Data Intelligence Applications
Pricing and Matching

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

2020/2021

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 belong to different classes and the promo codes 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 Racing Ski Helmet	Professional racing skis Professional racing skis helmet
Customer categories	Sport addicted Gifter Worried Amateur	Who loves and practices ski frequently Who wants to give away both the items Who pays a lot of attention to the price of the items Who sometimes practices ski
Seasons	Spring-Summer Autumn Winter	Buyers are not tempted to spend a lot, ski season is far away Buyers are willing to spend more as the ski season approaches 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} and 8^{th} requests, while for all the others, 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 is called Spring-Summer.
- 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 category that receive a specific promotion. In the next steps, instead, we use online learning algorithm to find the best combination for the assignment promotion-category.

Step 1 - Formal Model

Variables definition

- i = user category
- j = promotional discount: $P_0 = 0\%$, $P_1 = 10\%$, $P_2 = 20\%$, $P_3 = 30\%$
- $p1$ = 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 the price $p2$
- $s_{ji}(p2)$ = 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 promo-category assignement (note that the dependence of the second item with the first is not taken into account in this formula)
- $(p1 * q1_i(p1) - c1 * q1_i(p1)) * avgCustomer_i$ = 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:

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

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

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 combinations 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. Then the retrieved proportions, are normalized category per category.

Pseudocode Below we schematize the previous described algorithm:

1. For both the items fix a prices called $item1PriceFull$ and $item2PriceFull$
2. Calculate the fraction of the customers per class that buy the first item through the conversion rate of the first item (at the predetermined 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.
3. Calculate the discounted price for the different promo codes: $discountedPrice = [item2PriceFull, item2PriceFull*(1-ctx.discountPromos[1]), item2PriceFull*(1-ctx.discountPromos[2]), item2PriceFull*(1-ctx.discountPromos[3])]$
4. Initialize the matching matrix where the rows are the user categories and the columns are the promotions. Cells are weighted as $(conversionRateSecondItem* discountedPriceSecondItem * firstItemBuyers)$ of that class.
5. The matching is using a Linear Sum Assignement algorithm and iterated four times, retrieving a matrix called iteration matrix. Every iteration determine the optimal solution of the matching problem which allow to maximize the profit, discarding the solution obtained in the iterations before. The iteration matrix saves all these optimal solutions.

```
for i in range(4):
  rowInd,colInd = linearSumAssignment(matchingMatrix,maximize=True)
  temp = np.zeros((4,4))
  for ind in range(0,len(rowInd)):
    temp[rowInd[ind],colInd[ind]] = matchingMatrix[rowInd[ind],colInd[ind]]
        \matchingMatrix[rowInd[ind],colInd[ind]] = np.iinfo(np.int64).min
  iterationMatrix.append(temp)
```

6. Computing the final matrix thta holds all the distributions:

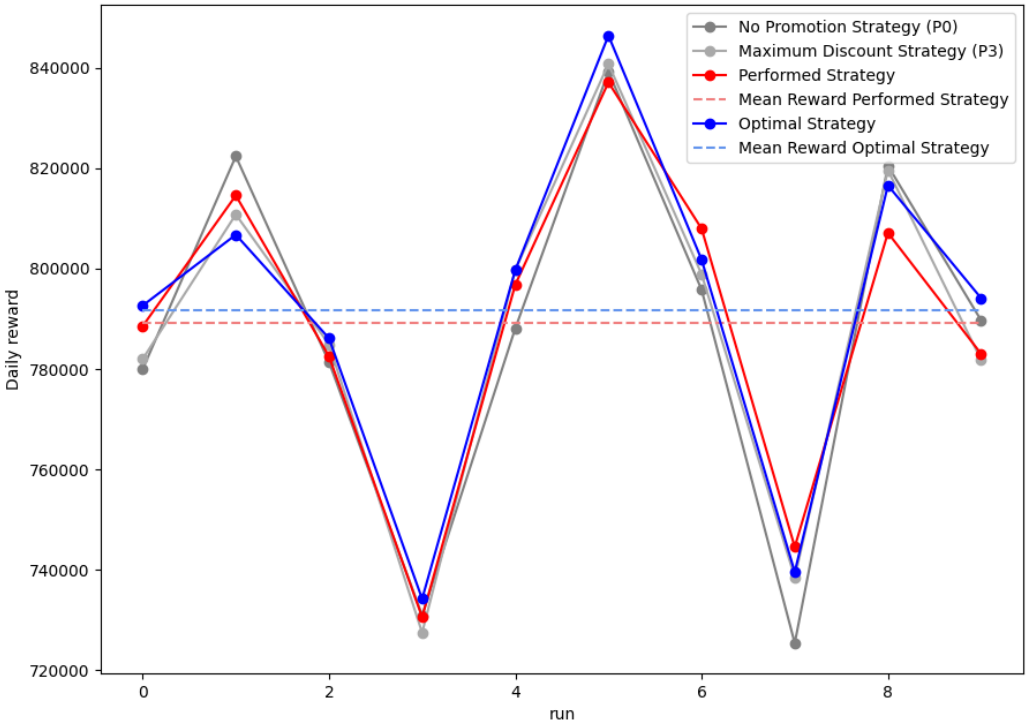
```
w = 1
for i in range(4):
  iterSum = np.sum(iterationMatrix[i])
  coordinates = np.nonzero(iterationMatrix[i])
  for idx in range(len(coordinates[0])):
    classFinalDistribution[coordinates[0][idx], coordinates[1][idx]] =
      (100 *iterationMatrix[i][coordinates[0][idx], coordinates[1][idx]] /iterSum ) * w
  w = w/2

for i in range(0,4):
  sumPerClass=(np.sum(classFinalDistribution[i]))
  for j in range(0,4):
    classFinalDistribution[i,j] = (classFinalDistribution[i,j]*100/sumPerClass)/100
```

