# Data Intelligence Applications
# Pricing and Matching

Ivan Cavadini (941927)
Simone Marforio (944320)
Nicolò Molinari (942404)

2020/2021

POLITECNICO
MILANO 1863

# Contents

# Introduction

## Scenario

Consider the scenario in which a shop has a number of promo codes to incentivize the customers that buy an item to buy a different item. The customers can belong to different classes and the promo codes can provide different discounts.

## Environment

Imagine two items (referred to as first and second items; for each item we have an infinite number of units) and four customers' classes. The daily number of customers of each class is described by a potentially different (truncated) Gaussian probability distribution. Each class is also associated with a potentially different conversion rate returning the probability that the user will buy the first item at a given price.

Once a buyer has bought the item, she/he can decide to buy the second item that can be or not promoted. There are four different promos P0, P1, P2, P3, each corresponding to a different level of discount. P0 corresponds to no discount. Given the total number of customers, the business unit of the shop decides the number of promos as a fraction of the total number of the daily customers and is fixed (use two different settings in your experiments that you are free to choose). Each customers' class is also associated with a potentially different conversion rate returning the probability that the user will buy the second item at a given price after she/he has bought the first. The promos will affect the conversion rate as they actually reduce the price. Every price available is associated with a margin obtained by the sale that is known beforehand. This holds both for the first and the second item. The conversion rates will change during time according to some phases due to, e.g., seasonality.

## Steps

1. Provide a mathematical formulation of the problem in 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.

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.

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.

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.

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.

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.

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

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

## Context Modeling

We have considered a ski shop that sells racing skis as first item and racing ski helmets as second item. The optimization problem have a time horizon of one year, splitted in three seasons that change the conversion rates of the two items. Customers are splitted into four different categories that define their purchasing behavior (conversion rates), according to the season and the price of the item.

| | | |
|---|---|---|
| Items | Racing Skis<br>Racing Ski Helmet | Professional racing skis<br>Professional racing skis helmet |
| Customer categories | Sport addicted<br>Gifter<br>Worried<br><br>Amateur | Who loves and practices ski frequently<br>Who wants to give away the both items<br>Who pays a lot of attention to the price of the items<br>Who sometimes practices ski |
| Seasons | Spring-Summer<br><br>Autumn<br><br>Winter | Buyers are not tempted to spend a lot, ski season is far away<br>Buyers are willing to spend in anticipation of the arrival of the ski season<br>Ski season has begun, those who have not yet bought the equipment have hurried to buy it so as not to waste the season |

## Assumption

- Seasonality is taken into accout only for the $7^{th}$, $8^{th}$ requests, while for all the other, the seasonality of the products is not considered and the conversion rates remain static. For this requests the default season is the first one, in our context called Spring.
- In our mathematical formulation, for the total reward maximization problem, we consider the production cost of both the items equal to zero.
- For the first step the objective of promo assignment is to find the best values for the fractions of clients of the various classes that receice a specific promo code. In the next steps, instead, we use online learning algorithm to find the best combination for the assignment promo code - class.

# Formal Model

## Variables definition

- $i$= user category
- $j$ = promotional discount: $\mathrm{P}_0 = 0\%$, $\mathrm{P}_1 = 10\%$, $\mathrm{P}_2 = 20\%$, $\mathrm{P}_3 = 30\%$
- $p1$ = full price of the first item (*Racing skis*)

- $p2$ = full price of second item (*Racing ski helmet*)
- $p2_j$ = price of the *Racing ski helmet* when applied the promo $j$
- $c1$ = production cost of *Racing skis* $= 0$
- $c2$ = production cost of *Racing ski helmet* $= 0$
- $q1_i(p1)$ = conversion rate for user category $i$, for *Racing skis* sold at the price $p1$
- $q2_i(p2)$ = conversion rate for user category $i$, for *Racing ski helmet* sold at price the $p2$
- $s_{ji}(p2)$ = discounted price of *Racing ski helmet*, for user category $i$, according to promo discount $j$
- $d_{ij}$ = amount of promo $j$ distributed to user category $i$
- $d_{max}$ = maximum number of promos to be to distributed ($\#\mathrm{P}_1 + \#\mathrm{P}_2 + \#\mathrm{P}_3$)
- $avgCustomer_i$ = average number of customers for category $i$

## Formulation of elaborated variables

- $p1 * q1_i(p1) * avgCustomer_i$ = revenue for the sale of *Racing skis* at price $p1$ to user category $i$
- $s_{ji}(p2) * q2_i(s_{ji}(p2)) * d_{ij} * avgCustomer_i$ = revenue for the sale of *Racing ski helmet* at the discounted price $p2$, according to the user-promo assignement
- $(p1 * q1_i(p1) - c1 * q1_i(p1)) * avgCustomer_i$ = revenue for the sale of *Racing skis* taking into account the production cost $c1$
- $(q2_i(p2) * (s_{ji}(p2) * q2_i(s_{ji}(p2)) * d_{ij} - q2_i(s_{ji}(p2))) * c2) * avgCustomer_i$ = revenue for the sale of *Racing ski helmet* taking into account the production cost $c2$

## Objective Function

We have a maximization problem with the following objective function:

$$\max(\sum_{i=0,j=0}^{i=4,j=4}[(p1*q1_i(p1)-c1*q1_i(p1)+q2_i(p2)(s_{ji}(p2)*q2_i(s_{ji}(p2))*d_{ij}-q2_i(s_{ji}(p2))*c2))*avgCustomer_i])$$

**s.t:** $\forall j > 0 : [\sum_{i=0}^{i=4} d_{ij}] = d_{max}$

We have fixed the full prices of the two items: $p1$, $p2$. We retrieve the discounted prices of $p2$, applying the promos $j$.
We know: the average number of customers per category $i$, $avgCustomer_i$, the conversion rate for both products ($q1_i(p1)$, $q2_i(p2)$) and the maximum number of promos to distribute ($d_{max}$).
As assumption the production costs of the two items is zero ($c1 = 0$, $c2 = 0$).
It is possible to retrieve the total revenue for *Racing skis* as the product between the full price of the first item, the conversion rate for the considered user category and the average number of customers for that category: $(p1 * q1_i(p1) * avgCustomer_i)$.
For the second item the calculation of the reward is the same except for the fact that the product is buyed only if also the first one is purchased (so we multiply also the conversion rate of the first item) and the considered price have to be discounted according to the assigned promotion.

