Online Learning Applications Project

Social Influence & Advertising

Pablo Giaccaglia
Gianmario Careddu
Marco Bonalumi
Sara Zoccheddu
Riccardo Ambrosini Barzaghi

Index

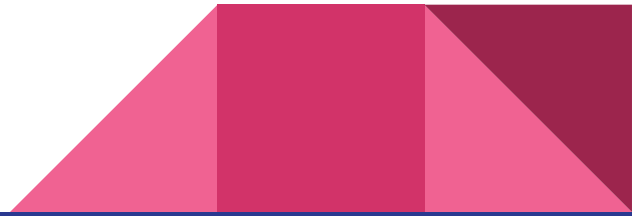
1. Intro and Assumptions

Scenario

The **scenario** we intend to model is related to an e-commerce grocery website, which is involved in selling unlimited number of units of 5 different items. Each product is associated to an advertising campaign.

The e-commerce website has a daily budget B to spend for advertising its products.

The **goal** is to find every day the best allocation of budgets into the campaigns, thus maximizing the profit of the e-commerce.



Glossary

- **α value:** used as relative value to express number of users
- **N_{users}**: expected total number of customers without advertising (reference for α=1)
- **α function:** every user class has **5 alpha functions**, one per campaign, that describe how the user responds to that campaign
- **$\alpha_{c,max}$**: maximum alpha value reachable with infinite budget for a campaign
- **B**: budget that website can spend in advertising
- **λ**: probability with which the slot of secondary product is observed (1 for first secondary product).
- **Reference price**: price of product 5 (the most expensive)

Fundamental assumptions and choices

- We can observe all the activities of the users on the website so from historical data we can segment them
- Campaigns are independent
- The total alpha value can be > 1 to allow the simulator to be more elastic
- For every primary product, the pair and the hierarchy of the secondary products to be displayed is fixed by the business
- We disregard profits coming from users navigating our website daily, instead we only consider profits coming after clicking on an ad (new customers)
- The price given to products is a relative price from 0 to 1, where 1 is the price of product 5 (maximum of products prices)

2. Simulator Engine

How a user navigates the website (assignment)

Every webpage displays a **primary product** and its price. The user can decide to add a certain number of units of the product to his cart. He/She will decide to buy the item only if the single item's price is below the user's reservation price. When the user adds the product to the cart, two secondary products are displayed in two slots, one above the other, and their price is hidden.

If the user clicks on a **secondary product**, a new tab is opened and the clicked page is displayed as primary product with its price. At the end of his visit in the ecommerce website, the user buys the product added to the cart.

Simulator working principles

Our simulator considers a number of users $\mathbf{N_{users}}$ as reference (that is, the expected number of users buying on the E-commerce without any advertising campaign)

The **alpha value** determines what fraction of N_{users} will land on the E-commerce website and what fraction will land on the specific primary products pages.

1. The sum of the alpha values over all primary product pages can be greater than 1. This allows to be more general, also considering the case in which a really good allocations of the budget into campaigns brings a number of customers greater than N_{users}.

All the transaction are made with a “generic currency” that takes as a **reference price** the price of the product 5 (details later).

Simulator working principles (Campaigns)

Every campaign campaign can receive a budget [0, +∞).

As a function of the budget a campaign c can increase the total alpha value by $[0, \alpha^{c}_{max}]$

The campaign can be seen as a function c: budget → α value.

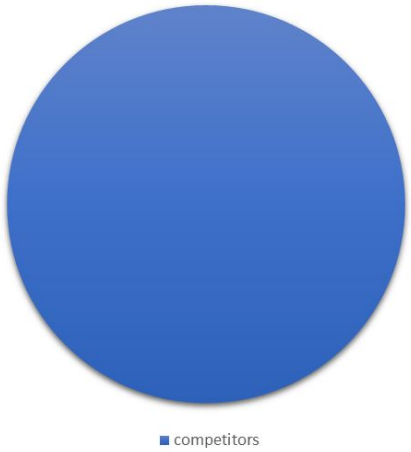
The shape of the c function is a property of the user affected by the campaign, and it is called the α **function** of a user versus campaign c.

Given all α^{c}_{max} we can define the **max α value** as

$$\alpha_{\max} = \sum_{c \in C} \alpha^{c}_{\max}$$

α values: graphical representation

Case: $B = 0$



Case: $B \neq 0 < \infty$

Case: $B = +\infty$

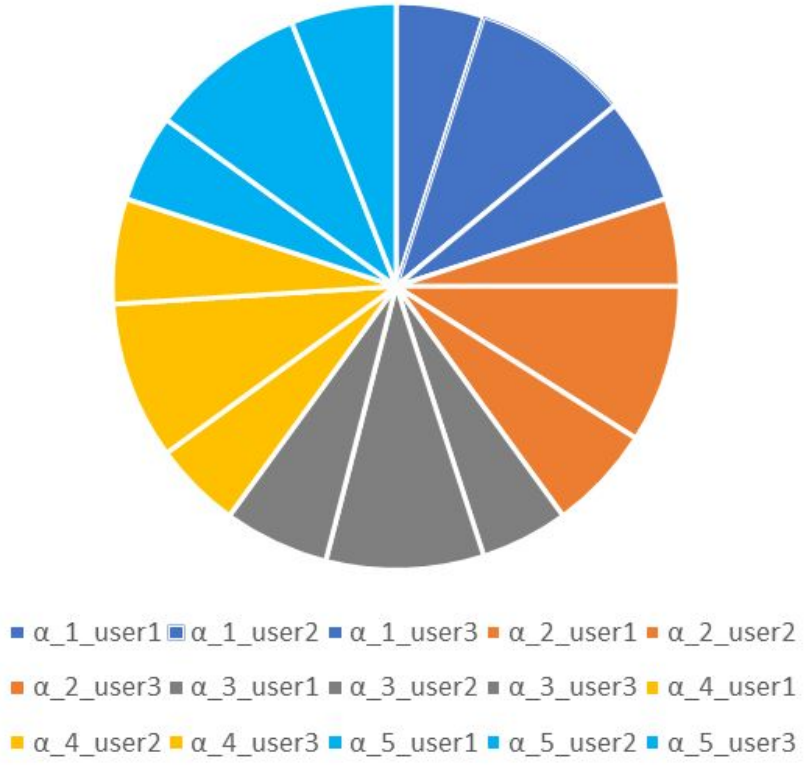
Comment on prev. example

We assume that all the users not landing on our E-commerce website, will land on a competitor site.

The more budget we allocate on the campaigns, the more users will land on our E-commerce website, up to the saturation of each campaign to their corresponding $\alpha_{max}^{campaign}$ ($B \rightarrow +\infty$).

Since we have 3 user classes and 5 campaign, the alpha value generated by each campaign has to be splitted into 3 components. Therefore, there are 15 alpha functions, and the alpha_value for a single campaign is obtained scaling the alpha function results [0, $\alpha_{max}^{campaign}$] by the user class probability .

α values: complete picture



Comment on prev. example

Allocating more and more budget on the campaigns causes the total alpha_value to increase, meaning that more users will land on the primary product pages.

The alpha_value produced by a campaign *c* is the sum of the alpha_functions associated to that campaign of each user class, weighted by the class probability.

$$\alpha_c = \sum_{j \in U} p_j * f_{j,c}(b * p_j)$$

b budget allocated to campaign c

$f_{j,c}$ alpha function of user j for campaign c

According to the user class/type, there will be an expected number of purchases using the general currency.

Now, given the N_{users} and the reference price in euro we can compute the total revenue expressed in euro.

Working on expected values simplifies the simulation, because we do not have to simulate each single user and repeating it day over day

Mathematical formulation revenue

$$\text{Revenue}_{\text{user } i} = p_i * N_{\text{users}} * R_{\text{price}} \sum_{(c,b)\in A} f_{i,c}\,(b * p_i) * E_{\text{profit} \sim \text{user } i}\,(\text{Primary}_c)$$

- **f** is the alpha function of user i per campaign c
- $\mathbf{p_i}$ can be interpreted as user class/ type probability without loss of generality
- **B** set of budgets
- **C** set of campaigns
- **A** set of allocations (c,b)
- $\mathbf{Primary_c}$ is the primary product targeted by campaign c

Mathematical formulation revenue cont.

$$\text{Revenue}_{\text{user } i} = p_i * N_{\text{users}} * R_{\text{price}} \sum_{(c,b)\in A} f_{i,c}\,(b * p_i) * E_{\text{profit} \sim \text{user } i}\,(\text{Primary}_c)$$

The information of the expected number of items sold is embedded in the computation of the **Expected profit** and is computed by the **DFS (depth-first search)** path following algorithm

$\mathbf{N_{users}}$, $\mathbf{R_{price}}$, $\mathbf{p_i}$ are properties of the environment

The α functions are properties of users classes (identical if we consider types for type 1 and 2)

Noise on $\boldsymbol{\alpha}$ **values** and on **expected number of purchases** is introduced as a product on the revenue

Code overview (graphs)

The navigation of each user class is modelled using a **fully connected graph** where the nodes are products and the weight on edges (i, j) is the probability to buy P_j given the purchase of P_i.

One random graph per user class is generated when the object Environment is created.

Despite the graph being fully connected, the users can buy only 2 of the products Pj for each edge (i, j) for all products Pi. This is modelled using a **secondary list** for each product, containing the 2 product ids that can be purchased after.

Code overview (expected profit)

The expected profit of each user is computed running a **recursive DFS path following algorithm** for each product.

For each product and at each node visit the algorithm computes the **visit probability** given the trace of probabilities of previous visits. In particular if the user reservation price exceeds the node price it collects the profit (expressed in relative currency) and weights it by the **visit probability**.

From a node P_i the navigation continues into nodes P_j and P_k, with probability 1 for P_j and lambda for P_k. Where P_j and P_k are the products in the secondary list of P_i. If a node P_x has already been visited the algorithm stops following that subpath

Code overview (expected profit cont.)

Calling the function expected_profit of a user instance we will get a list of 5 expected profits, one for each product displayed as primary product.

If we were in a more complicated situation without the **secondary list** the DFS would need to follow an exponential number of path in the branching factor (4 in our case). That is why we decided to restrict the visit using the **secondary list** but the algorithm could be adapted only by changing the termination condition of the DFS.

We decided to use a fully connected graph to cover the most generic case, but the simulator can be easily adapted to work with any kind of graph, with or without secondary list employing any kind of weight values.

3. Environment Setup

Products

The e-commerce sells **5 products**, after each purchase there are **2 suggestions**.

The prices are expressed as a proportion of the most expensive product (5). Doing so, we only need to set **one price** into the simulator.

Product	Price	Suggestion 1	Suggestion 2
Product 1	0.5	P2	P3
Product 2	0.625	P3	P4
Product 3	0.75	P1	P5
Product 4	0.875	P2	P5
Product 5	1 (reference)	P1	P4

Campaigns

A campaign can attract a finite amount of visitors (α_{max}) even when allocating infinite budget to it.
Each campaign can target only one product, but it affects all user types.

Campaign	α_{max}
Campaign 1	0.4
Campaign 2	0.4
Campaign 3	0.2
Campaign 4	0.3
Campaign 5	0.2

Users

The users are divided into **4 types** and they are described by **2 binary features**: student/worker and living alone/family.

In practice **3 classes** exist, the students and workers living with family have the same behavior.

- **User1: Family-Student** → a student that lives with family
- **User1: Family-Worker** → a worker that lives with family
- **User2: Alone-Student** → a student that lives alone
- **User3: Alone-Worker** → a worker that lives alone

	STUDENT	**WORKER**
FAMILY	User1	User 1
ALONE	User 2	User 3

Users cont.