Implementation For completeness we implemented the previously described algorithm (*n1.py*) and the results of the running are the following distributions matrix of promotion-category:

```
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]]
```

In terms of rewards the results are the following:



Step 2 - Random Variables

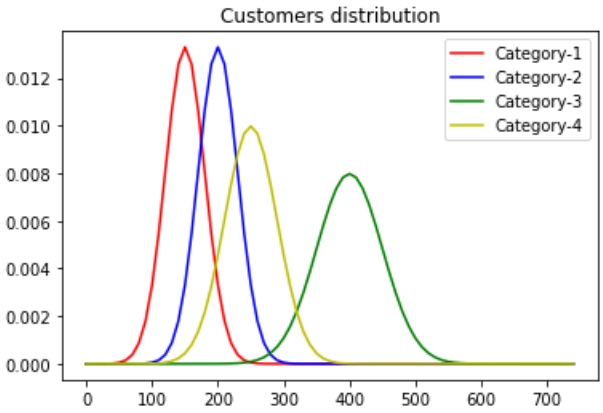
For the online problem we have identified and modeled the following radom variables.

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

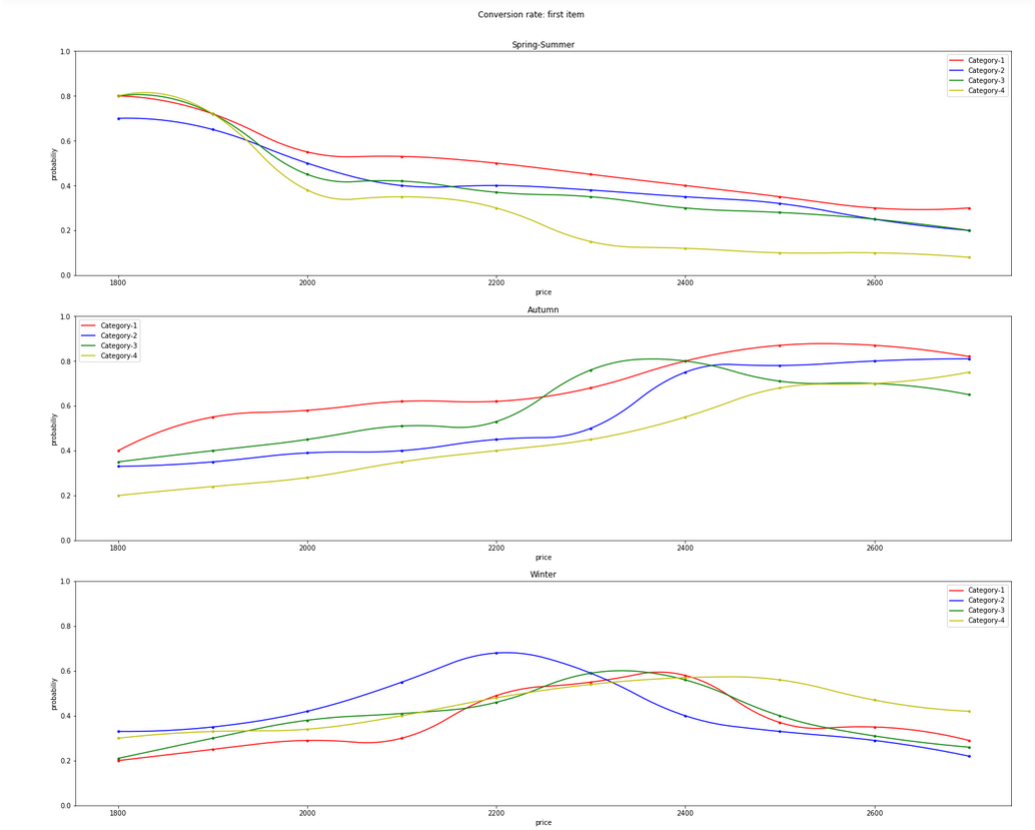


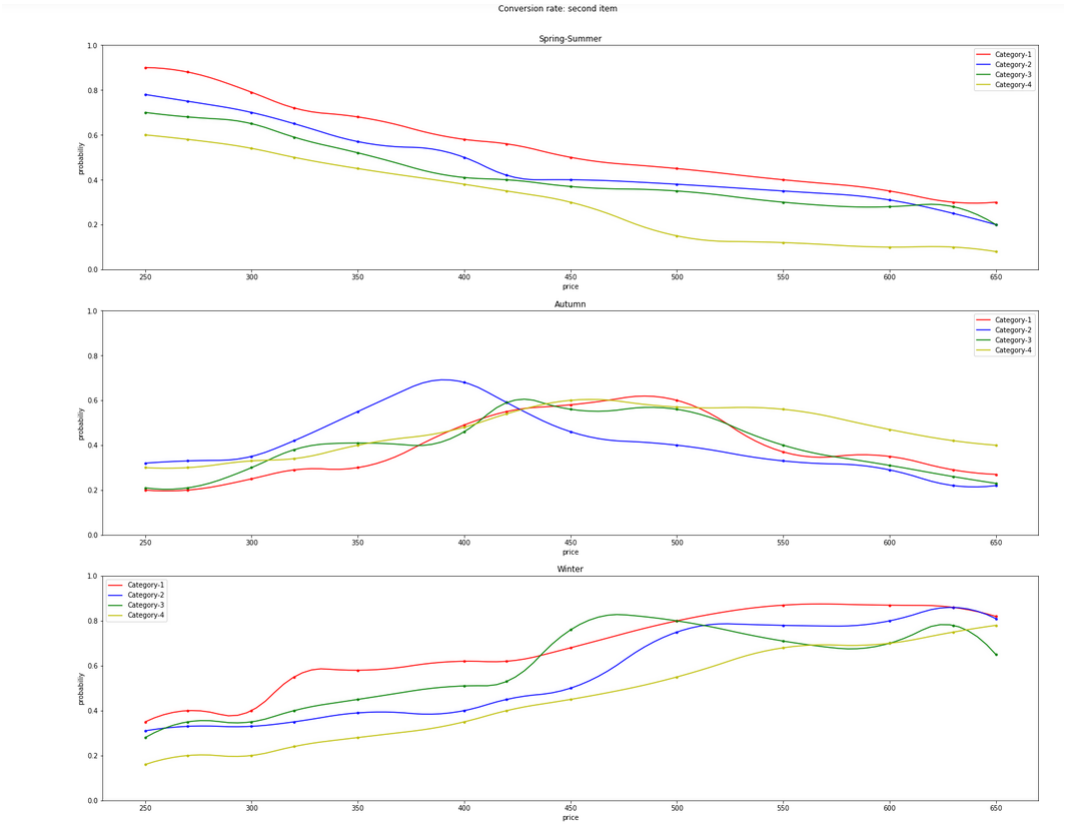
Conversions rates

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 $\sim$ 0,1
- Probability of buying *Racing ski helmet*: Bernoulli $\sim$ 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 seasonalities (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
Autumn
Winter
Category-1
Category-2
Category-3
Category-4
probability
price
1800
2000
2200
2400
2600
0.0
0.2
0.4
0.6
0.8
1.0



Online Approach

Our general approach for the online problem is to simulate the shop, day by day, generating the customers and emulating their behaviors, 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 proposed to the client, in order to maximize the overall 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 suggest which is the best discount to apply to the user in order to maximize the reward.
This procedure is repeated for all clients during the entire time horizon of 365 days.