The solution of our optimization problem consists in the distribution of the fraction of promo codes among the user categories.

## Offline problem - designed algorithm

We have to find the optimal solution in an offline manner (solve the maximization problem when all the parameters are known), considering the constraint that the shop uses a randomized approach, to assure that a fraction of a given customer category, gets a specified promotion, according to the optimal solution.
We have used an iterative approach to reach the optimal solution: we build a customer category-promotion (matching) matrix, which contains the mean expected rewards for every matching, calculated as the product between the conversion rate of the *Racing ski helmet* and its discounted price. The goal is to obtain, for each customer-promo matching, the fraction of customers that will receive this discount, in order to maximize the total reward.
We select the best reward for every class, for four times, retrieving, at each iteration, the four best combination of category-promotion and assigning an infinite weight to

the obtained sub-optimal matching.
Every matching is represented by a reward configuration that maximize the total reward, every iteration is weighted and represent a different goodnesses of the solution (the first is the best, the last is the worst).
Through the sub-optimal matchings, we have retrieved the fractions of different promos to assign to every customer categories, based on the proportional weight of the previous sub-optimal matching.
The proportions retrieved, are normalized category per category.

PESEUDOCODICE

1. Calculate the fraction of the customers per class that buy the first item through the conversion rate of the first item (at the defined price), as fractions of buyers: $firstItemBuyers[i] =$ $int(customerDaily[i] * conversionRateFirstElement(item1PriceFull, i))$ Once we know the number of customers that bought the first item, we aims to maximize the profit making them buy the second item.

2. Calculate the discounted price for the different promo codes: $discounted_price = [item2_price_full, item2_price_full * (1 - ctx.discount_promos[1]), item2_price_full * (1 - ctx.discount_promos[2]), item2_price_full * (1 - ctx.discount_promos[3])]$

3. Initialize the matching matrix where the rows are the user categories and the columns are the discount type. Cells are the weighted as $(conversionRateSecondItem * discountedPriceSecondItem * firstItemBuyers)$ of that class.

4. The matching is performed iterating over the matching matrix four times, retrieving the iteration matrix. Every iteration determine the optimal solution of the matching problem, which allow to maximize the profit. The iteration matrix save all these oprimal solutions.

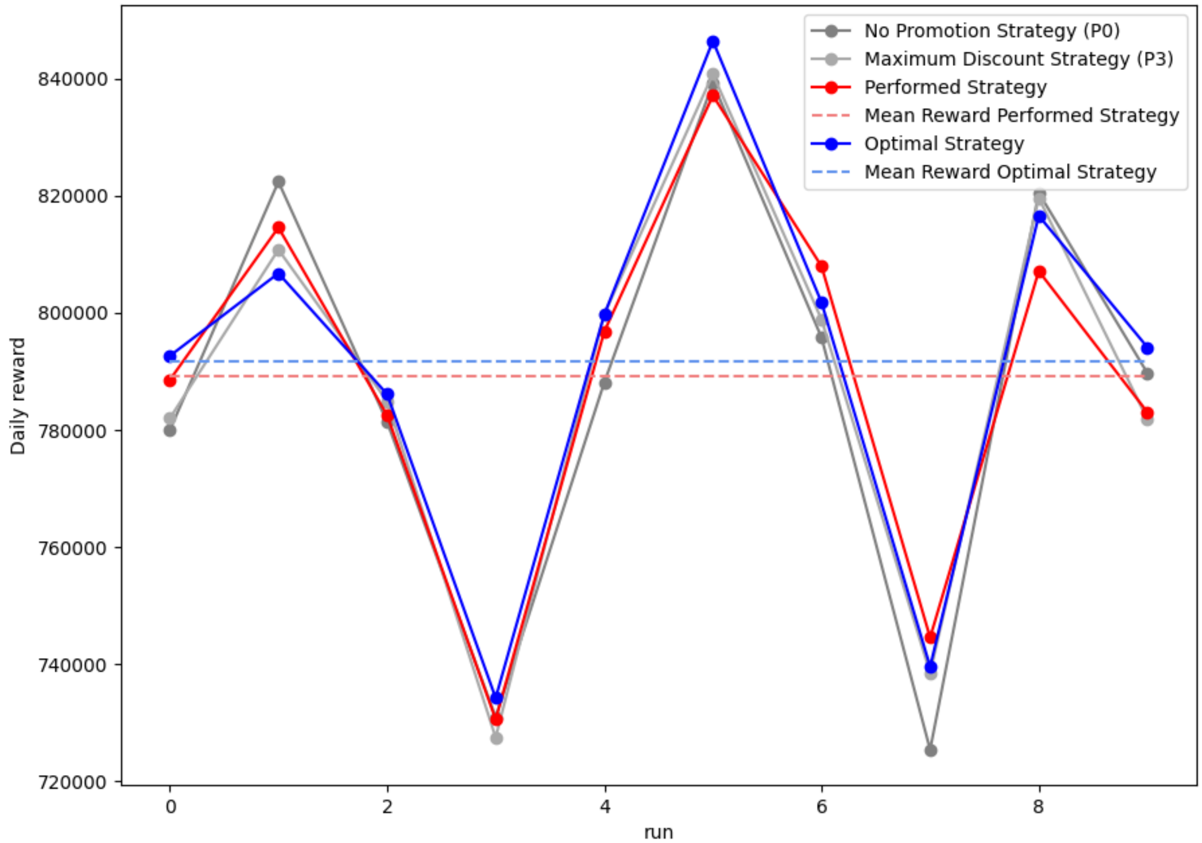
   for i in range(4):
   0.6cm rowInd,colInd = linearSumAssignment(matchingMatrix,maximize=True)
   0.6cm temp = np.zeros((4,4))
   0.6cmfor ind in range(0,len(rowInd)):
   1.2cmtemp[rowInd[ind],colInd[ind]] = matchingMatrix[rowInd[ind],colInd[ind]]
   1.2cmmatchingMatrix[rowInd[ind],colInd[ind]] = np.iinfo(np.int64).min
   0.6cmiterationMatrix.append(temp)

