according to the prices.
- *Big Thompson Sampling*: it learns prices and matching at the same time; and it uses all enumerations of the prices and the corresponding enumerations of discounts for matching. When we do an update, we update everything at the same time instead of doing it in two separate steps. This is the one that costs more.

We combined all these different approach together and ran a common simulation.

3.6.1 Analysis of the results

After the first simulations, we noticed that in general the results with TS Learner were better than UCB Learner, hence we focused on studying different approaches in order to improve the TS learning process. In the end, the methods

described above all converge to the solution, but they also show that there is a trade-off between more in-depth examination and faster convergence. After all, at the beginning of each day we had to decide both prices and the matching all together and all of these variables are dependant on each other.

On one hand, we had the Big-TS to be able to not miss any kind of relation. But this involved a heavy computation and too many arms to be considered. In the next steps we decided to drop this method, as the time horizon needed to make up for the initial regret is way higher than one year.

On the other hand, we had 3-Step TS which decides prices independently, which was the simplest and fastest learner to converge. This approach is computationally less heavy, however we thought it may be not be so reliable because it will miss most of the dependencies between prices and promo assignment. We introduced 3-Step TS with dependent prices and 2-Step TS in order to find a balance in this trade-off. Both approaches do miss some dependencies, but not as much as 3-Step Independent TS and they are also computationally lighter than Big-TS. However, it would be safe to state that 2-Step TS approach is closer to Big-TS, while 3-Step Dependent TS is closer to the independent one.

After running our simulations and seeing the results, we can say that 2-Step TS finds a better balance in this trade-off and performs better. Big-TS is slow to converge, 3-Step Dependent TS may occasionally perform better than 2-Step TS but it is not stable enough. Its performance varies too much and sometime falls behind even Big-TS. Thus, out of all options, we select 2-Step TS as our best approach and we continue with this method in the later steps.