Step 3 - Learning price for racing skis with known demand curve

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

Implementation: *n3.py*

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
- μ_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
- θ_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 learners, 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:

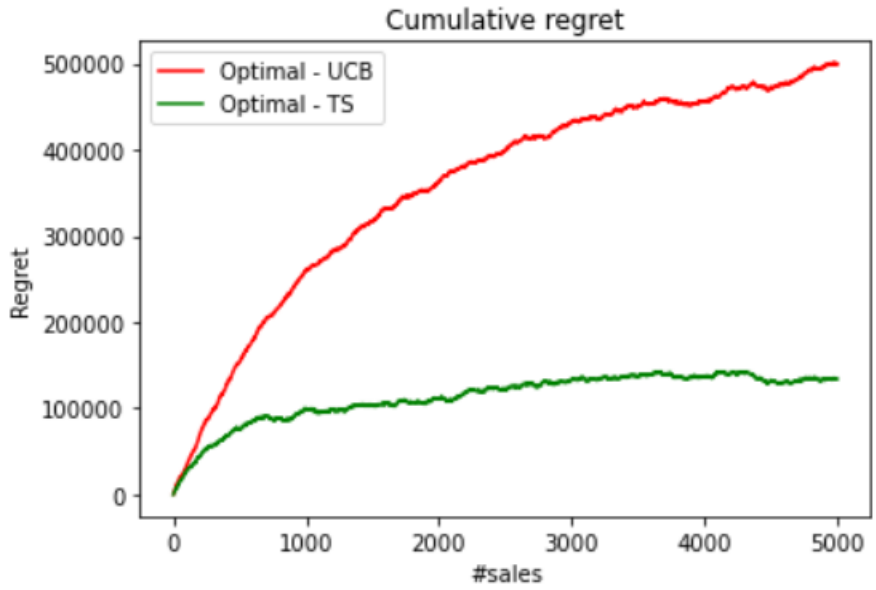
User category	Assigned promotion	Racing Ski Helmet price
Sport Addicted	P_2 : 20%	504.0
Gifter	P_1 : 10%	567.0
Amateur	P_0 : 0%	630.0
Worried	P_3 : 30%	441.0

- Conversion rate associated with the second item is known

Both UCB and TS learner expect as learning parameter a binomial value, while our reward is a value composed by the sold of the first item plus the eventual sold of the second item (equal to zero or grater than one). 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 the approaches 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 allows to have a lower regret.

Step 4 - Learning price for racing skis 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.

Implementation: *n4.py*

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

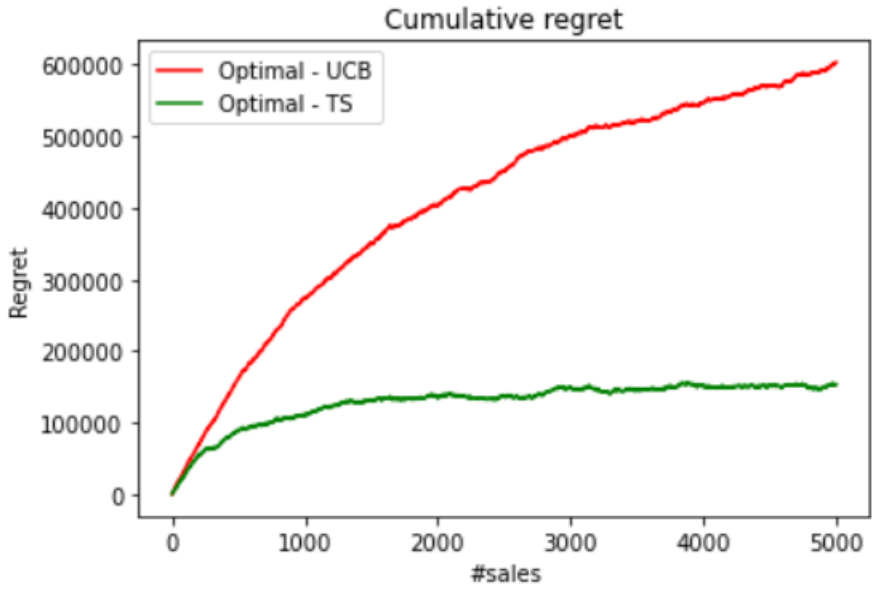
- No seasonality, conversion rate do no change
- The number of customers per class is not known
- The conversion rates associated to the second item are not known
- Candidates for the *Racing Skis* are: {2260.0,1910.0,2130.0, 2010.0, 2340.0}
- Conversion rate associated with the first item is not known
- 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 learner, increasing the inaccurancy of the estimations.