5. Compiling the class final distribution matrix:
   w = 1

for i in range(4):
0.6cmiterSum = np.sum(iterationMatrix[i])
0.6cmcoordinates = np.nonzero(iterationMatrix[i])
0.6cmfor idx in range(len(coordinates[0])):
1.2cmclassFinalDistribution[coordinates[0][idx], coordinates[1][idx]] = 1.2cm(100 *iterationMatrix[i][coordinates[0][idx], coordinates[1][idx]] / 1.2cm iterSum ) * w
0.6cmw = w/2

for i in range(0,4):
0.6cmsumPerClass=(np.sum(classFinalDistribution[i]))
0.6cmfor j in range(0,4):
1.2cmclassFinalDistribution[i,j] = (classFinalDistribution[i,j]*100/sumPerClass)/100

```
Optimal solution: probability distribution of promos per class (rows: class, col: promos)
[[0.12 0.28 0.52 0.08]
 [0.06 0.53 0.15 0.26]
 [0.52 0.07 0.28 0.14]
 [0.17 0.09 0.05 0.69]]
```



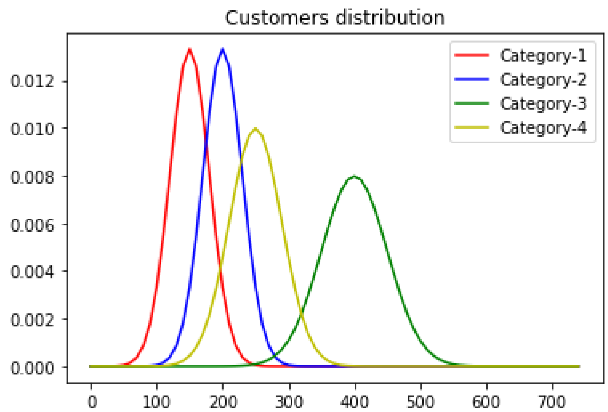

# Random Varaibles

For the online problem have identified and modeled the following radom variables

## Daily customers distribution

We modeled the daily customers distribution as a **Gaussian Distribution**, with a normalizing factor of 1000 daily customers. Average and variance of the Gaussian distribution are reported in the table below.

| Customer Category | Average | Variance |
|---|---|---|
| Sport addicted | 0.15 | 0.03 |
| Gifter | 0.20 | 0.03 |
| Worried | 0.40 | 0.05 |
| Amateur | 0.25 | 0.04 |