Figure 8: Cumulative Revenue plot step 6

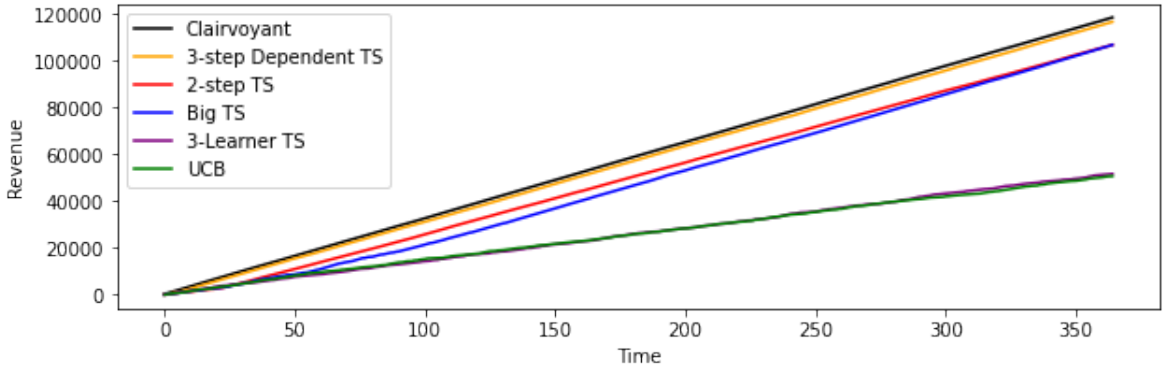


Figure 9: Revenue plot step 6

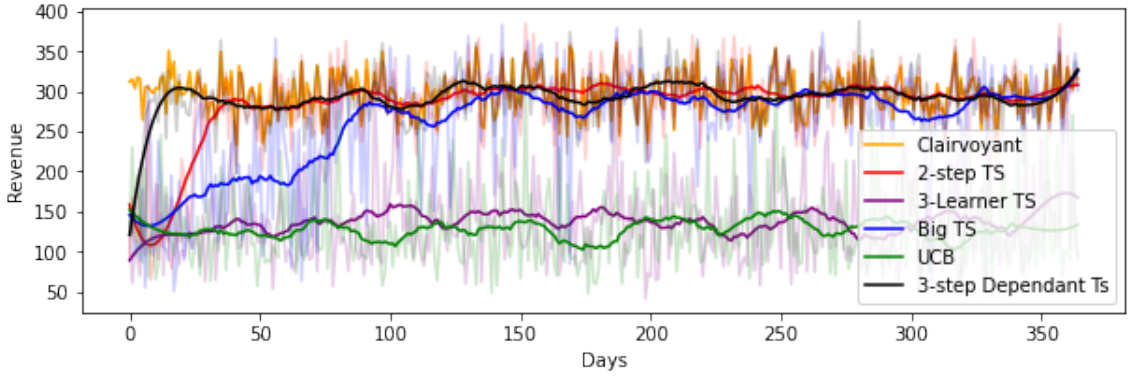
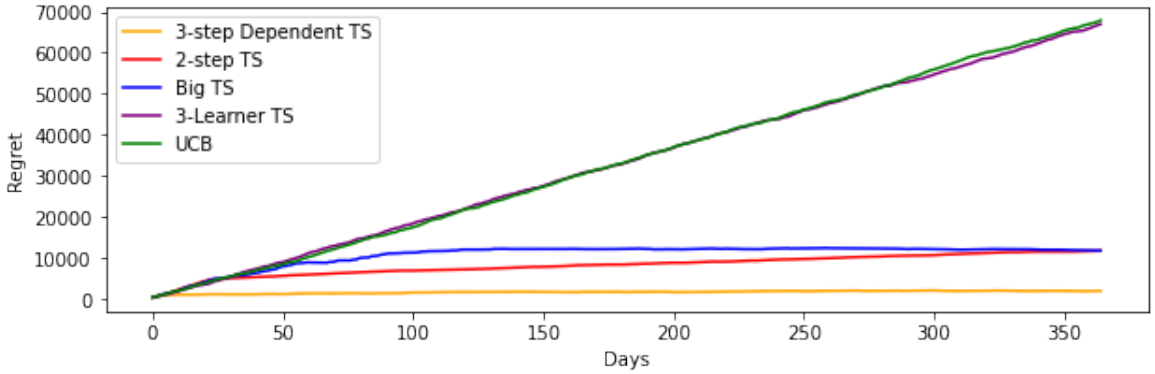


Figure 10: Cumulative Regret plot step 6



4 Non-Stationary Environment

In the last steps we consider conversion rates as non-stationary, i.e. they change over time. The setting we are testing is **abrupt changes**: our time of observations is divided into phases, and in each phase conversion rates are constant, while between phases they are different. This means that every rounding is performed with respect to the current phase we are in.

4.1 Step 7

"Do the same as Step 6 when the conversion rates are not stationary. Adopt a sliding-window approach."

Since the time horizon refers to one year, we thought of phases as seasons, hence we divided the time horizon in 4 phases, each of 90 days. In this setting we compared two different learners: the 2-steps-TS defined in the previous chapter, and a new TS learner that introduces a **sliding window** approach. Given a constant number τ called *window size*, this method considers only the samples that are τ days old. When a samples becomes outdated, the effect of that sample on the Sliding Window TS (SWTS) is canceled: this means, for Thompson sampling, to delete its update on the (α, β) parameters: it chooses the maximum between 1 and the parameters, in order to avoid beginning from 0 or negative numbers; on the other hand, when Upper Confidence Bound is working, if the day is out of the sliding window, we calculate the average again after having cut out that sample.
For the flow, look at step 6. Only main difference is here we calculate the clairvoyant for each phase, in order to have a different one every 90 days. Even here, we used 2-step Thompson Sampling to a comparison because it is more efficient.

4.1.1 Sliding Window in Non-Stationary Environment

For the Non-Stationary Environment, we consider abrupt changes. We have 4 phases in this setting and our SW Learner has window size 91. We arranged the conversion rates to make drastic changes from phase to phase. In this setting we can see the effect of sliding window. When we change the phase, the previous phases rewards become actually misleading. While, the version with no sliding window still take those rewards into account, the sliding window forgets them over time and achieves a better result. Below you can find see the revenues and regrets.