Step 5 - 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 categories. This time we are dealing with a matching problem.

Implementation: *n5.py*

Basic knowledge

A matching problem can be modeled as a graph $G = (N, A)$, where N is a set of nodes and A is a set of arcs. 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 matching 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 the subset of arcs that corresponds 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 modified 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 represents 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 bounds. 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 starting delay in which we explore all possible configurations in an exhaustive 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 purchases the *Racing Skis*, we ask the learner (our custom UCB matching learner) to retrieve the optimal assignemnt promotion-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. It is 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 rewards obtained by the sold of the first item plus the eventually sold of the second. However, since our goal is to learn the matching for the promo-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 rates do not 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

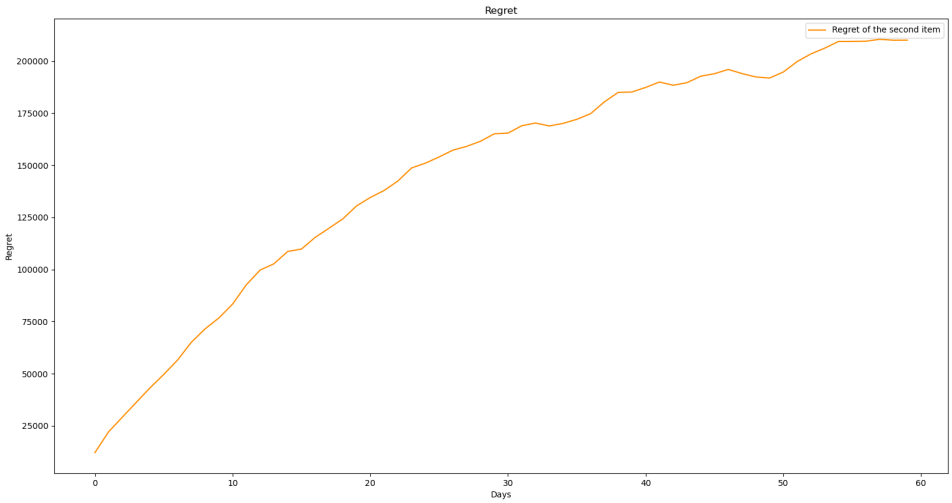
Optimal strategy

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.

Step 6 - Pricing and Matching

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.

Implementation: *n6.py*

Strategy

We simulate the random arrival of the customers. We have used a TS learner, with a number of arms equal to the product between the number of pricing candidates for the first item and the number of pricing candidates of the second item. We pull the learner to retrieve a superarm, that specifies the couple of prices of the two items. For each possible superarm we have a Promo-Category UCB learner that is used to learn the promo-category assignment that maximize the reward. We evaluate, for each customer, the purchase of the first item at the suggested price. In case that the customer buys the first item, using the Promo-Category UCB learner associated with the superarm, we pull the optimal matching and we propose to the client the second item at the discounted price, according to the category he belongs to. 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.