Each user **type** can be observed with a given probability, as a consequence we can derive the **class** probability. User1 can be of both types with uniform probability

Types	Type Probability	**Classes**	Class probability
FAMILY + STUDENT	0.125	User 1	0.25
FAMILY + WORKER	0.125		
ALONE + STUDENT	0.45	User 2	0.45
ALONE + WORKER	0.3	User 3	0.3

Users classes and behavior

Users belonging to the same class share the following properties

- **Reservation price**: price after which they buy an item
- **Expected number of purchases**: expected number of items purchased
- **λ :** probability to observe suggestion 2 after a purchase
- **Segmentation size**: probability to observe the user class

and same functions (α functions) mapping budget allocated to a specific campaign to the effectiveness of the campaign over the user class (α value). In details:

- **Resistance to ads**: the alpha functions have low saturation speed
- **Activation budget**: the alpha function is activated for higher budgets

α functions behaviour

The behavior of the user classes has been represented designing the following alpha functions using a sigmoid-like function

$$f(x) = \max\left(0, -1 + \frac{2}{1 + e^{-\frac{v}{x-a}}}\right)$$

the alpha function by default saturates to 1, they need to be scaled by the alpha max of the campaign when employed

v = saturation speed

a = activation point

α functions and budget

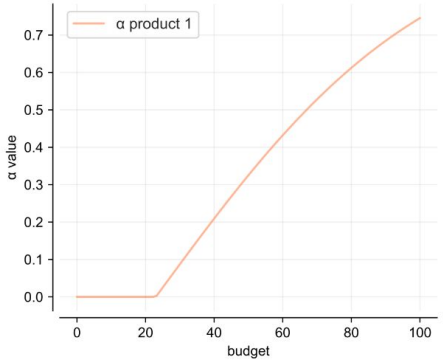
The input of the function is the budget allocated to the campaign

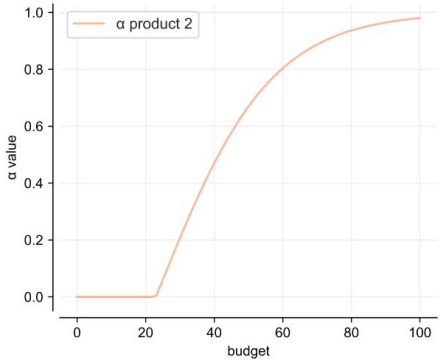
$$f(x) = \max\left(0, -1 + \frac{2}{1 + e^{-\frac{v}{x-a}}}\right)$$

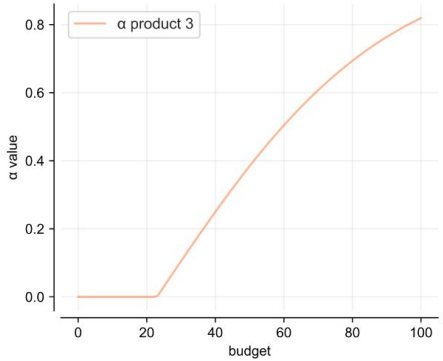
When considering the aggregated revenue the input budget need to be scaled according to the class/type probability depending on how the aggregation is done.

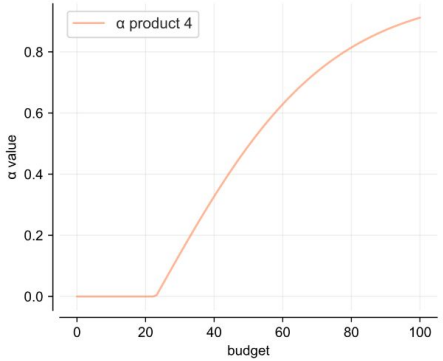
Otherwise we obtain the same effect of spending 3/4 times the budget but spending it once

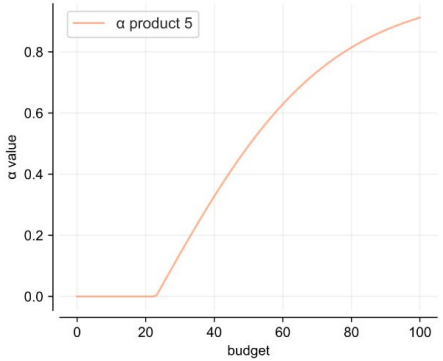
α functions - user 1



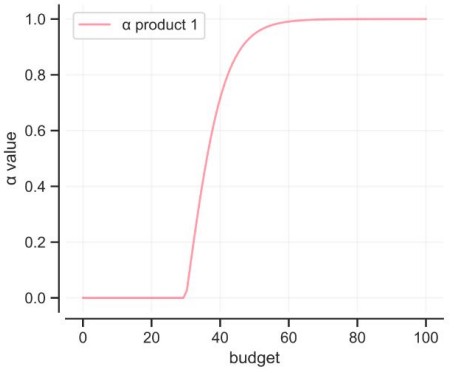


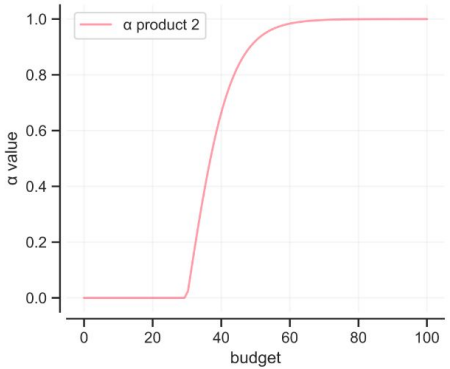


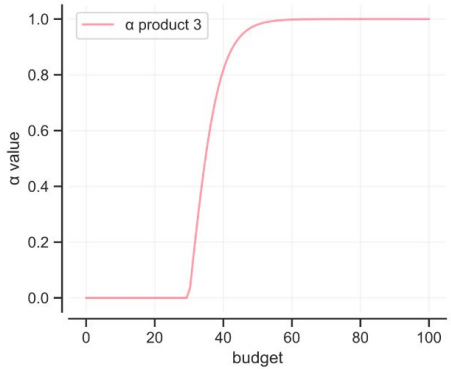


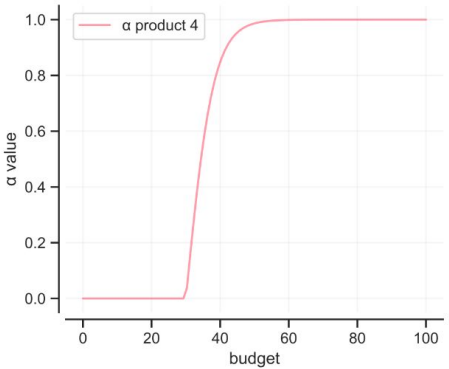


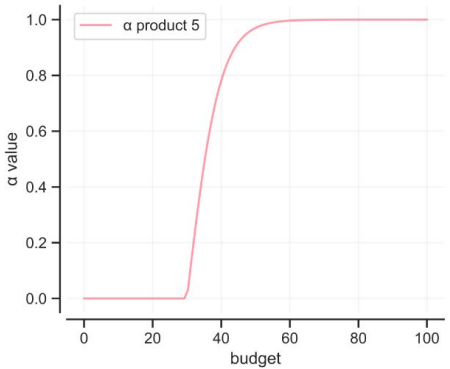
α functions - user 2



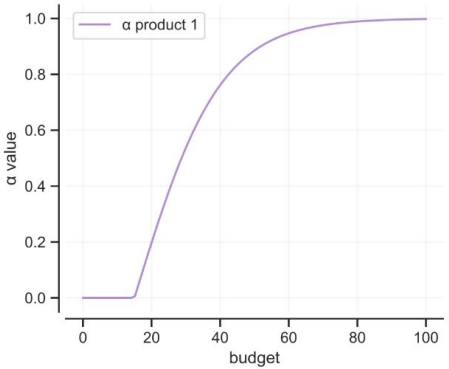


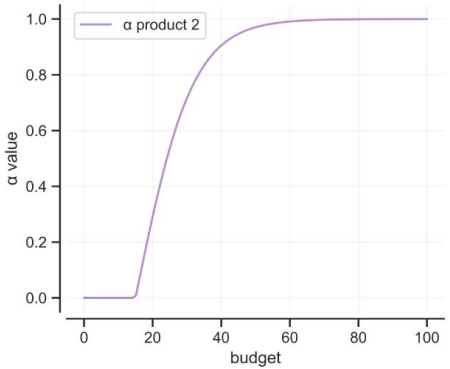


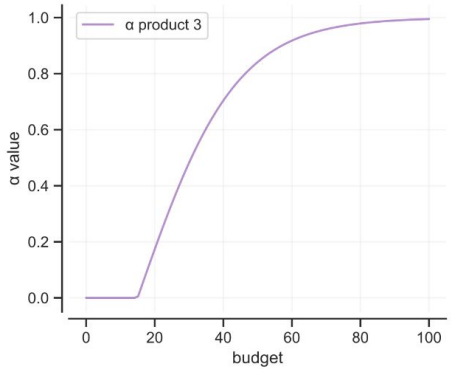


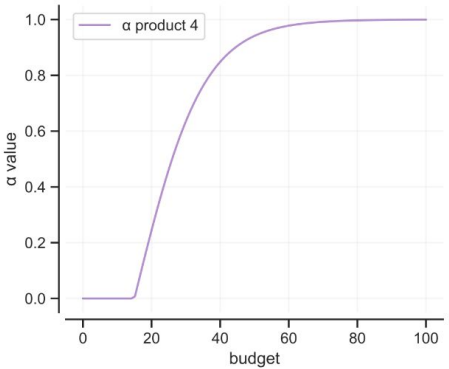


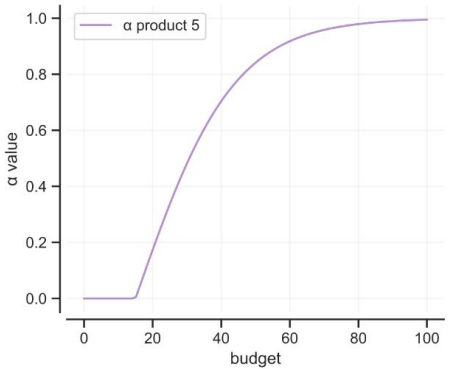
α functions - user 3











User classes: details

User1 (family)

- Segment size: MEDIUM
- Resistance to ads: HIGH
  (it depends also on other members)
- Activation budget: MEDIUM
- Reservation price: MEDIUM
- Expected n_purchase: HIGH
  (many people to satisfy)

User2 (student)

- Segment size: HIGH
- Resistance to ads: LOW
  (but more spam is needed)
- Activation budget: HIGH
- Reservation price: LOW
  (little money)
- Expected n_purchase: LOW

User3 (worker)

- Segment size: MEDIUM
- Resistance to ads: MEDIUM
- Activation budget: LOW
  (they need little exposition to start buying)
- Reservation price: HIGH
- Expected n purchase: LOW

SETUP	probability	saturation speed	activation	reservation price	range expected num purchases	λ
User1	0.25	(0.025, 0.06)	23	buy all except P3	(2, 3.5)	0.8
User2	0.45	(0.16, 0.25)	30	buy P1/P3/P5	(1, 1.5)	0.5
User3	0.30	(0.07, 0.12)	15	buy all	(1.5, 2)	0.65

4. Optimization Problem

Problem to face

Allocation of budget over advertising campaign in order to maximize the total profit

What to learn:

- The function $p_c(b)$ of the profit versus the budget spend for each campaign

How to employ the knowledge:

- Having a perfect estimation of all $p_c(b)$ we can find the exact combination of budgets maximizing the profit

In practice:

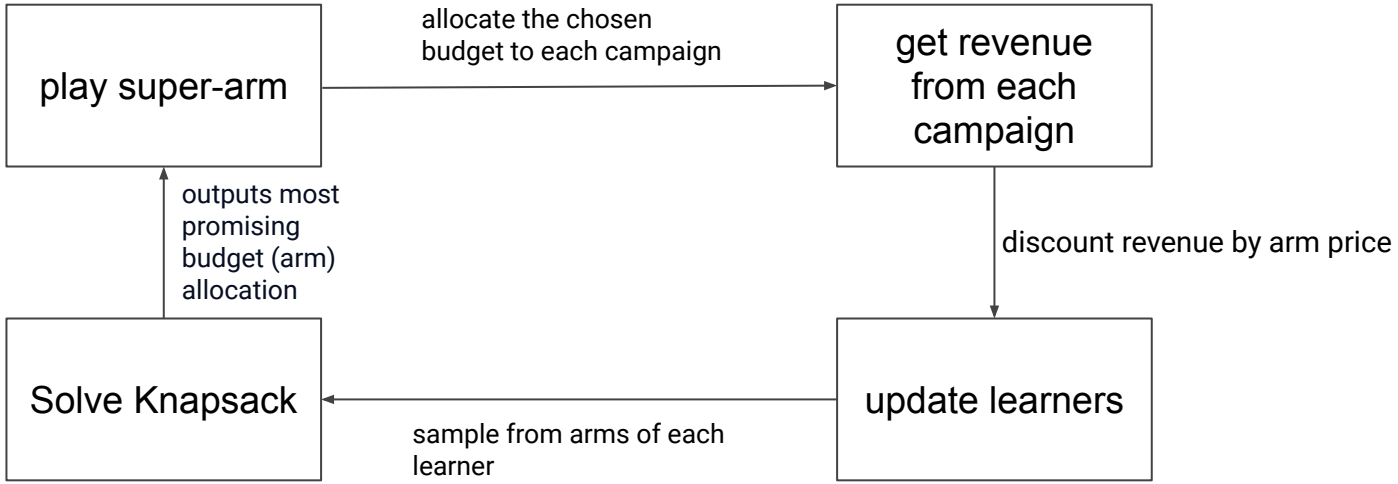
- We can only estimate the functions $p_c(b)$ and the problem of finding the best combination is **NP hard**