Figure 11: Cumulative Revenue plot step 7

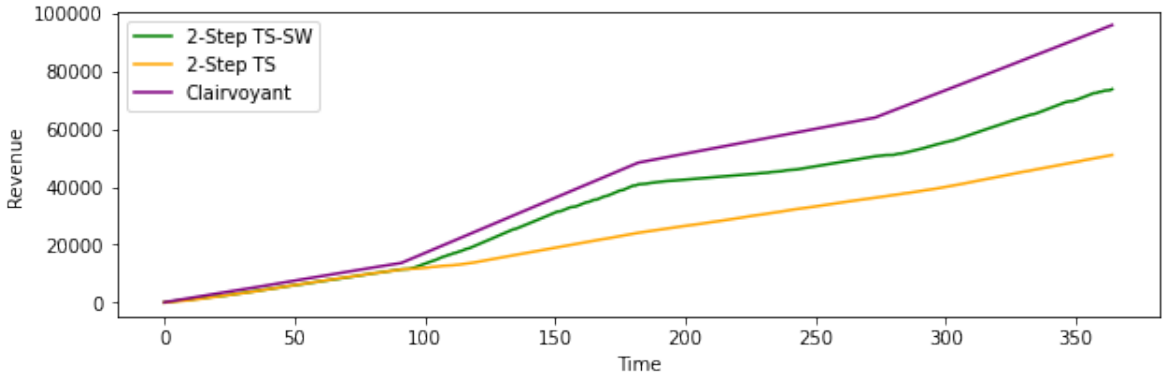
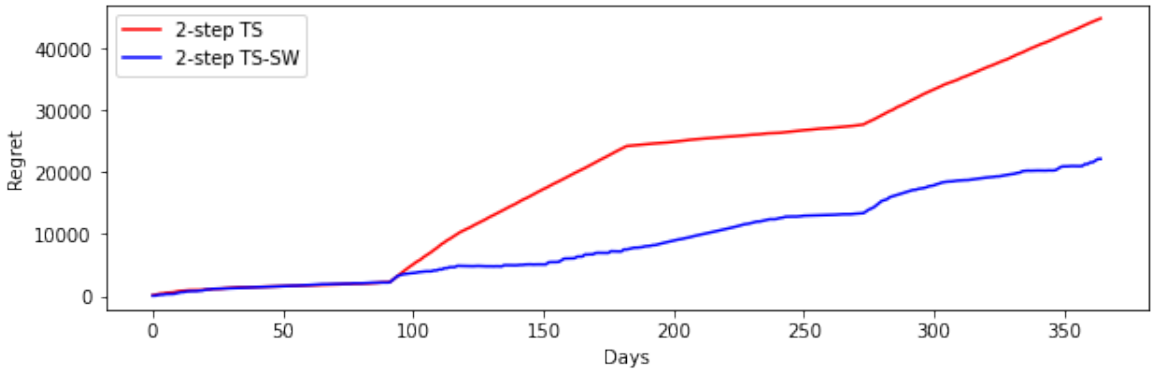


Figure 12: Cumulative Regret plot step 7



4.2 Step 8

"Do the same as Step 6 when the conversion rates are not stationary. Adopt a change-detection test approach."

This step is similar to the one above, but instead of using a sliding window approach, we make use of a **change-detection test**. This test is able to detect when there is a change in the environment conditions, and, in case the test result is positive, the learner parameters are reset . In practice, The test works by comparing the slope of the cumulative reward curve over a recent time frame with the slope of the overall curve. When their ratio is bigger than a given threshold, a change is detected. In this step we used Thompson Sampling, so our beta distribution are reset to $(\alpha,\beta)=(1,1)$.
After a detection, the observation frame is flushed and the whole process is restarted. The workflow is pretty much the same we saw in Step 7, only substituting the sliding window approach with the phase change detector.