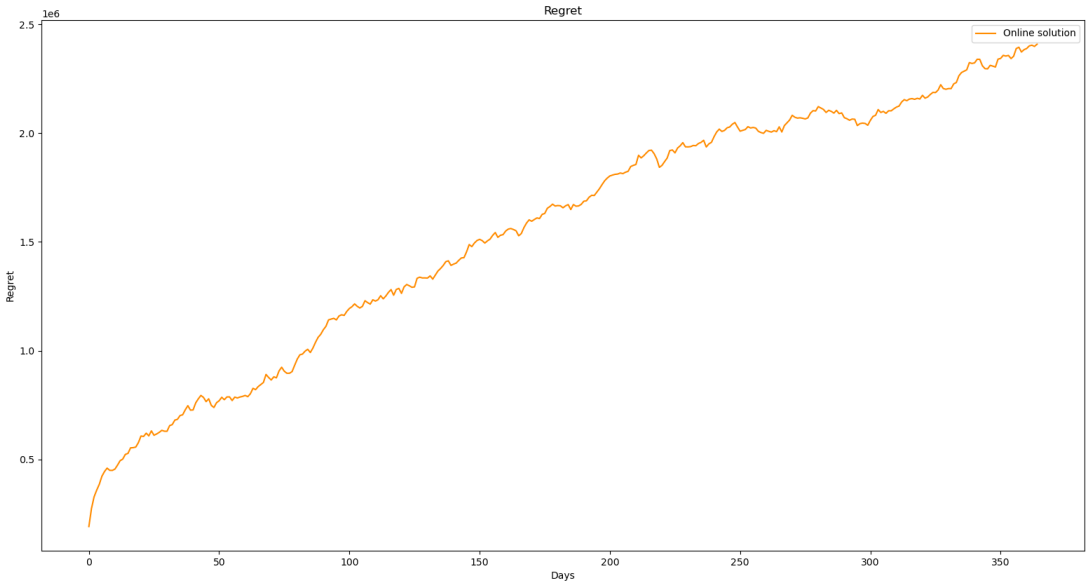
Implementation

- No seasonality, conversion rate do no change
- Number of customers per class is not known
- Candidates for the *Racing Skis* are: {2110.0, 1900.0, 2420.0, 2690.00}
- Candidates for the *Racing Ski Helmet* are: {360.0, 410.0, 530.0, 600.0}
- Conversion rate associated with the first item is not known
- Conversion rate associated with the second item is not known
- Promotion assignment is not known

Optimal strategy In order to calculate the regret, we have defined the optimal solution calculation in the following manner, using an offline approach. We calculate the maximum achievable reward choosing all the possible configuration of prices for the two items. We compute the achievable reward for every couple of candidates as the sum of the achievable reward of the two items, taking into account that the sold of the second item is limited by the number of the customers that by also the first one. According to our candidates the optimal solution is:

Season	*Racing Skis optimal price*	*Racing Ski Helmet* optimal price	Optimal promo-category matching
Spring-Summer	1900.0	410.0	Sport addicted: P_0 = 0% (410.0) Gifted: P_2 = 20% (328.0) Worried: P_3 = 30% (287.0) Amateur: P_1 = 10% (369.0)

Results



Days: 365
Experiments number: 3
Starting delay of the Promo-Category UCB Matching: 1000 clients

Considerations

We can notice that the learners take more time to learn the optimal solutions for both pricing and matching. The cumulative regret is increasing quite linearly up to the middle of the time horizon, after that, they start to stabilize on the optimal solutions. The cumulative regret still be jagged, because the performance of the matching are worst compared to the pricing, moreover we have to take in consideration that the choice of the first item price impact on the performance of the second item (an unattractive price for the first item decrease the number of samples usable for the second price learning).

Step 7 - Pricing and Matching: 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 we are in a non-stationary environment, meaning that we have to introduce the seasonality. It is required to use a sliding-window approach.

Implementation: *n7.py*

Basic knowledge

Sliding Window Thompson Sampling (SWTS) The Sliding Window Thompson Sampling is an alternative to the classical Thompson Sampling algorithm that is susceptible to changes. The main difference is the introduction of a parameter, the sliding window, of length $\tau \in \mathbb{N}$ such that the algorithm, at every round t, takes into account only the rewards obtained in the last τ 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

The strategy is the same of the previous submission with the introduction of the seasonality. We use two learner that will retrieve the optimal superarm describing the couple of prices for the two items. One learner is the classical Thompson

Sampling, the same used in the previous submission, while the other one is the Sliding Window Thompson Sampling. The size of the window used for the SWTS is computed as the $\sqrt{days * avgDailyCustomer} * \alpha$, where $days$ is the time horizon, $avgDailyCustomer = 1000$ is the normalizing factor used to compute the daily customer instances and $\alpha = 30$ is a multiplicative factor empirically computed. Regarding the matching phase we do the same as the previous submission using a Promo-Category UCB learner for every superarm. In order to deal with seasonality, at the beginning of every new season all the matching learners are reset.