Possible solution

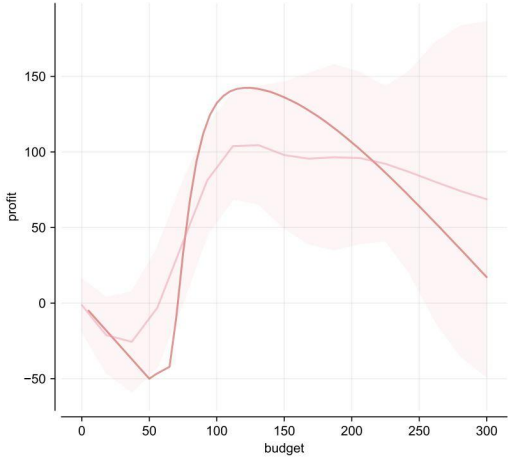
We can tackle this problem using combinatorial bandit algorithms

- We can employ a set of instances of bandit algorithms one per campaign
- Every day we sample/pull from the distribution of each bandit
- Given the arm samples we choose the best possible super arm (set of arms) using an approximate instance of the original combinatorial problem (Knapsack problem in our case)
- We play in the environment the super arm selected by the learner and we collect the revenue generated by each single campaign
- We give as a reward to each bandit the revenue generated by its arm discounted by the cost of the arm itself (basically it is learning the profit)

Combinatorial Online Learning Framework



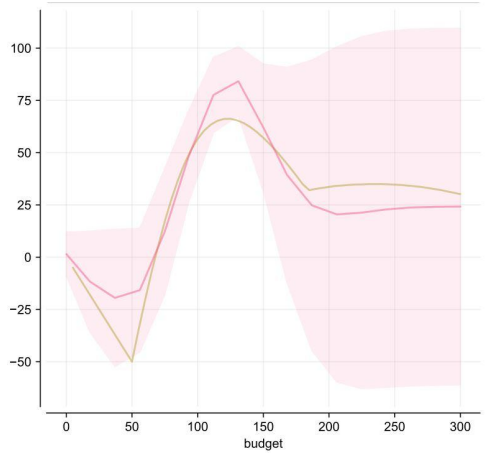
Regularities in the problem



The $p_c(b)$ functions will have a shape similar to the one on the left

- **loss part**: no user is activated the investment is too little
- **positive slope**: the more we invest the more is the profit
- **saturation**: investing more budget will cause less and less profit increase
- **over allocation**: investing more budget than needed

It can also happen to have a positive slope after saturation (right fig.) in case of late activation of users but in general we expect that activation happens close together



Regularities:

- Correlation among budgets
- Continuity

Exploiting regularities

To reduce the regret of the bandit instances we can employ techniques such as Gaussian process to exploit continuity and correlation.

Considering the reward per arm as a Gaussian Process in each bandit, we can employ a **Gaussian Process Regressor** to predict the reward of not yet pulled arms exploiting our knowledge over the $p_c(b)$ functions.

This optimization is also useful to have an estimation of the uncertainty of the estimations.

5. Code Overview: Simulations & Learners

Code overview:

Combinatorial Wrapper

Each learner class is passed to the CombWrapper object which is a wrapper implementing all the logic needed for combinatorial bandits, which is algorithm independent.

The CombWrapper stores a list of learners and can sample from their arms, then a feasible optimal super arm is found solving the knapsack combinatorial optimization problem and returned as suggested optimal allocation.

Given the rewards collected by an allocated super arm from the environment, it dispatches each reward to the corresponding learner.

Learners available to wrapper

	Full Bandit Name
G-TS	Gaussian Thompson Sampling
GP-TS	Gaussian Process Thompson Sampling
GP-UCB1	Gaussian Process UCB1
CUSUM-GP-UCB1	Cumulative Sum Change Detection Gaussian Process UCB1
CUSUM-GTS	Cumulative Sum Change Detection Gaussian Thompson Sampling
SW-GTS	Sliding Window Gaussian Thompson Sampling
SW-GP-UCB1	Sliding Window Gaussian Process UCB1

Code overview:

Environment

The Environment class contains all the logic for the simulation, it can simulate days returning all the random variables outcomes and the corresponding best allocations.

The noise affecting each day defines a “signature” over that day.

The noise “signature” is saved allowing further budget allocations with a perfect replication of all the noisy quantities.

This allows to run in parallel independent simulations on the same main loop without interference.

Code overview:

Simulation Handler

Given all the required metadata, it instantiates environment and combinatorial bandits, then it drives all the simulation steps over the environment.

Simulation are composed of number of days and number of iterations and the results are saved in the file system (results) as JSON and PDF. All the simulations are inside the directory **simulations**.

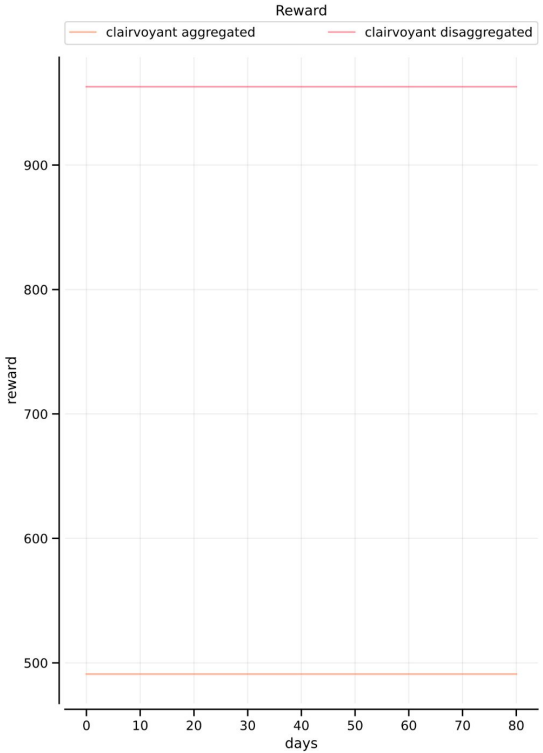
6. Experiments & Results

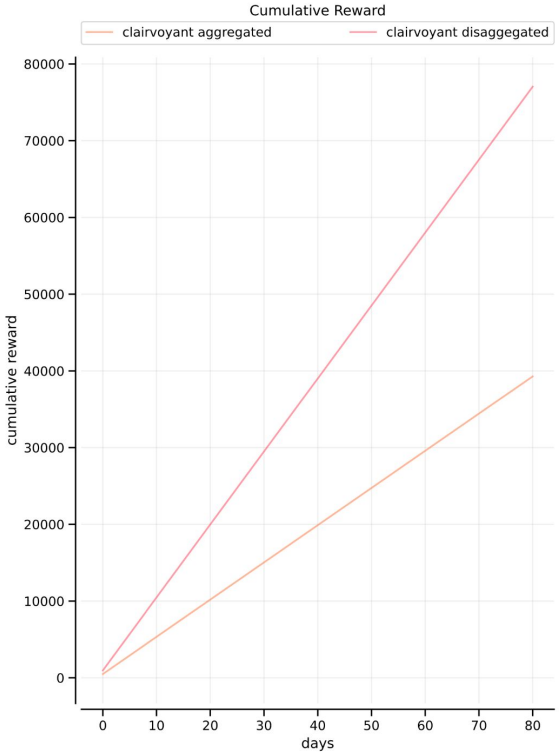
Simulations setup

We have run our simulations by setting the following parameters. In particular, we set the daily budget to €300 and divided it into 5 “chunks”, which is the value chosen for the parameter Budget step.

Experiments	100
Days	80
N_{users}	350
Reference price	4 €
Daily budget	300 €
Budget step	5 (step of knapsack solver)
Number of arms	$3\left\lceil (T\log(T))^{1/4} \right\rceil + 1$ = 13

Step 2 - Optimization algorithm: disaggregated vs aggregated case





	Average daily reward	Average cumulative reward
Clairvoyant disaggregated	77040	491
Clairvoyant aggregated	39280	963

Step 2 - cont.

From the previous graphs it is possible to notice the difference in terms of performance between the aggregate and the disaggregate case, with respect to the clairvoyant algorithm.

In fact, the algorithm run in a disaggregate setting outperforms the one run in an aggregate setting. This is due to the fact that an aggregate model does not allow to distinguish the specific class of each users, unlike the disaggregated model, which can reach a higher reward addressing the users separately.

Step 3 - Optimization with uncertain α functions

Next, we developed algorithms to execute the same task in case the α functions were unknown. Due to the definition of the α functions, we couldn't retrieve the information about the buyer's class and we therefore just aimed at estimating the overall profit (considering an aggregate α).

The optimization has been approached with Gaussian Processes, with both UCB and Thompson Sampling, for the estimation of the α given the budget for each campaign.

Each candidate is a tuple (campaign, budget) with budget evenly spaced allocations. The Knapsack searches for the best superarm that is then played, and from the given rewards of every campaign it is possible to reconstruct the value of the profit curve (aggregated), thanks to the Gaussian Processes that are able to model such curve, starting from noisy samples.

Noise generation

The noise has been generated as a scale factor for both alpha functions and expected number of purchased items in step 4 as:

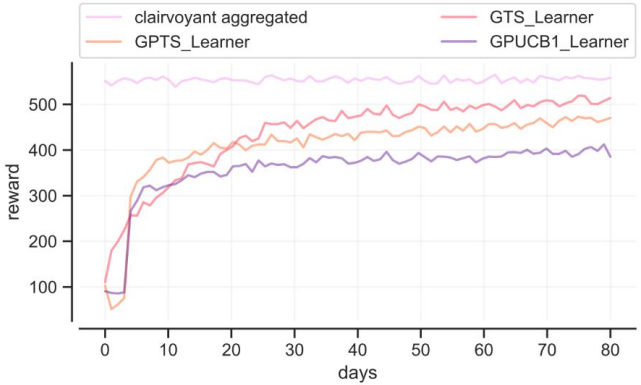
noise factor = 1+ uniform(-0.1,0.1) + gaussian(0, 0.13)

In the first case the reaction of the user to campaign is noisy.

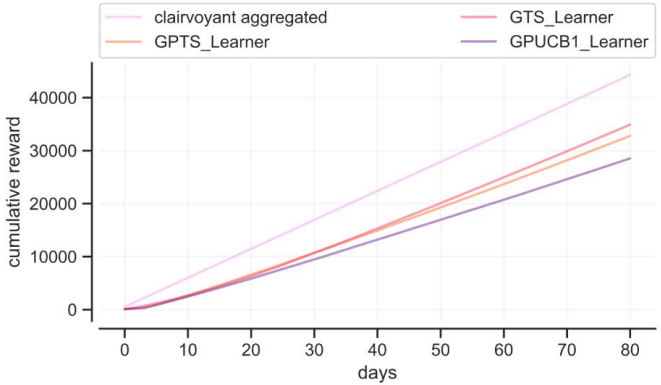
In the second case the noise affects the user expected profit.