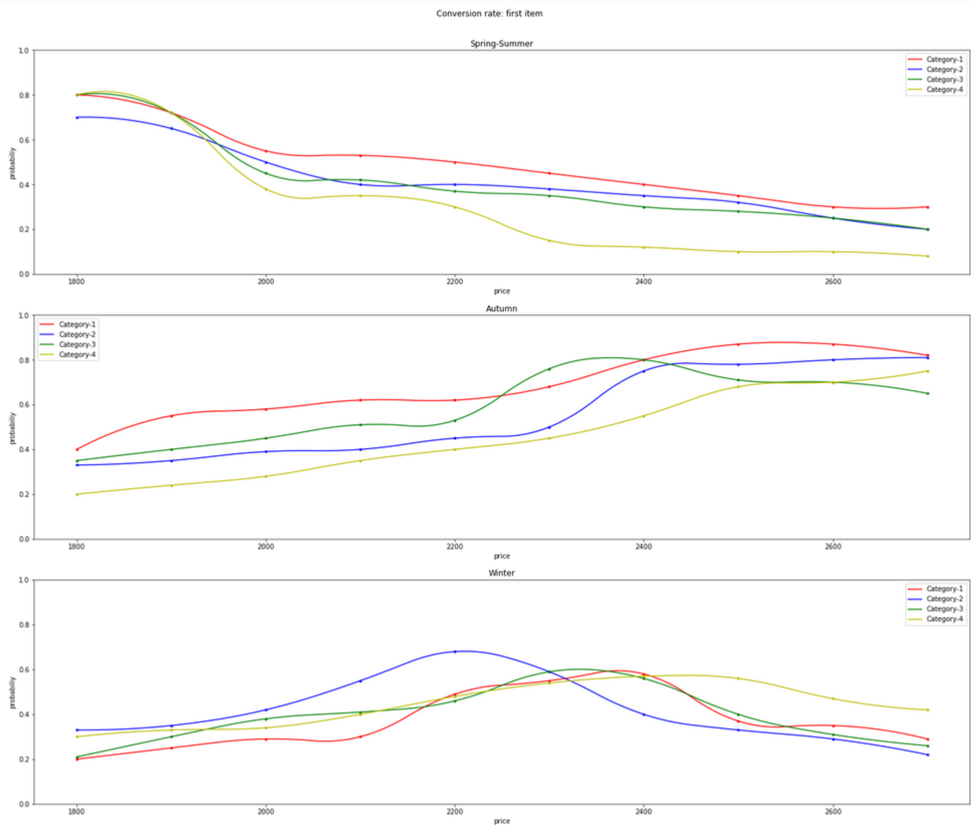


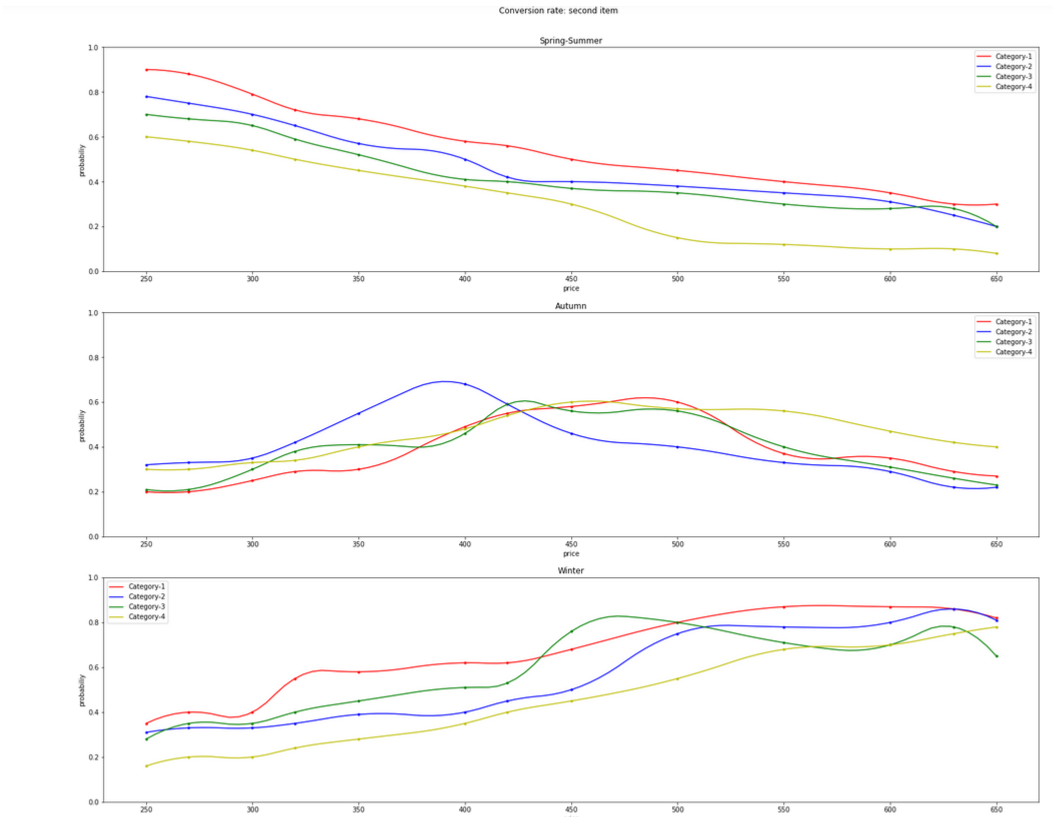

## Conversions rates

The odds of buying one item are modelled with a **Bernulli Distribution** different for every user category and item price.

- Probability of buying *Racing skis*: Bernoulli $\tilde{0}$,1
- Probability of buying *Racing ski helmet*: Bernoulli $\tilde{0}$,1

The following graphs are the demand curves of the two items: the first three graphs are the demand curves of the first item, associated to the three seasonality (Spring-Summer, Autumn, Winter), while the last three graphs are the demand curves of the second item with its respective seasonality.


Conversion rate: first item
Spring-Summer
Category-1
Category-2
Category-3
Category-4
probability
price
Autumn
Winter
1.0
0.8
0.6
0.4
0.2
0.0
1800
2000
2200
2400
2600




## Online Approach

Our general approach for the online problem is to simulate, day by day, the shop, generating the customers and emulating their behavior, collecting the results and, according to the considered scenario and constraints, provide an optimal solution that maximize the reward.

Every day we retrieve the daily customer distribution per class using the previously presented random variable that model the daily customer distribution. Randomly we simulate the entry of a new customer, of which we know the category of belonging, in the shop. With an online approach we select the best price to be prosed to the client, in order to maximize the reward. The purchase is simulated with the previously presented random variable with a Bernulli distribution. The second item is proposed to the client only if the first has been purchased. The price at which it is proposed is retrieved with an online matching approach that try to sugget which is the best discout to apply to the user in oreder to maximize the reward.

This procedure is repeated for all clients during the entire time horizon of 365 days.

# Online pricing for first item

**Submission :** *Consider the case in which the assignment of promos is fixed and the price of the second itm 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.*

The request is to learn the optimal price of the first item, comparing the adoption of a Thompson-sampling approach and an Upper-Confidence Bound approach

## Basic knowledge

The described problem is a combinatorial bandit problem, which is a decision making problem in where decision maker selects one single arm in each round, and observes a realization of the corresponding unknown reward distribution. Each decision is based on past observed rewards. The objective is to maximize the expected cumulative reward over some time horizon by balancing exploitation and exploration. We can solve it, through an online algorithm, where only for the feasible solutions we will have precise estimations. Thomspon-Sampling algorithm (TS) and Upper-Confidence Bound algorithm (UCB1) are well known algorithm used to solve combinatorial bandits problem.

**Upper-Confidence Bound (UCB1)**

- Every arm is associated with an upper confidence bound
- At every round, the arm with the highest upper confidence bound is chosen
- After having observed the realization of the reward of the arm, the upper confidence bound is updated

Notation:

- $t$ time
- $A$ set of arms

- $a$ arm
- $a_t$ arm played at time $t$
- $a^*$ optimal arm
- $X_a$ random variable (bernoulli) associated to arm a
- $\mu_a$ expected value of random variable $X_a$
- $x_{a,t}$ realization of random variable $X_a$ at time $t$
- $x_a$ realizations of $X_a$
- $\bar{x}_a$ empirical mean of $x_a$
- $n_a(t)$ number of samples of arm $a$ at time $t$

Pseudocode

1. Play once each arm $a \in A$
2. At every time $t$ play arm $a$ such that:
   $a_t \leftarrow \arg\max_a \left\{ \left[ \bar{x}_a + \sqrt{\frac{2log(t)}{n_a(t-1)}} \right] \times a \right\}$

**Thompson Sampling (TS)**

- For every arm, we have a prior on its expected value
- In the case the arms' rewards are Bernoulli distribution, the priors are Beta distributions
- Notice that, with the opportune parameters, a Beta distribution is a uniform distribution
- For every arm, we draw a sample according to the corresponding Beta
- We choose the arm with the best sample
- We update the Beta distribution of the chosen arm according the observed realization

Notation (in addition to classical UCB):

- $\mathbb{P}(\mu_a = \theta_a)$ prior of the expected value of $X_a$
- $\theta_a$ variable of $\mathbb{P}(\mu_a = \theta_a)$
- $(\alpha_{a_t}, \beta_{a_t})$ parameters of the beta distribution $P(\mu_a = \theta_a)$

Pseudocode

1. At every time $t$ for every arm $a$:
   $\tilde{\theta_a} \leftarrow Sample(\mathbb{P}(\mu_a = \theta_a))$
2. At every time $t$ play arm $a_t$ such that
   $a_t \leftarrow \arg\max_a \left\{\tilde{\theta_a} \times a\right\}$
3. Update beta distribution of arm $a_t$
   $(\alpha_{a_t}, \beta_{a_t}) \leftarrow (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t})$

## Strategy

In our solution we simulate a random arrival of the customers, through two different leaners, TS learner and a UCB learner, we extract from our candidates two prices and we emulate the purchase phase using both prices. In case the customer buys the first item, we propose the second one at a fixed price, computed according to the customer category. In this case we do not simulate the purchase phase, but we use the knowledge of the conversion rate of the second item to compute the final customer reward.