Figure 13: Cumulative Revenue plot step 8

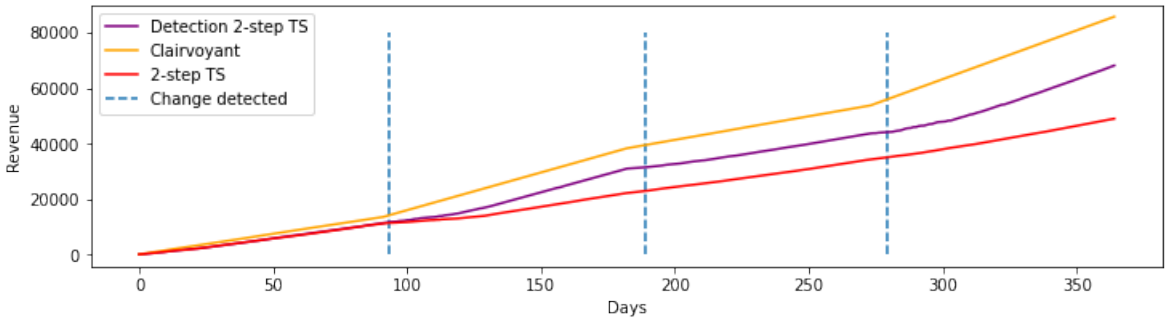


Figure 14: Revenue plot step 8

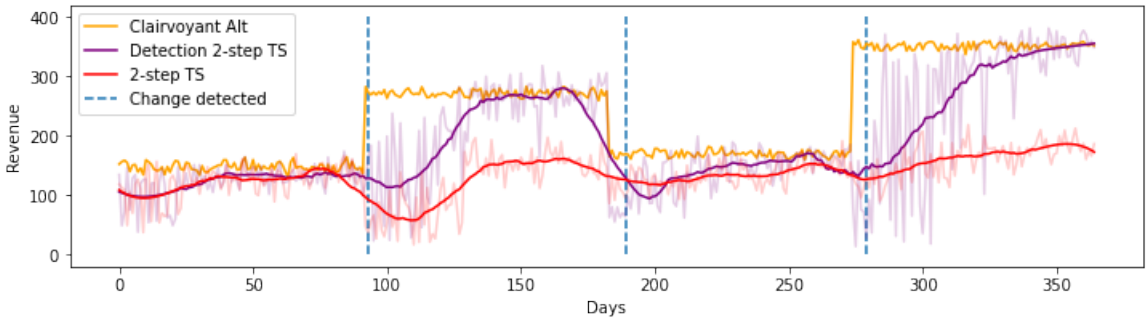
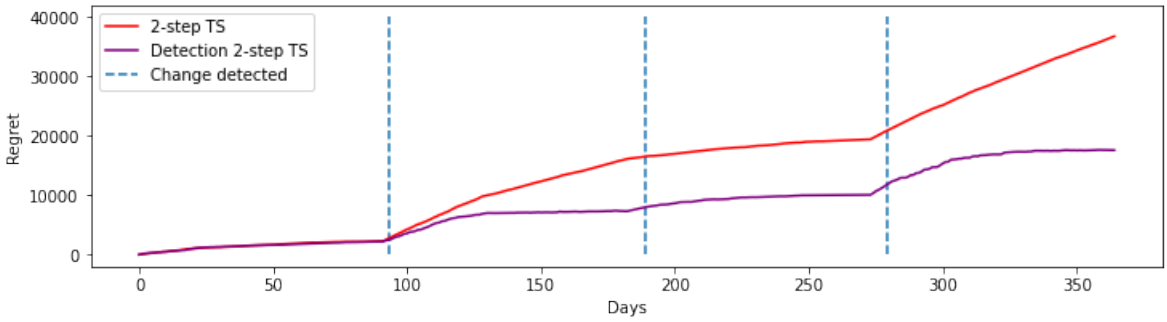


Figure 15: Cumulative Regret plot step 8



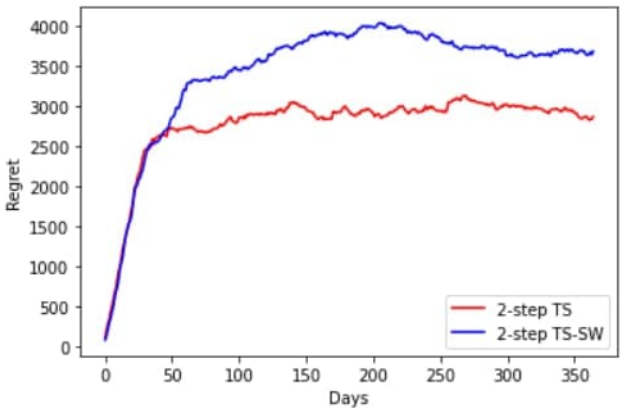
5 Conclusions

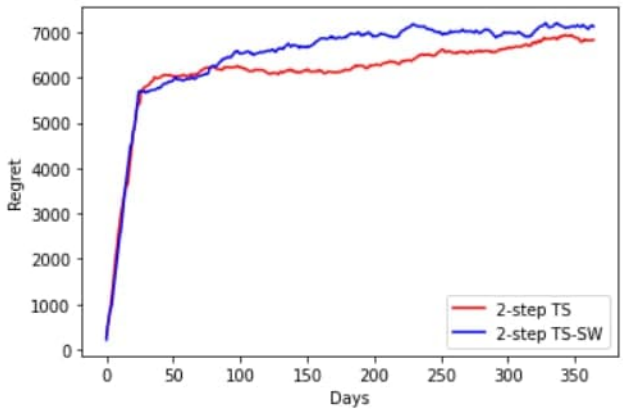
At the end of Step 8 we decided to wrap-up all the considerations about all the algorithms we used so far. Therefore, we decided to compare 2-Step TS Learner, 2-Step TS Learner SW and 2-Step TS Learner Change Detector in both Stationary and Non-Stationary Environment.