Step 3 - Graphical results

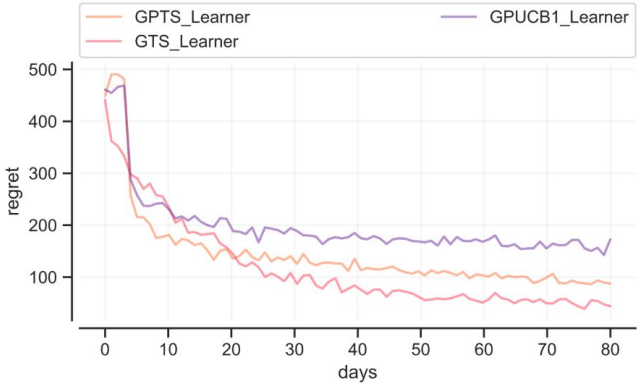
Reward



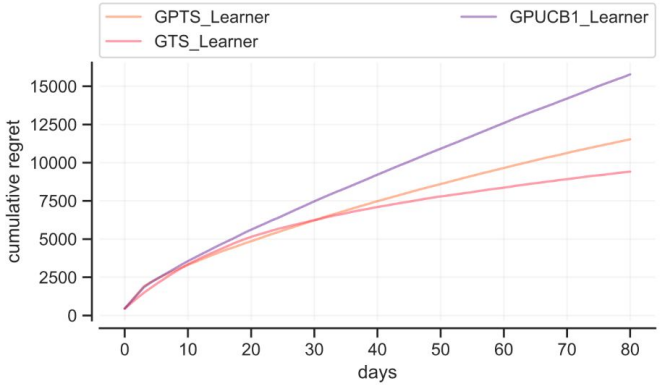
Cumulative Reward



Regret



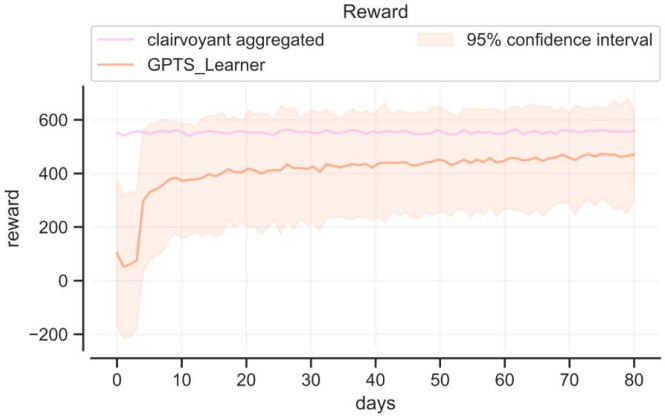
Cumulative Regret

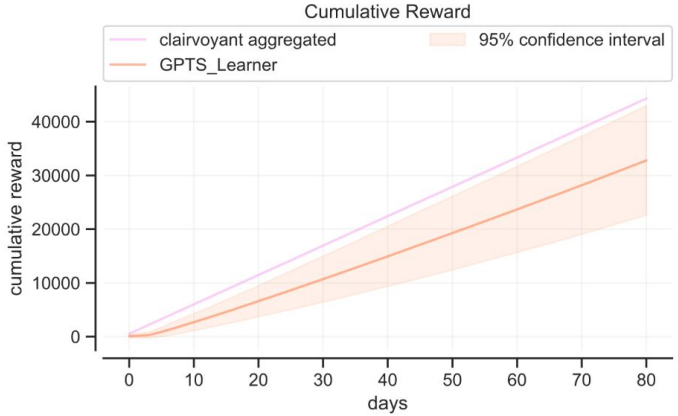


Step 3 - Numerical results

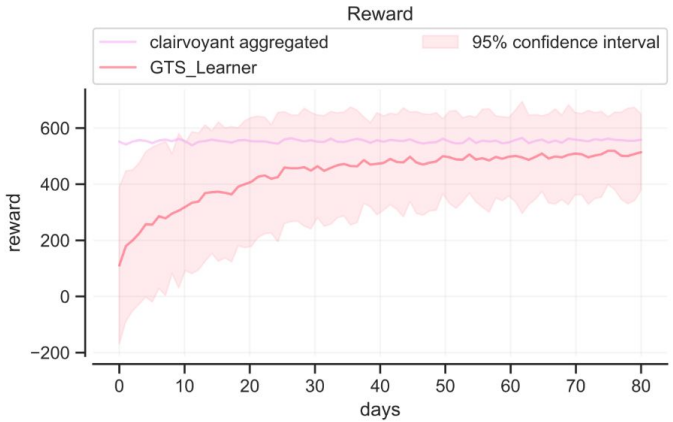
Bandit (combinatorial)	Average daily reward	Stad.dev on average daily reward	Average daily regret	Std.dev on average daily regret	Average cumulative reward	Std.dev on average cumulative reward	Average cumulative regret	Std.dev on average cumulative regret
Clairvoyant aggregated	553	55	0	0	44315	493	0	0
GP-TS	409	134	144	127	32784	5195	11530	5157
G-TS	**436**	**133**	114	125	**34900**	**1626**	**9414**	**1640**
GP-UCB1	356	136	197	132	28531	6273	15783	6288

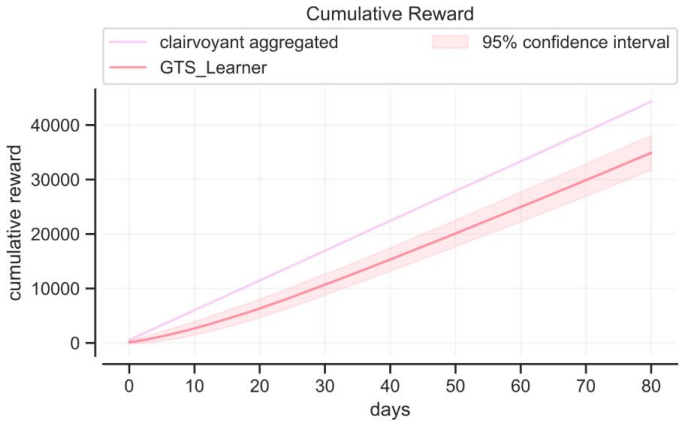
Step 3 - Confidence Intervals > GP-TS

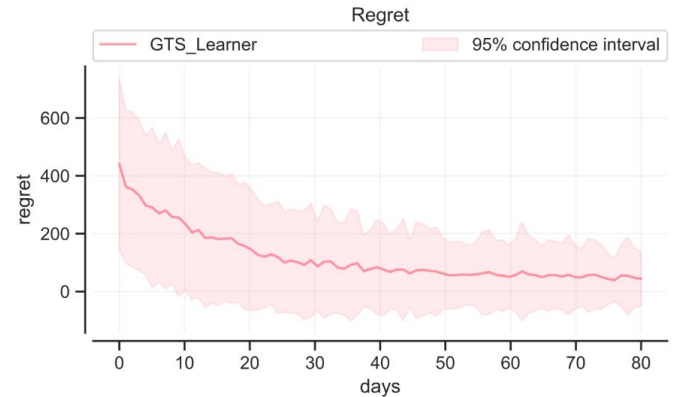




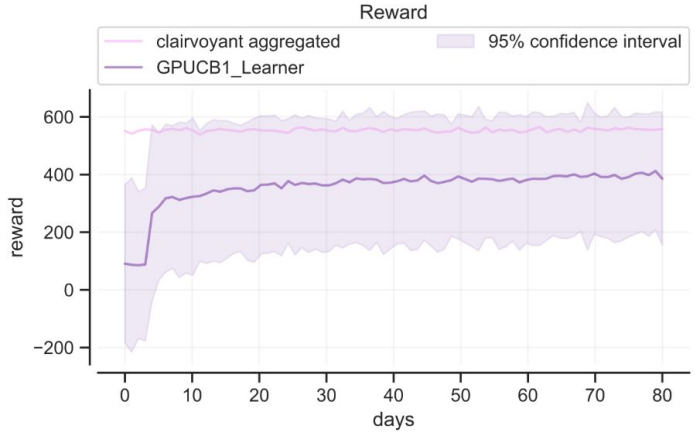
Step 3 - Confidence Intervals > G-TS

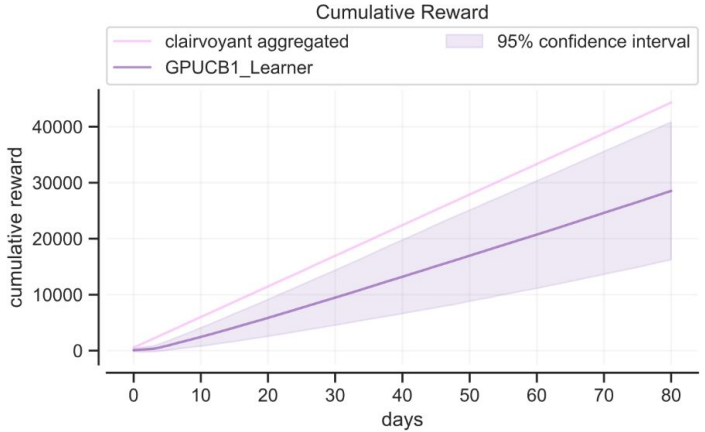


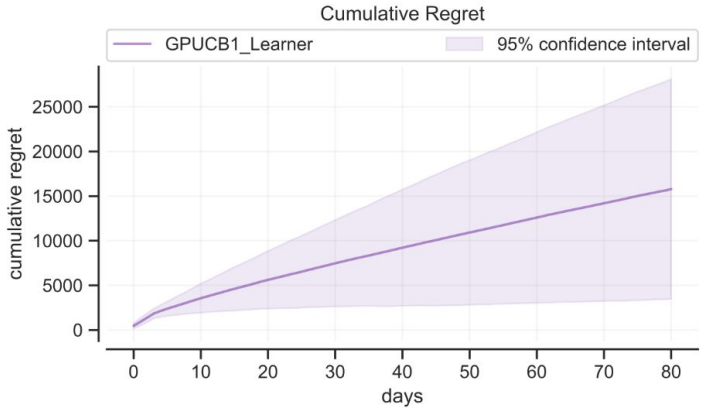




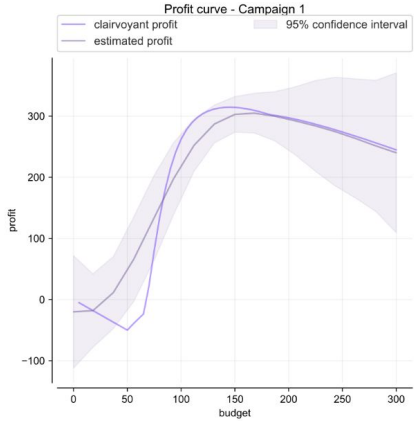
Step 3 - Confidence Intervals > GP-UCB1

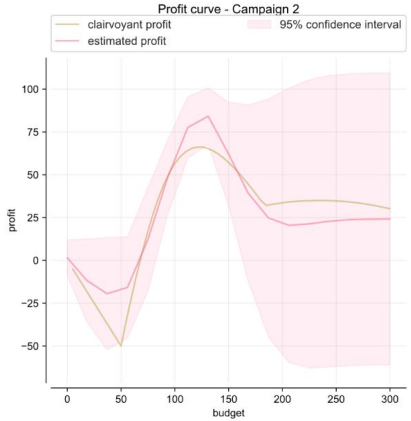


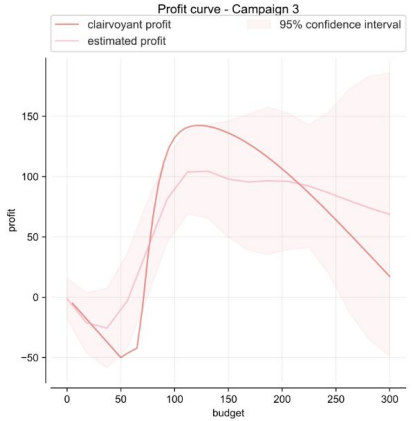


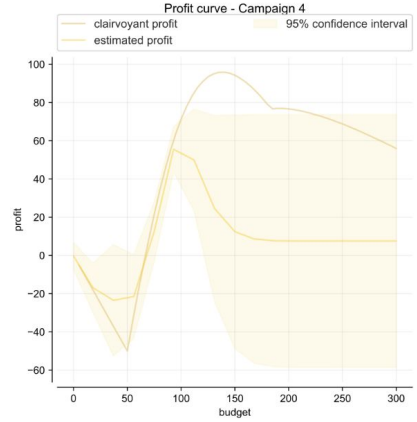


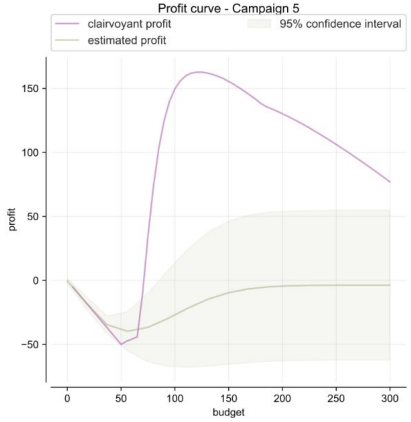
Step 3 - Estimated Profit Curve > G-TS











Gaussian Process Thompson Sampling Regret Bound

GP-TS regret at time **T**, where:

- **γ** is the information gain at time T. The regret bound does not hold in expectation but in high probability. A regret in high probability is weaker than a regret in expectation, because with small probabilities the bound might be violated.
- **M** is the number of arms
- **σ** is the maximum variance of the GP
- **Λ** > 0 is the Lipschitz constant
- **C** is the number of subsets (campaigns)

$$R_T \leq \sqrt{\frac{2\Lambda^2}{\log\left(1+\frac{1}{\sigma^2}\right)} CTB \sum_{k=1}^{C} \gamma_{k,T}} \quad \text{w.p. } 1-\delta$$

$$B = 8\log\left(2\frac{T^2MC}{\delta}\right)$$

GP-TS & GP-UCB1 Regret Bounds

Since the kernel of the Gaussian Process Bandits are squared exponential, the term $\sum_{i=1}^{C} \gamma_{i,NM_i}$ can be bounded by $\mathcal{O}\left(C \log(NM)^{(d+1)}\right)$, thus the following upper bound is provided:

$$\mathcal{R}_N(\mathfrak{U}) = \mathcal{O}\left(C\sqrt{N \log\left(CN^3M^3\right)(\log\left(NM\right))^{(d+1)}}\right)^{*}$$

Note that the bound scales linearly in the number of subsets **C**, but only logarithmically in the number of arms **M**.

* When Gaussian Processes Meet Combinatorial Bandits: GCB

Gaussian Thompson Sampling Regret Bound

$$\mathscr{R}_T(\mathrm{CGTS}) = O\left(C|M|\frac{\log(T)}{\Delta_{C,\min}}\right)$$

GTS regret at time **T**, where:

- **M** is the number of arms
- **C** is the number of subsets (campaigns)
- $\Delta_{C,\,min}$ is the Δ_{min} when considering all the campaigns

Step 4 - Optimization with uncertain α functions and number of items sold

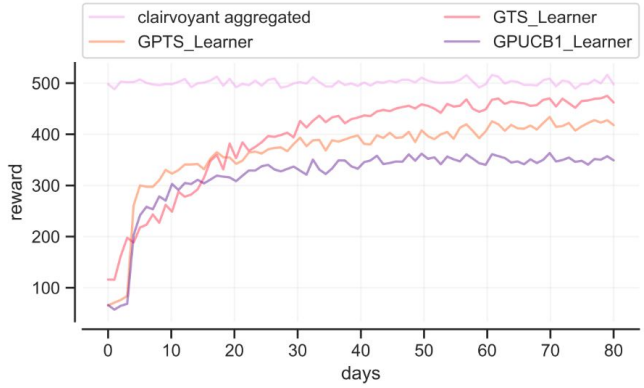
The approach is the same as in step 3, but now it is also unknown the number of items sold every time a user decides to buy a product.