### Implementation

- No seasonality, conversion rate do no change
- Number of customers per class is known
- Candidates for the *Racing Skis* are: {2260.0,1910.0,2130.0, 2010.0, 2340.0}
- Conversion rate associated with the first item is not known
- Basic price of the Racing Ski Helmet is fixed to 630.0
- Promotion assignment is fixed, according to the results of our offline solution:

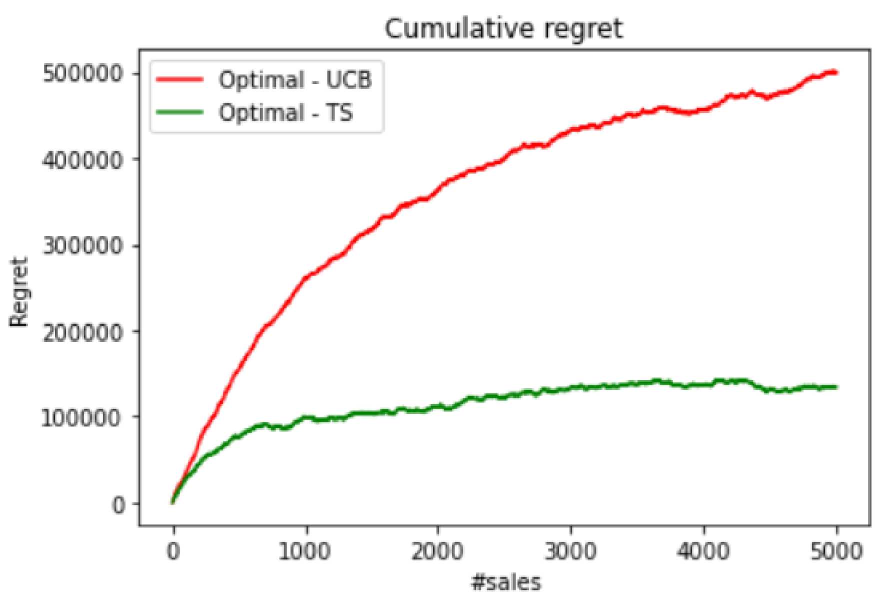
| User category | Assigned promotion | Racing Ski Helmet price |
|---|---|---|
| Sport Addicted | $P_2$ : 20% | 504.0 |
| Gifter | $P_1$ : 10% | 567.0 |
| Amateur | $P_0$ : 0% | 630.0 |
| Worried | $P_3$ : 30% | 441.0 |

- Conversion rate associated with the second item is known

Both UCB and TS learner expect as learnign parameter a binomial value, while our reward is value composed by the sold of the first item plus the eventual sold of the second item. In order to normilize the reward to be passed learner, we divide the customer reward by the maximum achievable reward.

**Optimal strategy** In order to compute a regret, the simulated rewards are compared with an optimal solution that, accordingly to the conversion rates for the first items, is to offer the *Racing Skis* at the lower price (1910.0 according to our candidates prices).

## Results




Days: 10
Experiments number: 10

Both UCB and TS strategy converge on 1910.0
*We decide to plot the regret of the first 5000 clients, since plotting the results of the entire time horizon made the plot unreadable*

### Considerations

As we can observe in the plot, both approach converge to a stable solution, however Thompson Sampling approach performs better than a UCB approach. Infact Thompson Sampling is faster to find the best price for the first item than UCB and this allow to have a lower regret.

## Online pricing for first item with purchase simulation

**Submission :** *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.*

The goal is the same of the previous problem: we have to learn the optimal price for the first item comparing TS and UCB approach. In addition, this time, we have no information about the conversion rate for the second item.

### Strategy

The script that we have implemented is similar to the previous one. We simulate a random arrival of the customers, proposing the *Racing Skis* to them at the prices suggested by the two learners (UCB and TS) and simulating the purchase of it. In case the customer buys the first item, instead of mathematically retrieve the reward of the second item using the conversion rate, we simulate the purchase of it proposing the *Racing Ski Helmet* at the disconted price accordingly to the user category. We calculate the total reward and we update the learners with the results, as the prevoius submission, normilizing the reward.

#### Implementation

- No seasonality, conversion rate do no change
- The number of customers per class is not known
- The conversion rates associated to the second item are not known

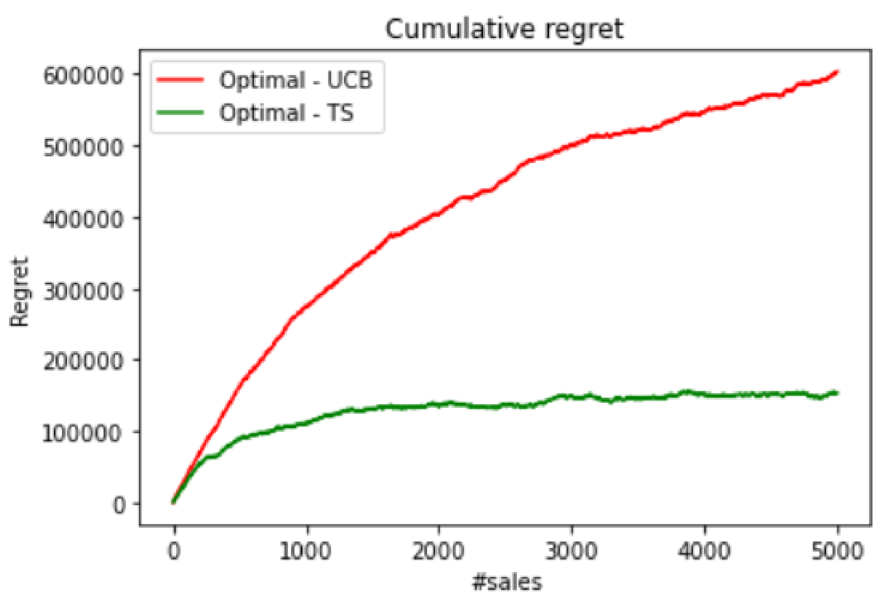
- Candidates for the *Racing Skis* are: {2260.0,1910.0,2130.0, 2010.0, 2340.0}
- Conversion rate associated with the first item is not known
- Basic price of the Racing Ski Helmet is fixed to 630.0
- Promotion assignment is fixed, according to the results of our offline solution:

| User category | Assigned promotion | Racing Ski Helmet price |
|---|---|---|
| Sport Addicted | $P_2$ : 20% | 504.0 |
| Gifter | $P_1$ : 10% | 567.0 |
| Amateur | $P_0$ : 0% | 630.0 |
| Worried | $P_3$ : 30% | 441.0 |

- Conversion rate associated with the second item is not known

**Optimal strategy** As in the previous submission, the regret is calculated comparing the online approach with an optimal solution. Again, the optimal solution, accordingly to the conversion rate, is to offer the first item at the lower price (1910.0 according to our candidates prices).