5.0.1 Comparison in Stationary Environment

2-Step TS Learner Change Detector in this case is exactly equal to the simple 2-Step TS Learner since no phase change will be detected and the parameters will never reset. Instead, it would be interesting to compare a sliding window approach with its stationary counterpart in a stationary environment: we expect the sliding window to keep the model simple and biased towards the last observations, hence having worse performances on a stationary environment. In the following plots, there are 2 cases. In the first one the sliding window size is 45 days, whereas in the second one it is 90 days.
Since there are 25 price arms in 2-Step TS Learner SW, it already takes 25 days to try out all the prices for price learner and if we also think about the matching part, our 2-Step TS Learner SW cannot possibly learn the correct prices and matching well enough in 45 days. And as soon as day day 45 ends, it will always have to delete previous days forgetting precious exploration trials. So, given that the window size is 45 days, the version with sliding window is considerably worse than the version with no sliding window.
For the case where the sliding window is 91 days, however, 2-Step TS Learner SW performs better than previous case and it is almost as good the version with no sliding window. We could say that in 91 days we have a better chance to learn the prices and matching by having more time to explore. Though, we still have to explore from time to time and lose our good rewards. So it is still worse than the version with no sliding window.

Below, you can see the regrets, for the case when the window size is 45 days and 90 days, respectively.





5.0.2 Comparison in Non-Stationary Environment

Using a non-stationary environment we expect a better performance from 2-Step TS Learner Change Detector and 2-Step TS Learner SW, while the 2-Step TS Learner should fall behind.

In this last scenario the change detector allows the learner to completely erase the learner's knowledge on the previous phase. This is in theory a good thing, as all data coming from previous phases is outdated, but in practice causes the learner to be retrained from scratch. In the end, the SWTS is more stable, even though having a worse cumulative reward, while the Change-Detector TS trades better rewards for a higher variance in its revenue curve.

Figure 16: Cumulative Revenue plot comparison

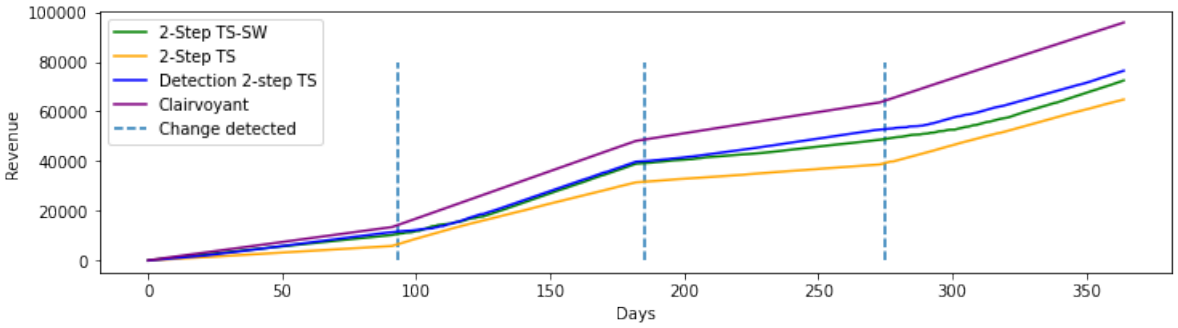


Figure 17: Revenue plot comparison

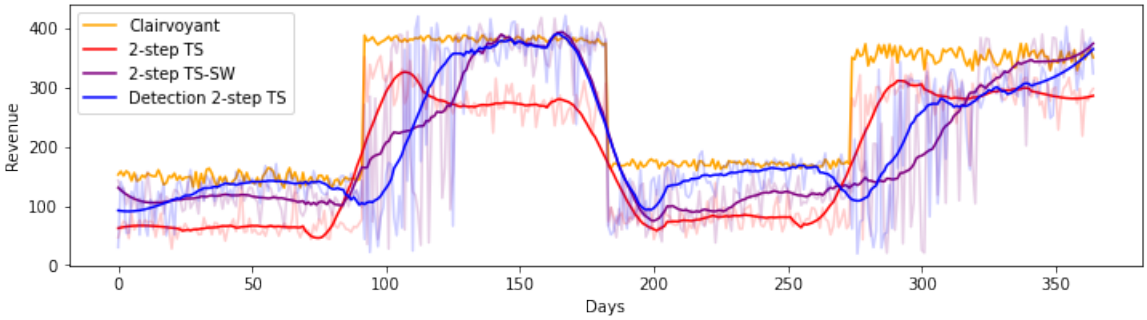


Figure 18: Cumulative Regret plot comparison