The algorithm still works because the learning is bound to the reward received by the environment, without actually caring about the differences in this.

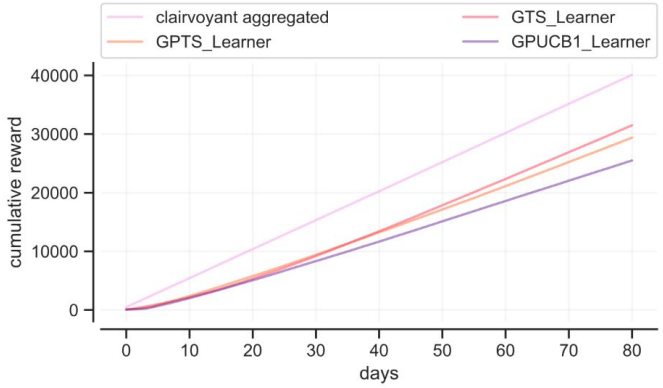
So the action the bandit takes is exactly the same as before and so is the update phase, with the only difference that the reward received will be even more noisy, so the regret will probably be larger.

Step 4 - Graphical results

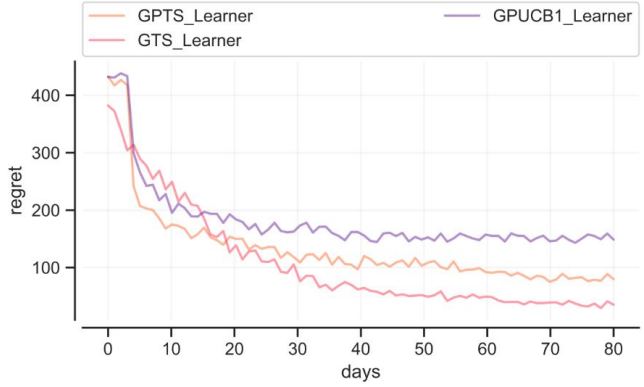
Reward



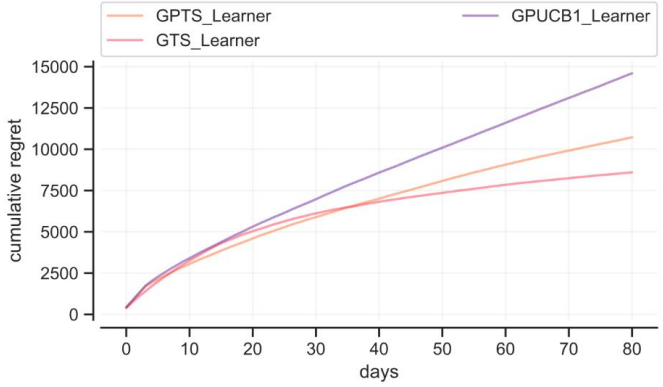
Cumulative Reward



Regret



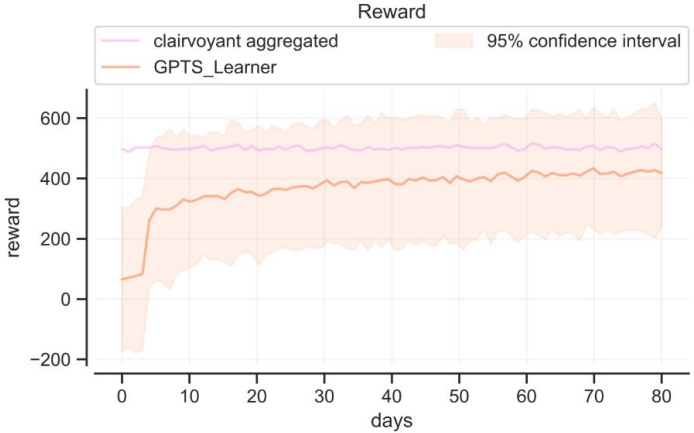
Cumulative Regret

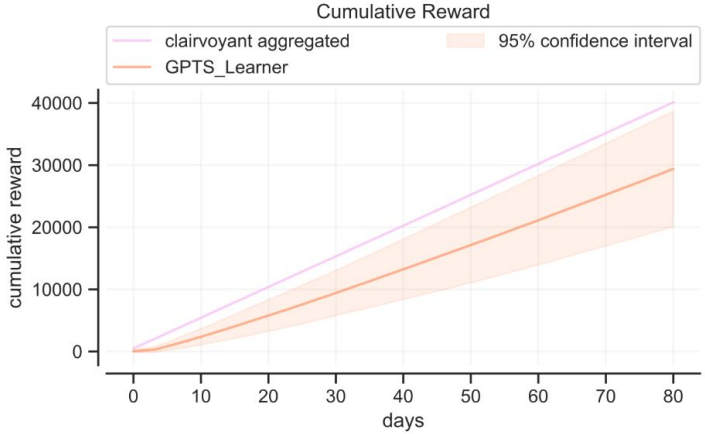


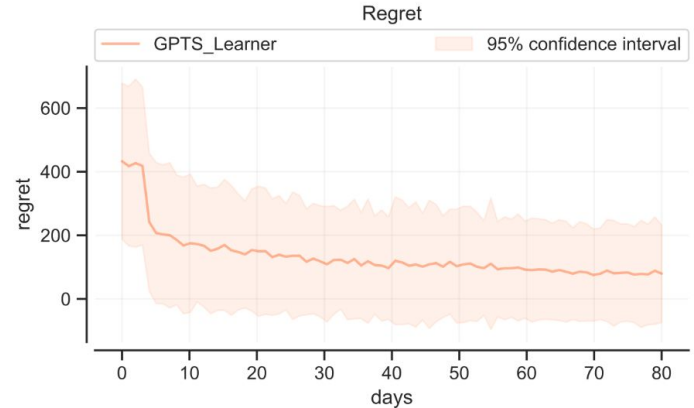
Step 4 - Numerical results

Bandit (combinatorial)	Average daily reward	Stad.dev on average daily reward	Average daily regret	Std.dev on average daily regret	Average cumulative reward	Std.dev on average cumulative reward	Average cumulative regret	Std.dev on average cumulative regret
Clairvoyant aggregated	501	71	0	0	40102	622	0	0
GP-TS	367	131	134	119	29380	4721	10722	4689
G-TS	393	138	107	126	**31497**	**1242**	**8606**	**1208**
GP-UCB1	319	119	182	113	25500	4011	14602	4049