## Results



Days :10
Experiments number: 10
Both UCB and TS strategy converge on 1910.0
*We have decided to plot the regret of the first 5000 clients, since plotting the results of the entire time horizon made the plot unreadable*

### Considerations

The result shows that a Thompson Sampling approach performs better than a UCB approach, as in the previous problem. The regret curves of both the algorithm are slightly higher than the previous scenario, because, unlike the previous case, we do not know the conversion rates associated to the second item so we have a less precise value that will feed the two leaner, increasing the inaccracy of the estimations.

## Matching problem: promo assignment

**Submission :** *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.*

The submission requires to learn the assignment of the promotions to the diffferent customer category. This time we are dealing with a matching problem.

### Basic knowledge

DIRE CHE BANDIT PROBLEM è e RIVEDERE SE QUELLI PRECEDENTI SONO GIUSTI. AGGIUNGERE UN PO' DI TEORIA

A Matching Problem can be modeled as a graph $G = (N, A)$, where $N$ is a set of node and $A$ is a set of arc. A matching $M$ is a subset of $A$ such that each node is connected to at most one node with at most one arc of $M$. The problem can be seen as a maximization problem where the objective is to determine the matching of maximum cardinality.

**UCB Matching** We can design a bandit algorithom that solve the mathing problem. A classical UCB approach is combined with a linear sum assignement algorithm, a well known algorithm that given a bipartite graph solve the matching problem. The learner will retrieve a the subset of arc that correspond to the optimal

matching. At $t$, play a superarm $a_t$ such that:
$a_t \leftarrow \arg\max_{\mathbf{a}\in M} \left\{ \sum_{a\in\mathbf{a}} \bar{x}_{a,t} + \sqrt{\frac{2log(t)}{n_a(t-1)}} \right\}$
where $M$ is the set of matches.

**Promo-Category UCB Matching** We have slightly modifying the previously implemented UCB Matching.
We introduce two matrixes, with the same shape of the promotion-category graph, used for the maximization problem, representing the matching between a specific customer category and a specific promo. The first matrix represent the total collected rewards for that specific assignement, the second is used as support and contains the number of occurences that the assignement has been chosen (a customer of that category to which it was proposed the second item with that promo). Those matrix are updated by the reward obtained by the currently served customer and used to calculate the average reward of each possible assignment. The average value are used to feed the default UCB Matching learner in order to update the confidence bound. An initialization phase is needed in order to explore and learn the average reward for all the possible configurations. For this reason we introduce a staring dalay in which we explore all possible configurations in an exaustive way and store the rewards. In this phase the learner do not compute the optimal solution for the matching problem, but retrieves a known matching that change at each iteration. As for the other learners, the rewards are normalized dividing the actual reward by the maximum possible reward.

## Strategy

As always we simulate the randomly arrival of the customers and the purchase at a fixed price of the first item. In case the customer purchase the *Racing Skis*, we ask to the learner (our custom UCB matching learner) to retrieve the optimas assignemnt promotions-category. According to the user category, and the pulled assignment promo-category, we compute the discounted price for the second item and we propose it to the customer. In important to note that the learner provides us a set of arms, but we use only the assignement that involve the category of the currently served user. The user reward is calculated as the sum of the reward obtained by the sold of the first item plus the eventually sold of the sencond. However, since our goal it to lorne the matching for the fromo-category of the second item, and the price of the *Racing Skis* is fixed, we feed the matching learner with the reward obtained by the sold of the *Racing Ski Helmet*.

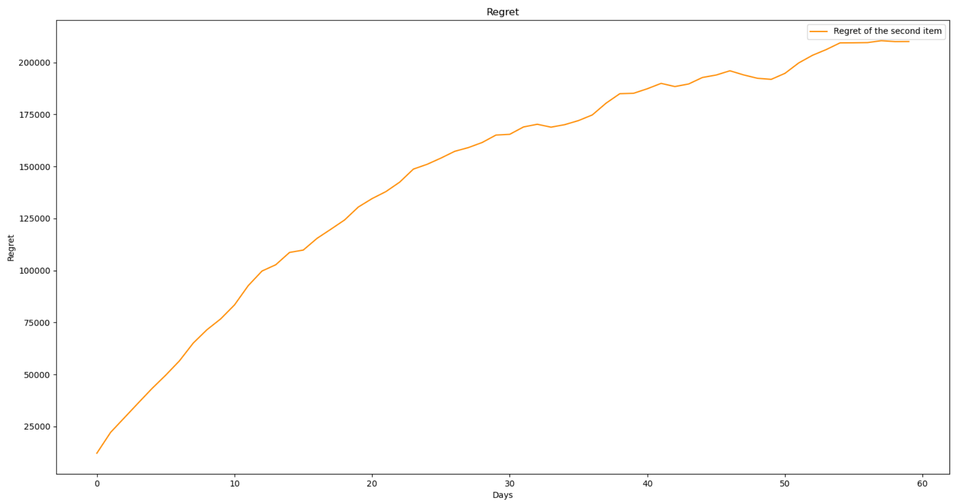
**Implementation**

- No seasonality, conversion rate do no change
- Price of the *Racing Skis* is fixed to 1980.0
- Conversion rate associated with the first item is not known
- Basic price of the *Racing Ski Helmet* is fixed to 630.0
- Conversion rate associated with the second item is not known
- Optimal promotion-category assignment need to be learnt