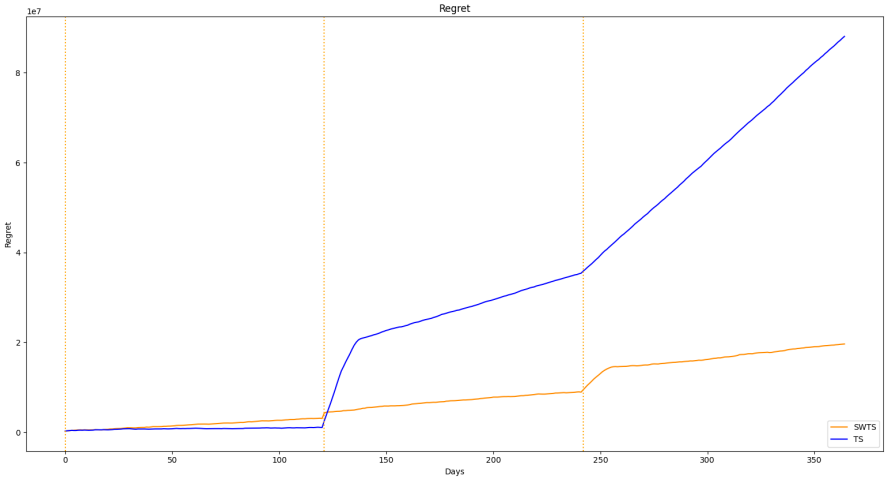
Implementation

- Conversion rates change according to the season
- The time horizon is equally divided into three seasons
- Candidates for the *Racing Skis* are: {2110.0, 1900.0, 2420.0, 2690.0}
- Conversion rate associated with the first item is not known
- Candidates for the *Racing Ski Helmet* are: {360.0, 410.0, 530.0, 600.0}
- Conversion rate associated with the second item is not known
- Optimal promotion-category assignment need to be learnt

Optimal strategy The computation of the optimal strategy is the same performed in the previous submission but is calculated three times, taking into account the three seasons.

Season	*Racing Skis optimal price*	*Racing Ski Helmet* optimal price	Optimal promo-category matching
Spring-Summer	1900.0	410.0	Sport addicted: P_0 = 0% (410.0) Gifted: P_2 = 20% (328.0) Worried: P_3 = 30% (287.0) Amateur: P_1 = 10% (369.0)
Autumn	2690.0	530.0	Sport addicted: P_1 = 10% (477.0) Gifted: P_3 = 30% (371.0) Worried: P_2 = 20% (424.0) Amateur: P_0 = 0% (530.0)
Winter	2420.0	600.0	Sport addicted: P_3 = 30% (420.0) Gifted: P_1 = 10% (540.0) Worried: P_2 = 20% (480.0) Amateur: P_0 = 0% (600.0)

Results



Days: 365
Number of seasons: 3
Season length : $|365/3| = 122$ days
Experiments number: 3
Starting delay of the Promo-Category UCB Matching: 1000 clients
SWTS window size : $\sqrt{365 * 1000} * 30$

Considerations

We can observe that in the first season the TS perform better since it has a complete knowledge of the collected samples, while the SWTS discards the older samples. However, when the conversion rates change, due to the change of the season, with the sliding window approach the newer samples become predominant. Thus, the algorithm changes its behaviour adapting the solution to the new season. We can note that the cumulative regret for the SWTS is about 4 times less than the TS.

Step 8 - Pricing and Matching: 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 we have to manage the seasonality during the matching learning phase, using a change detection approach.

Implementation: *n8.py*

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

A change detection mechanism is added to the classical UCB algorithm. The bandit algorithm take the optimal decision, according with its past observation. During the update phase, the rewards are observed also by the change detection algorithm that monitors the distribution of each arms, and sends to the UCB a positive (or negative) signals in case of detection and the bandit resets the arm.

Pseudocode

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

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

$a_t \leftarrow$ 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

Promo-Category CD-UCB Matching

Also in this case we have developed a custom version of the UCB algorithm that we use in our scenarios. As for the modification we have done for the UCB without the change detection mechanism, we have added two matrices (support matrix and total reward matrix), used to compute the average reward obtained by each arm. The same initial phase of exploration (the starting delay) is adopted also in this case. The main difference is that we use a CD-UCB implementation and when a detection is present the two additional matrices are reset.

Strategy

In order to solve this problem we have used our customized implementation of Change Detection UCB algorithm, built on top of CUMSUM algorithm and classical UCB bandit algorithm. As for the previous submissions, we simulate the purchase of the two items client by client. We select the optimal superarm with the pricing learner, then, usign the Promo-Category CD-UCB we retrieve the optimal matching promotion-category for the current user. Finally we compute the rewards and update the learners.