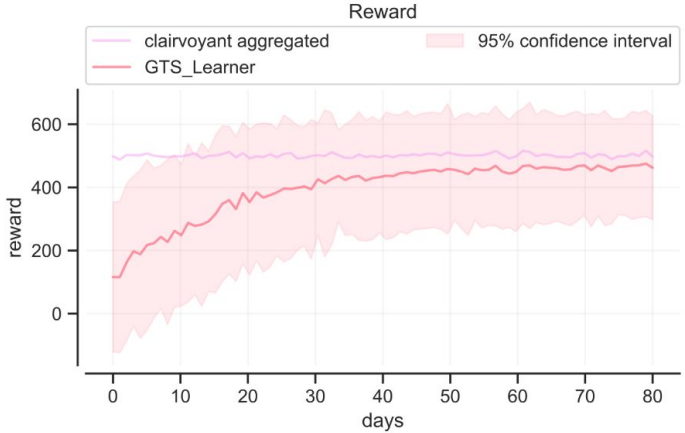
Step 4 - Confidence Intervals > GP-TS

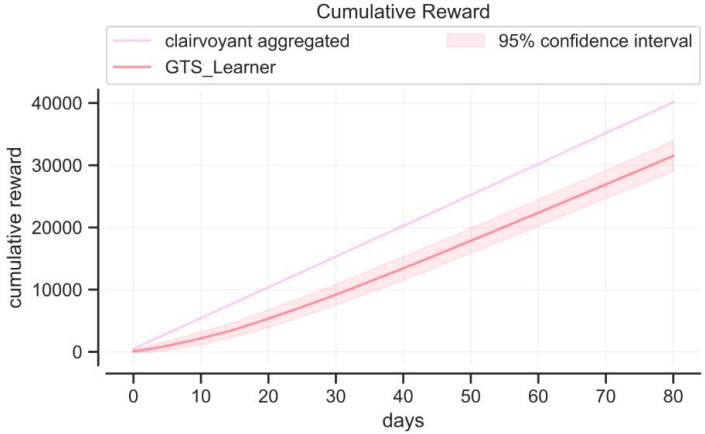


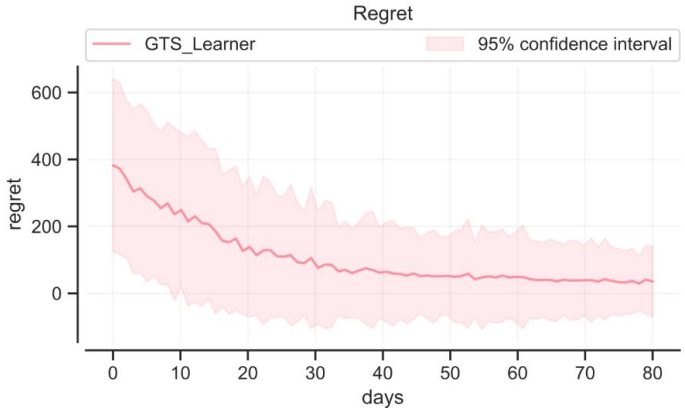


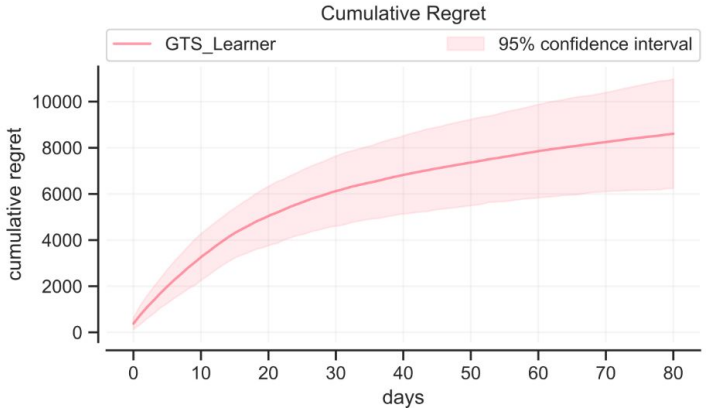


Step 4: Confidence Intervals > G-TS

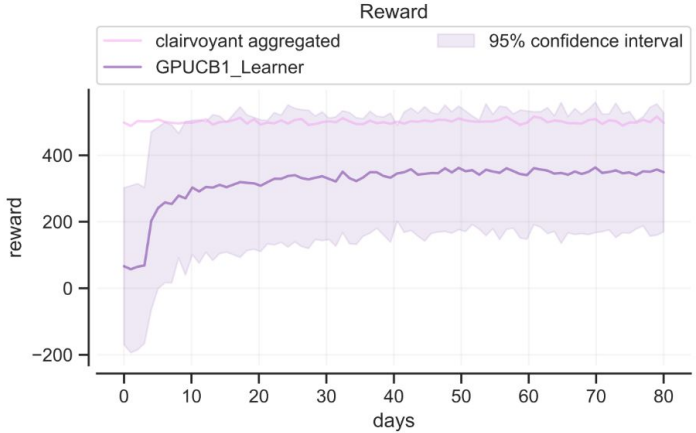


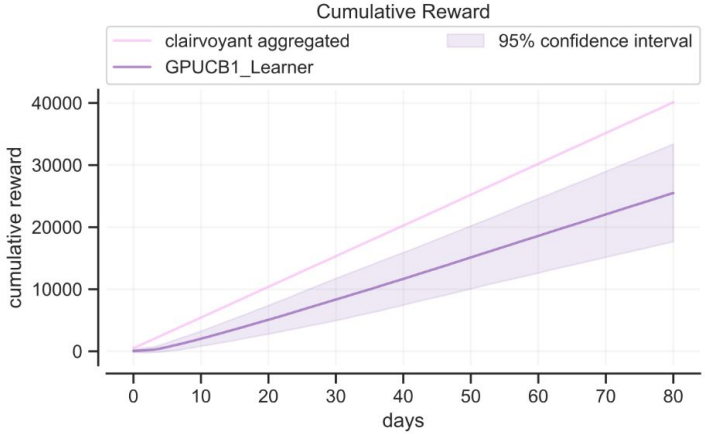


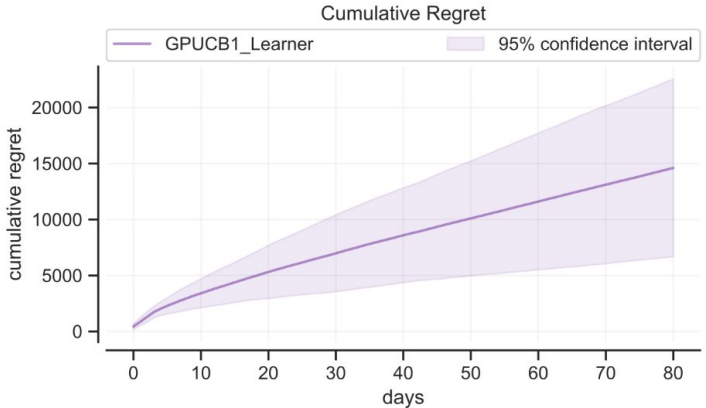




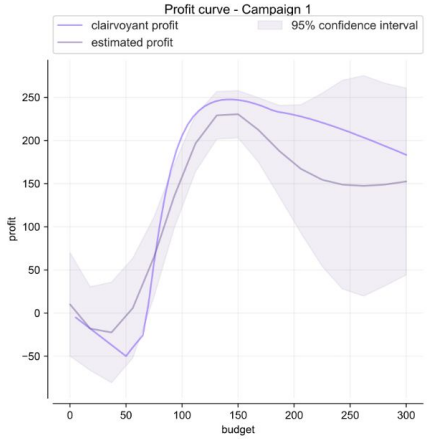
Step 4 - Confidence Intervals > GP-UCB1

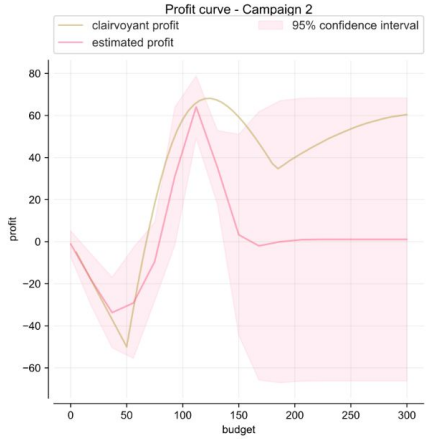


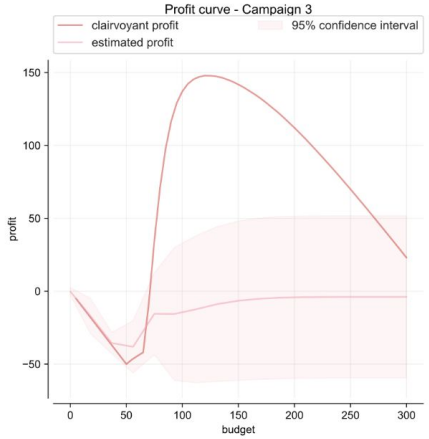


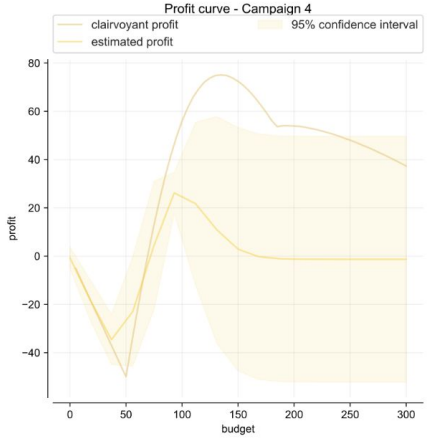


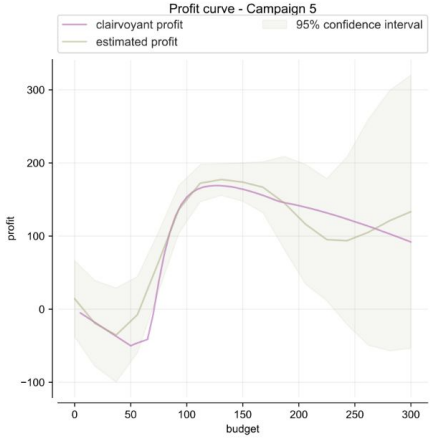
Step 4 - Estimated Profit Curve > G-TS











Step 3 & 4 : Conclusions

- Good performance for GTS and GP-TS that always outplay GP-UCB.
- GPTS works better than GTS when considering a larger number of arms, but in general the cumulative regret was higher than the reported case.
- GPTS gave the best results when tailored to have a proper prior information of the environment (smart restriction of arm range where to operate) and to choose arms close enough.
- In a real environment having access to good prior information and having a larger time window than 2 months over a stationary environment we would employ GPTS, in any other case GTS.
- The regret of step 4 **did not meet our expectations** as it was lower than step 3, however the profit was less

Step 5: Optimization with uncertain graph weights

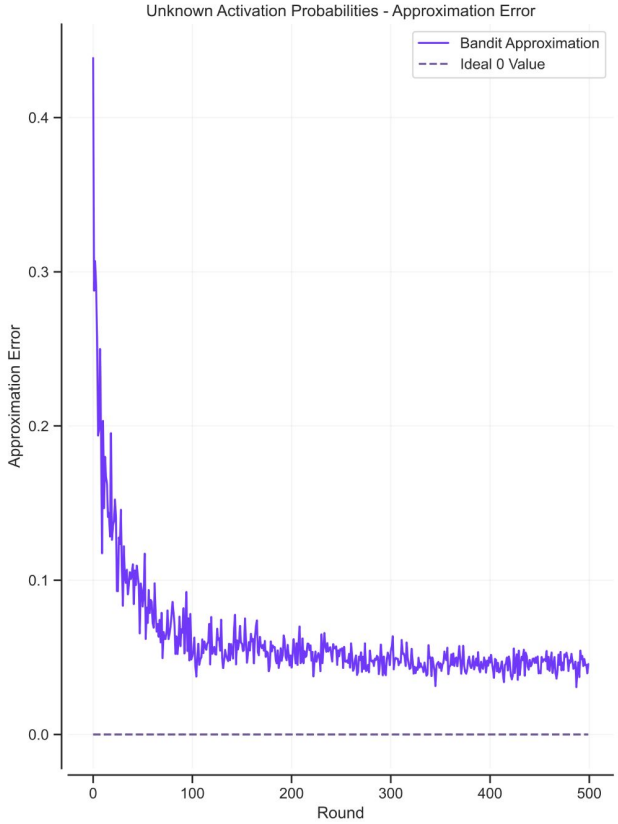
The approach for this step is the same as that of step 3, but now the unknown parameter is the graph weights, so those probabilities of clicking on the top secondary product once a user completes a purchase.

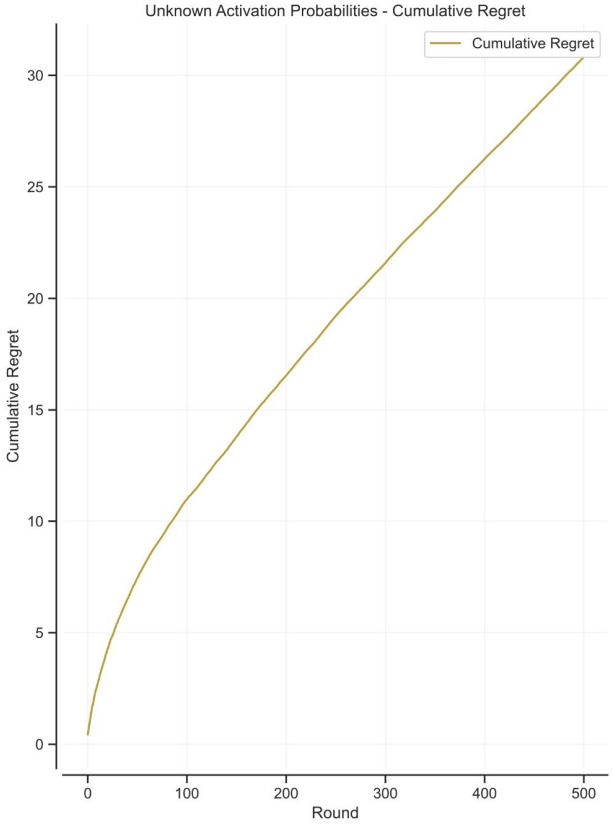
We employed both influence episodes and Monte Carlo Sampling to tackle the problem. Monte Carlo Sampling was employed to find the best seed for observing a large number of edges activating in an influence episode. The collected data about activations updates Beta parameters associated with each edge. The samples from the Beta distributions are used in the sampling process and provide the final estimated weights.

Step 5: Parameters

Simulations	500
Delta	0.2
Epsilon	0.1
Seeds (S)	1
Monte Carlo Repetitions	$R = \frac{1}{\epsilon^2}\log(\lvert S\rvert + 1)\log\left(\frac{1}{\delta}\right) \approx 30$