**Optimal strategy** The optimal strategy, used to compute the regret, is calculated in an offline manner, calculating for each possible promotion-category the expected reward, obtained multiplyng the conversion rate of the category with the discounted price. The obtained matrix of the expected rewards is then evaluated with a Linear Sum Assignement algorithm that maximize the total expected reward, retrieving the optimal assigment. The obtained optimal solution is:

| User category | Assigned promotion | Racing Ski Helmet price |
|---|---|---|
| Sport Addicted | $P_2$ : 20% | 504.0 |
| Gifter | $P_1$ : 10% | 567.0 |
| Amateur | $P_0$ : 0% | 630.0 |
| Worried | $P_3$ : 30% | 441.0 |

## Results




Days: 60
Experiments number: 5
Starting delay of the Promo-Category UCB Matching: 1000 clients
UCB Matching at the end converge to the optimal solution

## Considerations

We can observe that the UCB Matching algorithm has a linear increase on the cumulative regret for the first thirty days, but after that, it becomes more and more stable on the optimal matching, and the cumulative regret does not increase so much.

# Pricing and Matching problem

**Submission :** *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.*

The problem requires to find the optimal prices for the two items and to optimize the assignment of the promos, in the scenario where all the parameters need to be learnt.

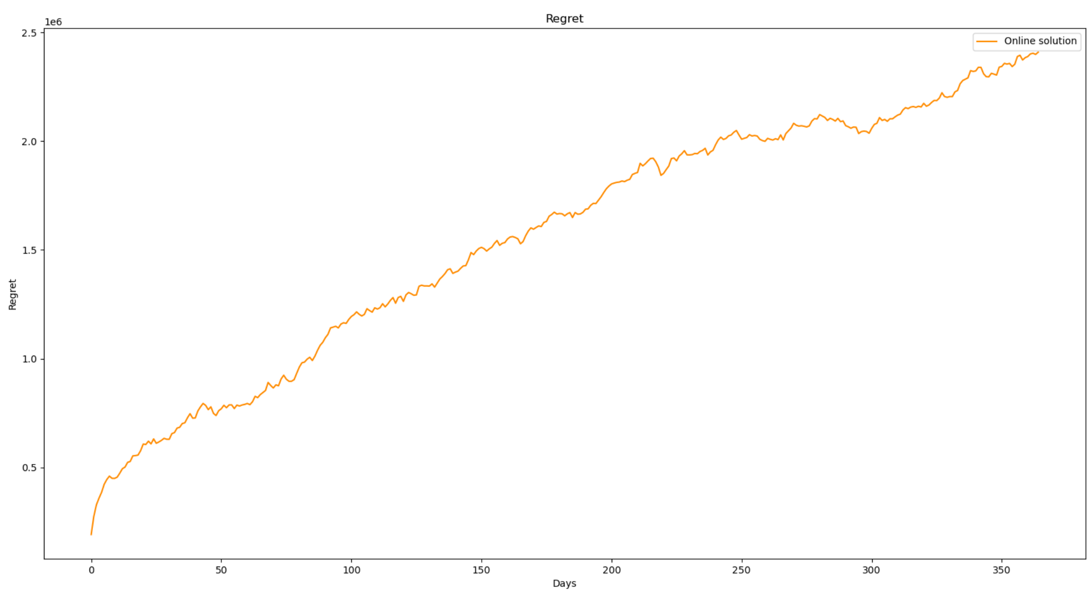
## Strategy

We simulate the random arrival of the customers. For every customer we retrieve a superarm from the Thompson sampling learner. A superarm is associated to the couple of the candidates prices of the two items. We evaluate the purchase of the first item at the suggested price. In case the customer buys the first item, we pull the matching from the UCB learner, and we do the same procedure of purchase for the second item at the proposal discounted price. Before the arrival of a new customer, we update the TS learner with the entire reward given by the two items, and the UCB learner with the reward of the second item. We have used the same two matrixes in the same way of the previous request.

### Implementation

### Optimal strategy

## Results



## Considerations

We can notice that the learners take more time to learn the optimal solutions both for pricing and matching. The cumulative regret is increasing quite linearly until the day 200th, after that, they start to stabilize on the optimal solutions. The cumulative regret still be jagged, because both the learner can pull random arms with some probabilty, this has impact on the curve.

# Seasonal Pricing and Matching problem: Sliding Window

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

The goal is the same of the previous problem, but in this case the conversion rates are not stationary and is required to use a sliding-window approach.

## Basic knowledge

**Sliding-Window Thompson Sampling (SW-TS)** We use a sliding window of length $\tau \in N$ such that the algorithm, at every round $t$, takes into account only the rewards obtained in the last $\tau$ rounds. Based on these realizations, we apply a TS-based algorithm to decide which is the arm to pull in the next round. In particular, the expected value of each arm is coupled with a posterior distribution from which we draw samples, and the arm with the highest value is the next arm to play.

**Pseudocode:**

1. At every time $t$ for every arm $a$:
$\tilde{\theta_a} \leftarrow Sample(\mathbb{P}(\mu_a = \theta_a))$

2. At every time $t$ play arm $a_t$ such that
$a_t \leftarrow \arg\max_{a \in A} \left\{ \tilde{\theta_a} \right\}$

3. Update beta distribution of arm $a_t$
if $t \leq \tau$: $(\alpha_{a_t}, \beta_{a_t}) \leftarrow (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t})$
if $\tau < t$: $(\alpha_{a_t}, \beta_{a_t}) \leftarrow \max\left\{(1,1), (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t}) - (x_{a_{t-\tau},t-\tau}, 1 - x_{a_{t-\tau},t-\tau})\right\}$