Implementation

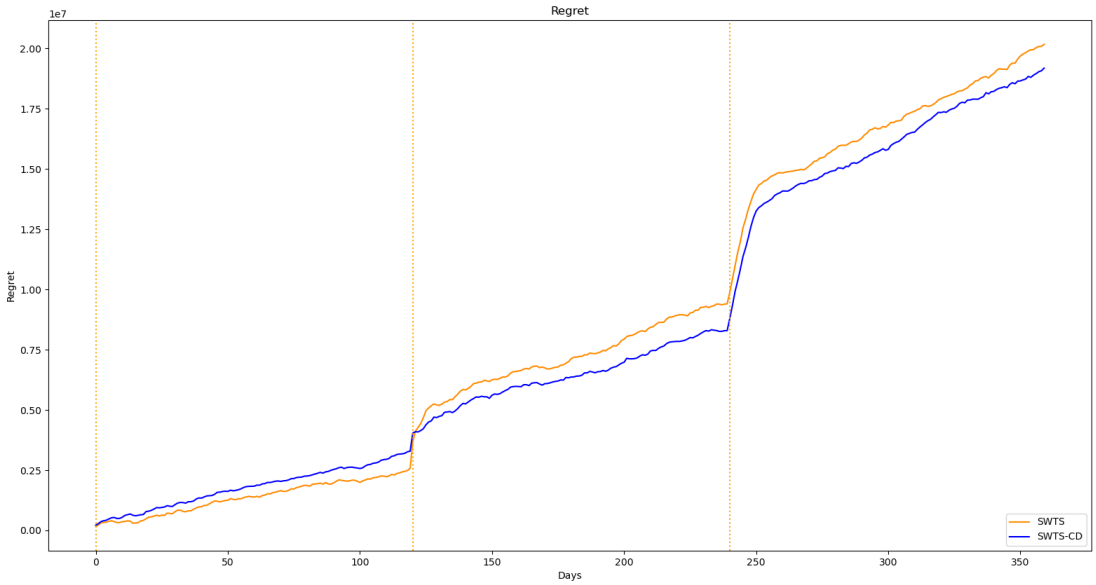
- Three seasons: Spring-Summer, Autumn, Winter
- Number of customers per class is not known

- Candidates for the *Racing Skis* are: {2110.0, 1900.0, 2420.0, 2690.00}
- Candidates for the *Racing Ski Helmet* are: {360.0, 410.0, 530.0, 600.0}
- Conversion rate associated with the first item is not known
- Conversion rate associated with the second item is not known
- Promotion assignment is not known
- Change-detection approach

Optimal strategy We have calculate the optimal solution in the same way as in step 7.
According to our candidates the optimal solution is:

Season	*Racing Skis optimal price*	*Racing Ski Helmet* optimal price	Optimal promo-category matching
Spring-Summer	1900.0	410.0	Sport addicted: P_0 = 0% (410.0) Gifted: P_2 = 20% (328.0) Worried: P_3 = 30% (287.0) Amateur: P_1 = 10% (369.0)
Autumn	2690.0	530.0	Sport addicted: P_1 = 10% (477.0) Gifted: P_3 = 30% (371.0) Worried: P_2 = 20% (424.0) Amateur: P_0 = 0% (530.0)
Winter	2420.0	600.0	Sport addicted: P_3 = 30% (420.0) Gifted: P_1 = 10% (540.0) Worried: P_2 = 20% (480.0) Amateur: P_0 = 0% (600.0)

Results



Days: 365
Experiments number: 2
Season length : $|365/3| = 122$ days
Starting delay of the Promo-Category CD-UCB Matching: 1000 clients
SWTS window size : $\sqrt{365 * 1000} * 30$

Considerations

We can observe that the change detection approach has a smaller impact respect to the performance of the SWTS. This is due to the fact that the change detection approach catches some false positive detection; the matching of promo-category of the second item is influenced by the prices chosen for the two items.

Folder Structure

We have done a script file for every step of the project *(n1.py, n3.py, n4.py, n5.py, n6.py, n7.py, n8.py)*. The first two steps are theoretical and they are included in

this report, by the way we have implemented, for completeness, a script for the first step that find the optimal solution of the offline optimization problem.
In addition, we implemented the following codes, in order to compute the presented results.

Config.py is the configuration files and contains:

- The samples used to reconstruct the demand curves of the two items
- The parameters of the customer distribution
- Other description information about the environment

Context.py Context.py contains the definition of the class that simulates the behaviour of the context. All the experiments use this class. It is initialized with the values written in the config.py. It implements the methods related to the purchase of the two items based on their conversion rates, the generation of the daily customer numbers per class, the algorithm to calculate the optimal solutions for the non-stationary environment and the plot functions for the demand curves and the other distributions.

Algorithms folder The Algorithms folder contains all the learning algorithms: UCB and Thompson Sampling, with their customized version.