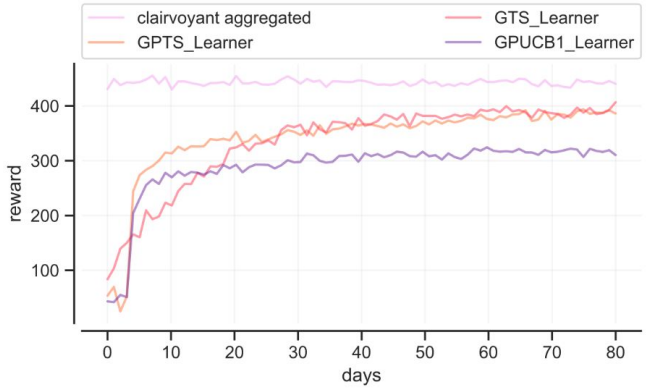
Weights estimation - User 2



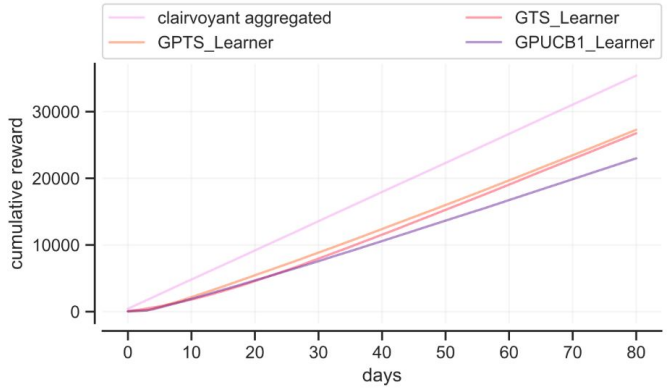


Step 5 - Graphical results

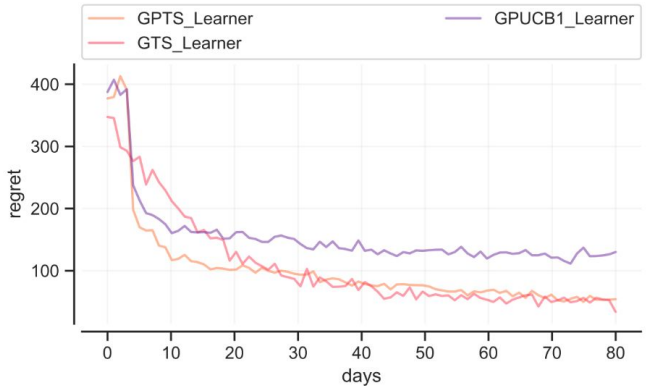
Reward



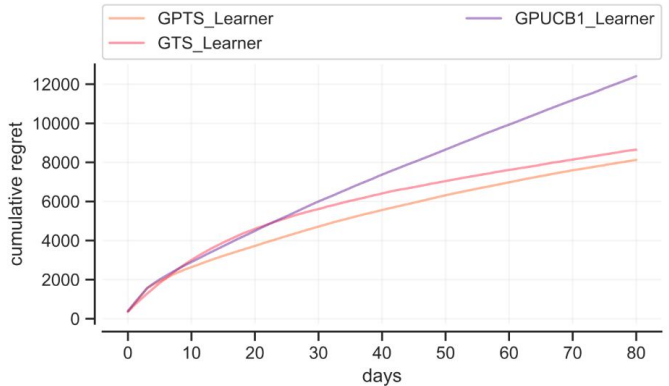
Cumulative Reward



Regret



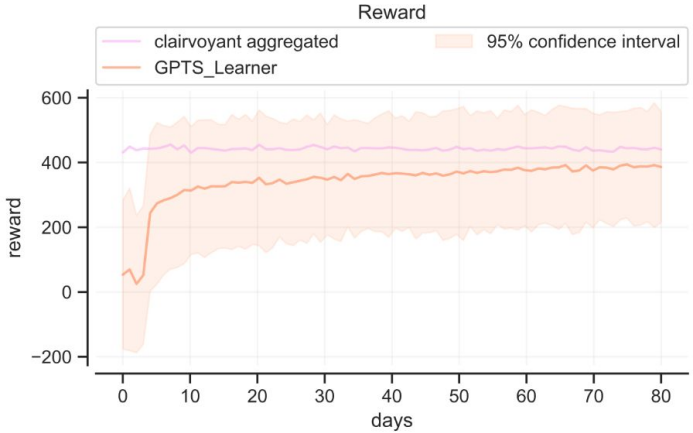
Cumulative Regret

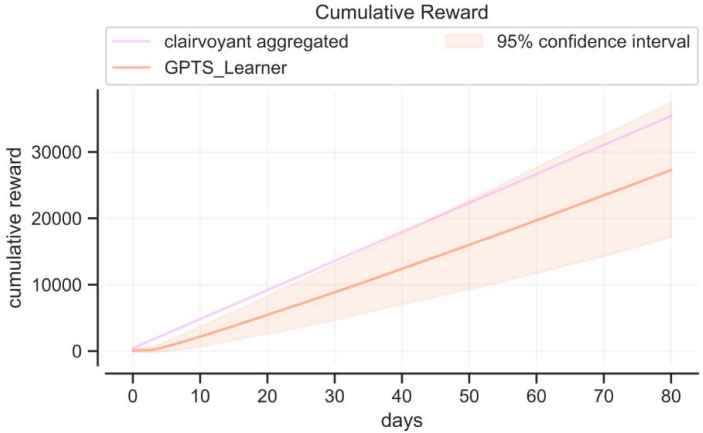


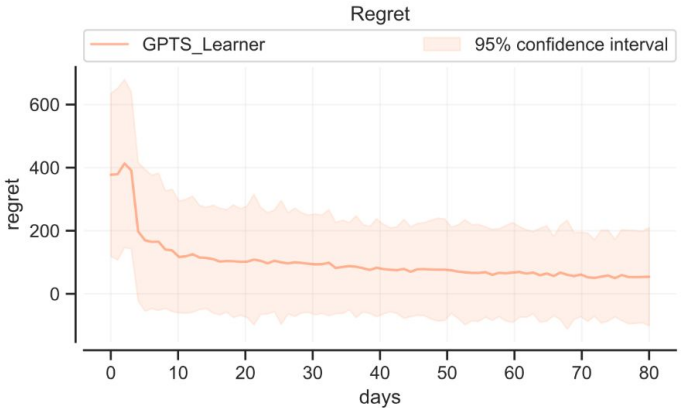
Step 5 - Numerical results

Bandit (combinatorial)	Average daily reward	Stad.dev on average daily reward	Average daily regret	Std.dev on average daily regret	Average cumulative reward	Std.dev on average cumulative reward	Average cumulative regret	Std.dev on average cumulative regret
Clairvoyant aggregated	443	73	0	0	35415	4400	0	0
GP-TS	**341**	**122**	102	111	**27286**	5165	8128	4530
G-TS	335	129	108	117	26767	**4052**	8647	2567
GP-UCB1	288	119	155	112	23002	5369	12413	4897

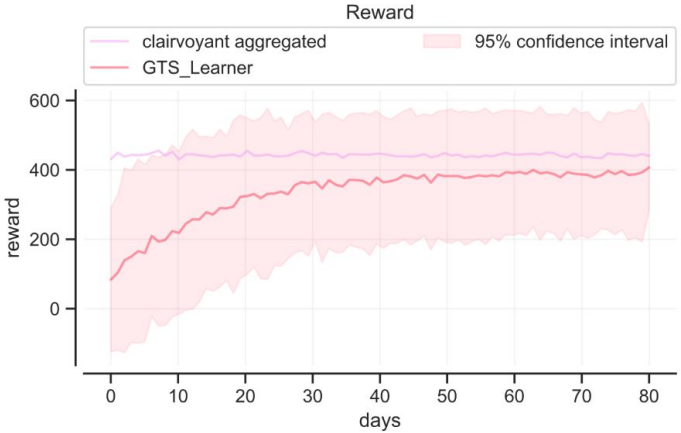
Step 5 - Confidence Intervals > GP-TS

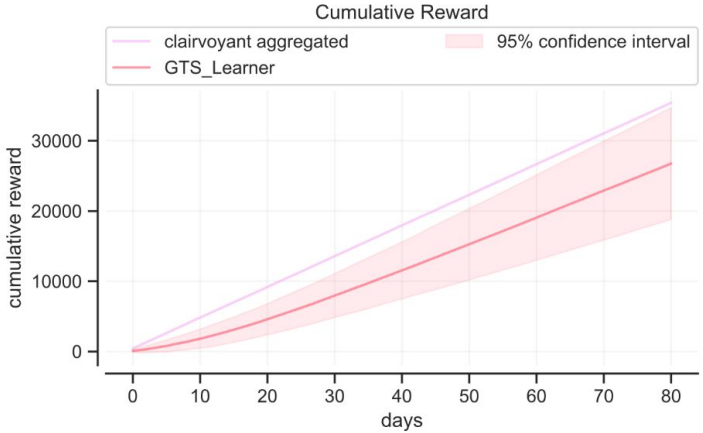


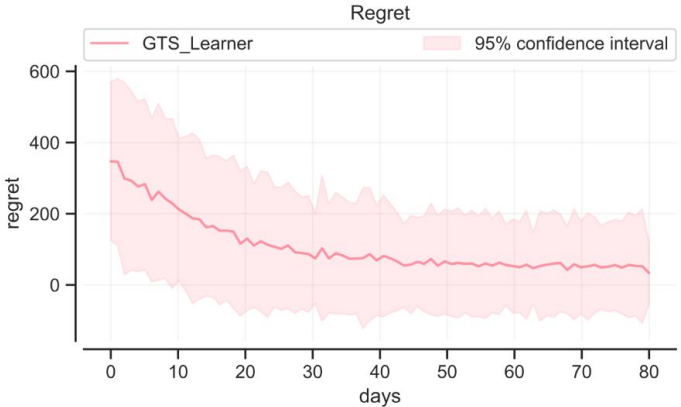




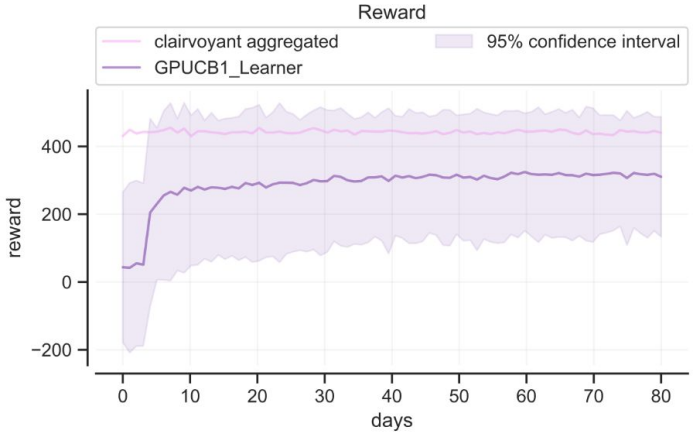
Step 5 - Confidence Intervals > G-TS

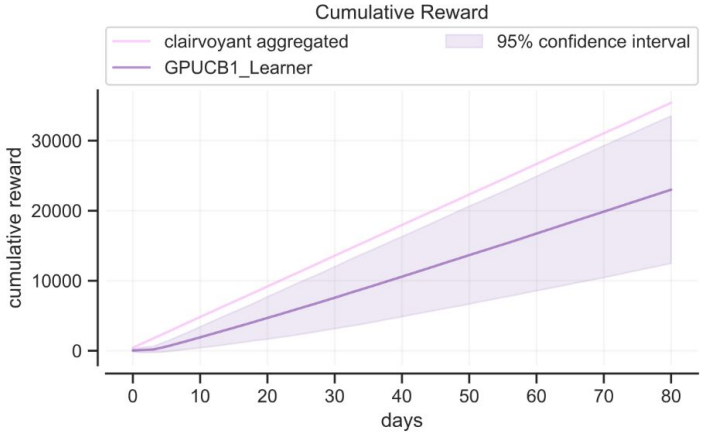


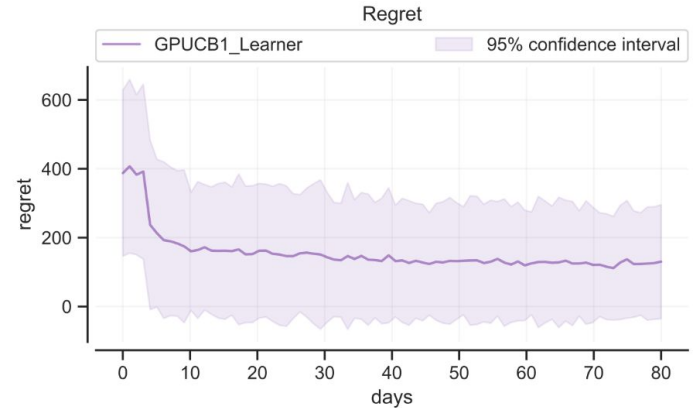


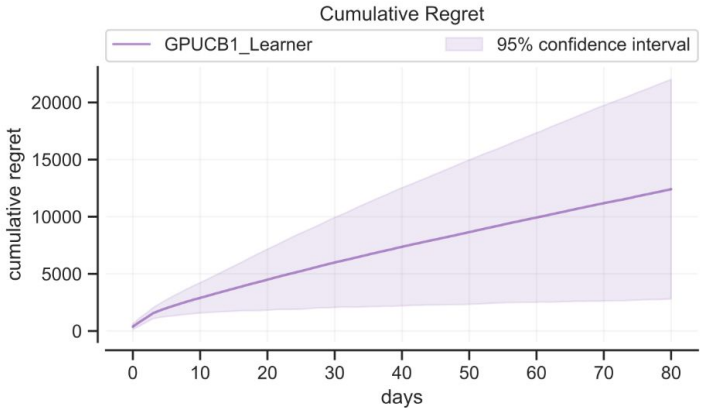


Step 5 - Confidence Intervals > GP-UCB1

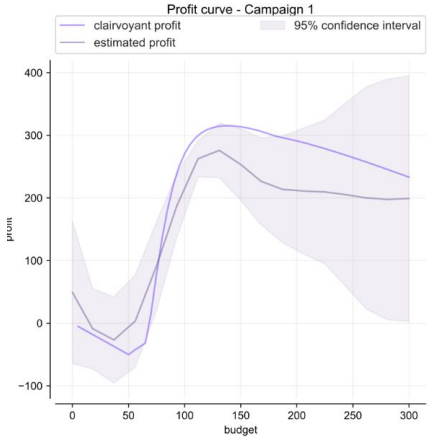


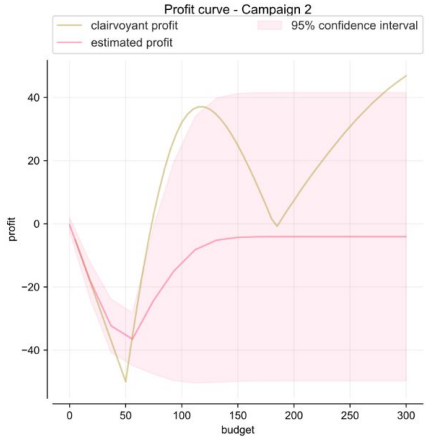


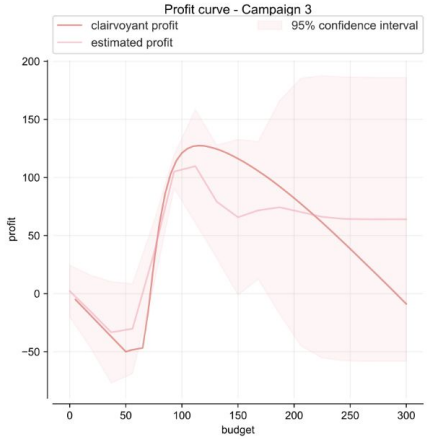


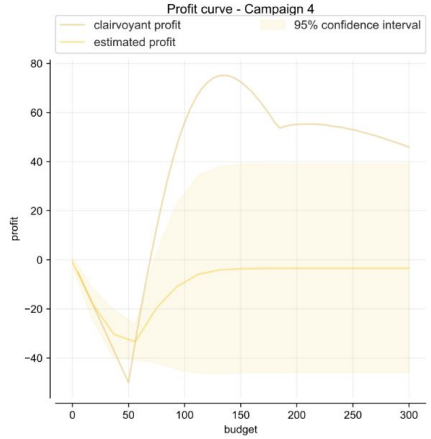


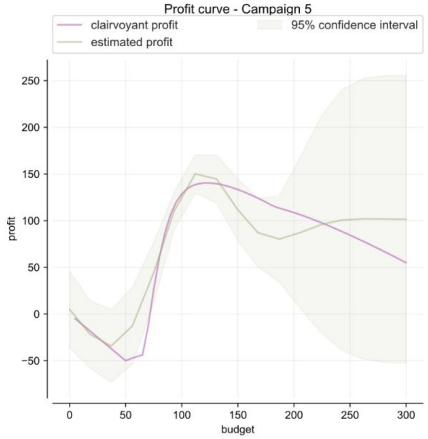
Step 5 - Estimated Profit Curve > G-TS











Step 5 - Conclusions

- In the short run, the GP-TS has the best performance, after a while it is reached by the GTS learner.
- GP-UCB performs again worse than the others.
- As expected the profit is much lower with respect to the case of the known graph
- It shows how a good estimation of unknown quantities is needed in practice, and overall the estimations given by the monte-carlo sampling were acceptable

Step 6 - Non-stationary demand curve

In this step the demand curve is subject to abrupt changes, meaning that it is composed of phases that follow each other through time.