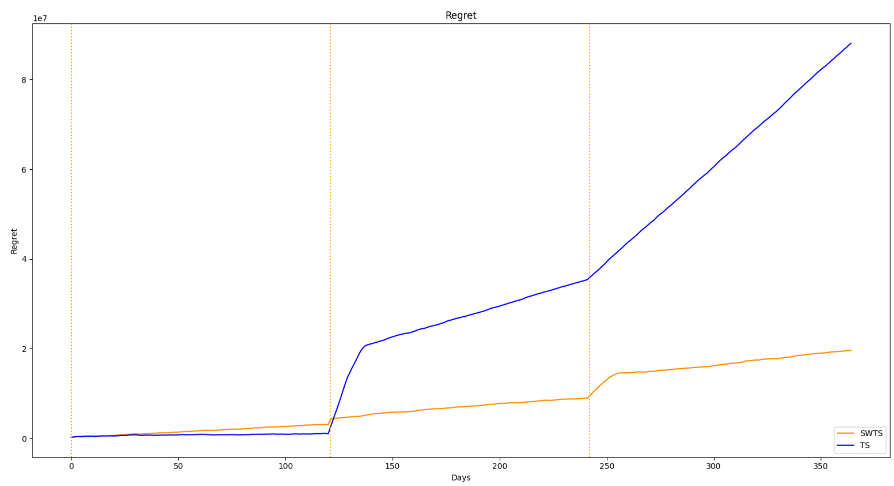
## Strategy

In a non-stationary environment the conversion rates are not stationary, so they change due to the seasonalities. We exploit the sliding-window approach combined with the Thompson sampling algorithm (SW-TS) to solve this problem. It is an algorithm that exploits a Sliding-Window approach to forget past information during the learning, which could provide a bias to the estimation process.

### Implementation

### Optimal strategy

## Results




## Considerations

We can observe that in the first season the SW-TS algorithm has an higher regret than the TS algorithm, because the first has a number of samples that is equal to the dimension of the window, instead the second collects all the samples. The SW-TS has a better performance from the second season, due to the sliding-window approach, it stabilizes faster than the TS, and at the end, the cumulative regret for the SW-TS is about 4 times less than the TS.

# Seasonal Pricing and Matching problem: Change Detection

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

The goal is the same of the previous problem, but instead of a sliding-window approach, is required to adopt a change-detection test approach.

## Basic knowledge

**Change Detection (CUSUM)** The first $M$ valid samples are used to produce the reference point.

Empirical mean of arm $a$ over the first $M$ valid samples $\bar{X}_a^0$

From the $M+1-th$ valid sample on, we check whether there is a change

Positive deviation from the reference point at $t$ $\quad s_a^+(t) = (x_a(t) - \bar{X}_a^0(t)) - \epsilon$

Negative deviation from the reference point at $t$ $\quad s_a^-(t) = -(x_a(t) - \bar{X}_a^0(t)) - \epsilon$

Cumulative positive deviation from the reference point at $t$ $\quad g_a^+(t) = \max\{0, g_a^+(t-1) + s_a^+(t)\}$

Cumulative negative deviation from the reference point at $t$ $\quad g_a^-(t) = \max\{0, g_a^-(t-1) + s_a^-(t)\}$

We have a change if $\quad g_a^-(t) > h$ or $g_a^+(t) > h$

**CD-UCB Pseudocode:**

1. Initialize $\tau_a = 0$ for $a \in A$
2. For each time $t$:

$$a_t \leftarrow \arg\max_{a \in A} \left\{ \bar{x}_{a,\tau_a,t} + \sqrt{\frac{2log(n(t))}{n_a(\tau_a, t-1)}} \right\} \text{ with probability } 1-\alpha$$

$$a_t \leftarrow \text{random arm with probability } 1-\alpha$$

$n(t)$ is total number of valid samples

$\bar{x}_{a,\tau_a,t}$ is the empirical mean of arm $a$ over the last valid samples

$n_a(\tau_a, t-1)$ is the number of valid samples for arm $a$

3. Collect reward $r_t$

4. If $CD_a(r_\tau, ..., r_t) = 1$ then $\tau_a = t$ and restart $CD_a$

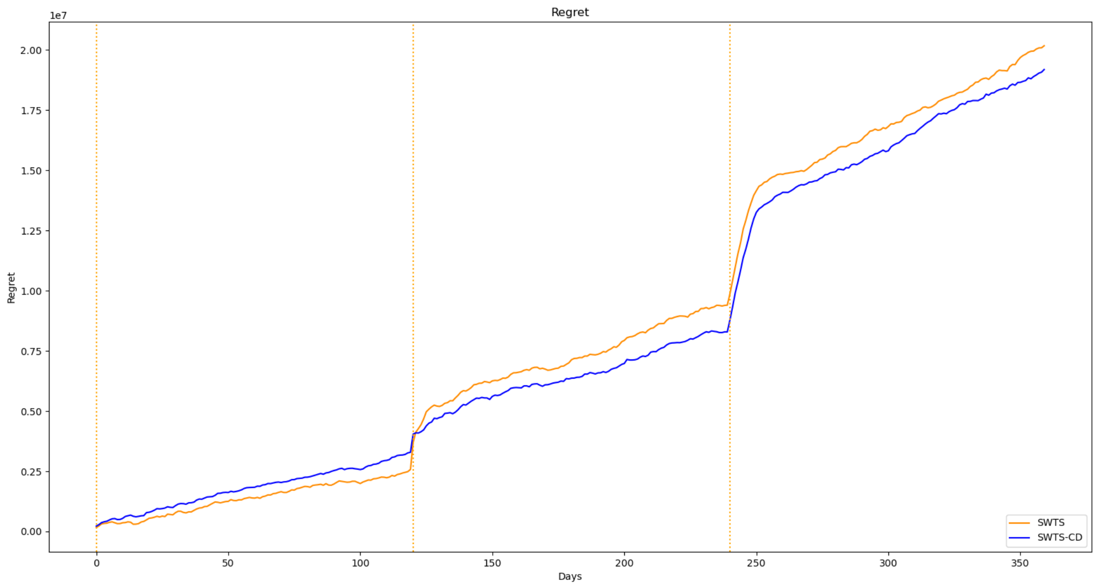
## Strategy

In order to solve this problem we have used a change detection (CUSUM) algorithm and a bandit algorithm (UCB). At each time $t$, the UCB outputs a decision $I_t \in K$, where $K$ is a set of arms, based on its past observations of the bandit environment. The environment generates the corresponding reward of arm $I_t$, which is observed by both the bandit algorithm and the change detection algorithm. The change detection algorithm monitors the distribution of each arm, and sends out a positive signal to restart the bandit algorithm once a breakpoint is detected.

### Implementation

### Optimal strategy

## Results




## Considerations

We can observe that the change-detection approach performs better than without it, this can be seen from the second season to the end. In the first season, the cumulative regret is higher with the change-detection approach, because it catch some false-positive detection, that can change the solution that it suggests.

# Folder Structure