In particular we decided to change the number of users and the user probability in order to simulate a non-stationary environment.

It is requested to compare a UCB solution exploiting a Change Detection algorithm and a Sliding-Window algorithm.

Step 6 - Abrupt changes

Context	Season	Number of users
Average e-commerce traffic	1 april - 30 april	350
New competitor enters the marker	1 may - 28 may	175
Competitor turns out to be of lower quality & more expensive	29 may - 20 june	700

Step 6 - SW & CUSUM Parameters

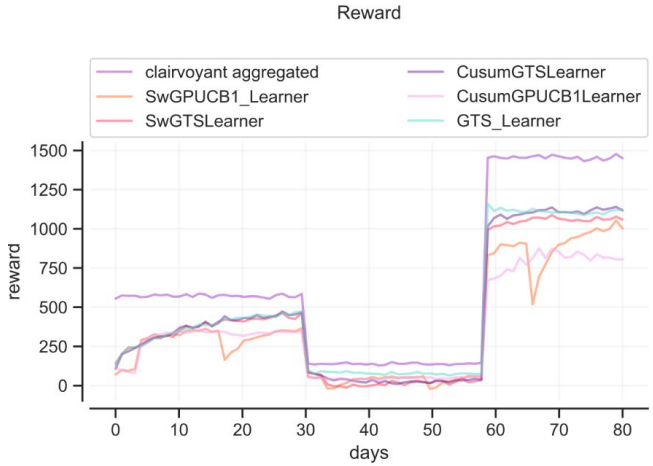
Sliding Window

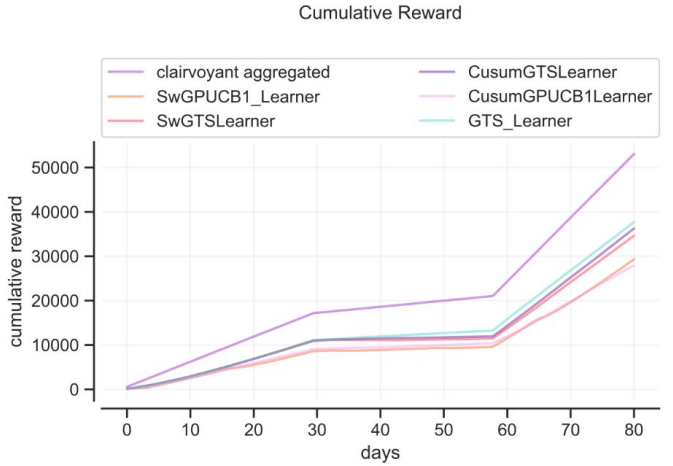
Window Size = $\lceil (\sqrt{T} + 0.1T) \rceil$

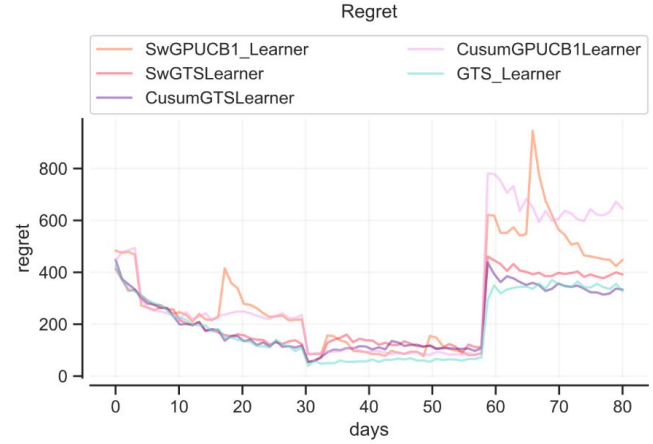
Change Detection CUSUM

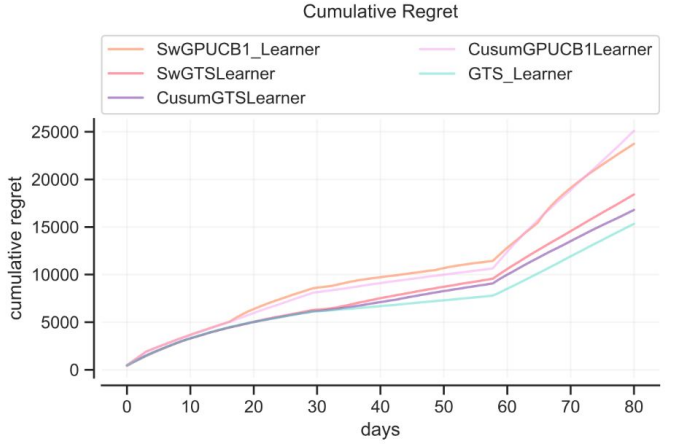
Samples for reference point	10
Epsilon	0.05
Detection threshold	200
Exploration factor	0.01

Step 6 - Graphical results





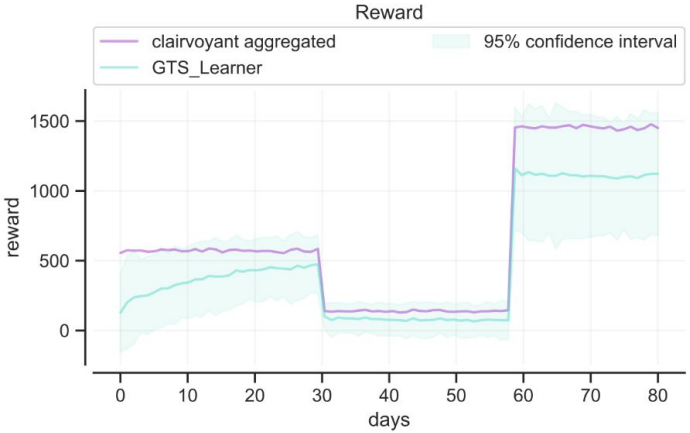


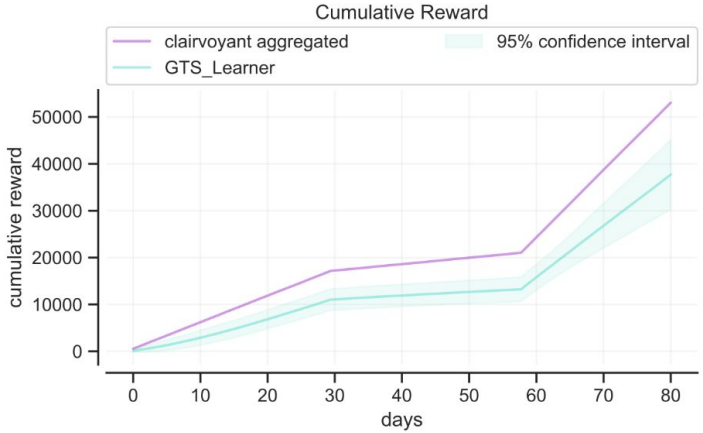


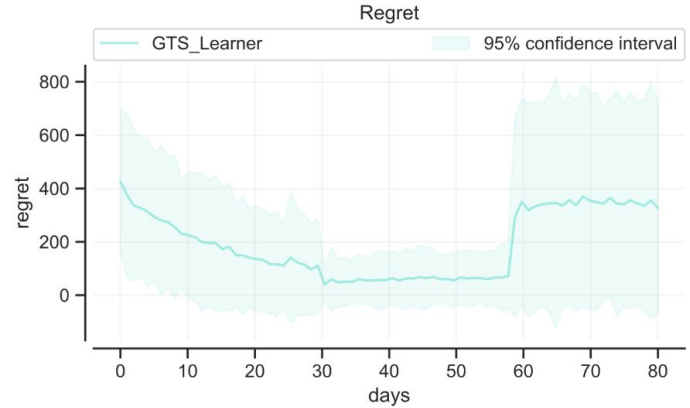
Step 6 - Numerical results

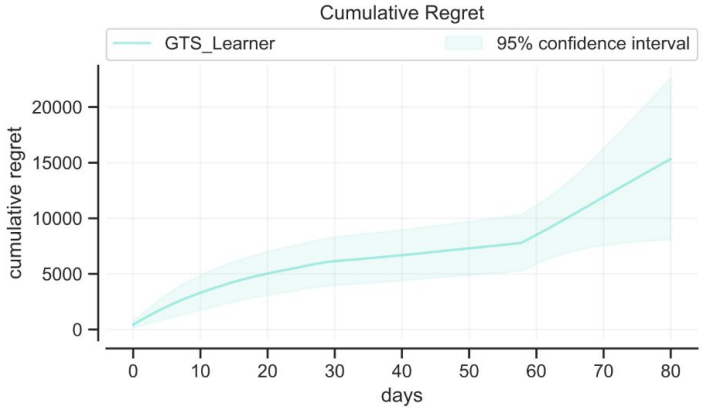
Bandit (combinatorial)	Average daily reward	Stad.dev on average daily reward	Average daily regret	Std.dev on average daily regret	Average cumulative reward	Std.dev on average cumulative reward	Average cumulative regret	Std.dev on average cumulative regret
Clairvoyant aggregated	663	530	0	0	53041	861	0	0
SW-GP-UCB1	366	399	297	269	29294	5230	23747	5172
SW-G-TS	433	447	230	206	34620	5630	18421	5548
CUSUM-GP-UCB1	349	**364**	314	308	27937	6060	25104	6076
CUSUM-G-TS	453	460	210	192	36245	4660	16797	4737
G-TS	**471**	442	192	182	**37710**	**3767**	15331	3702

Step 6 - Confidence Intervals > G-TS

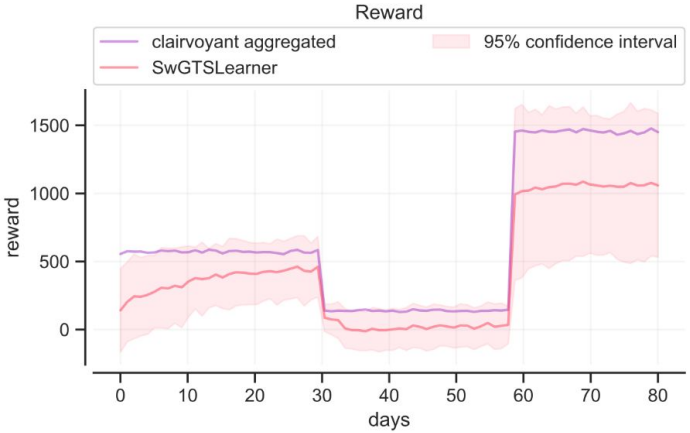


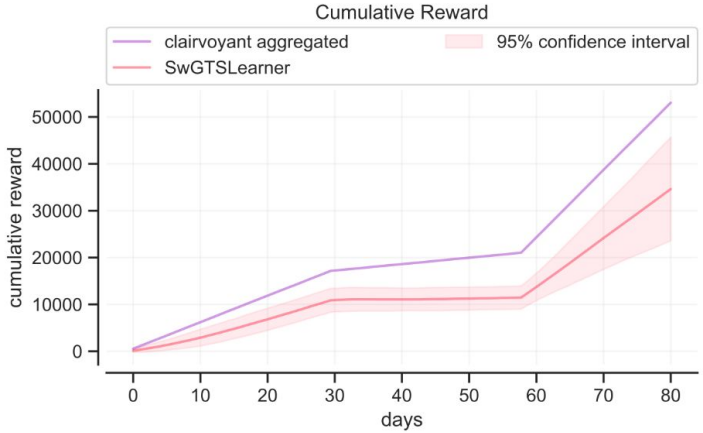


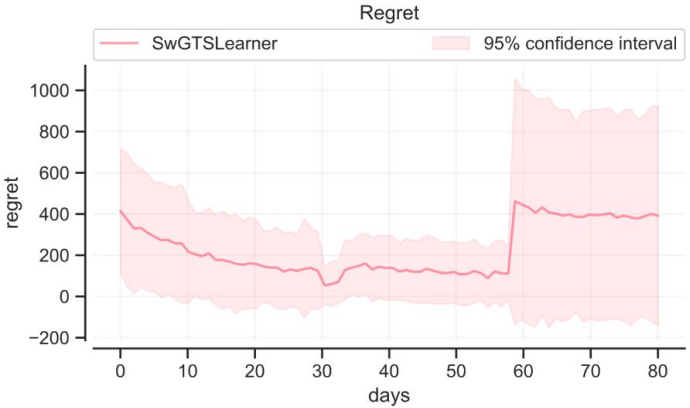


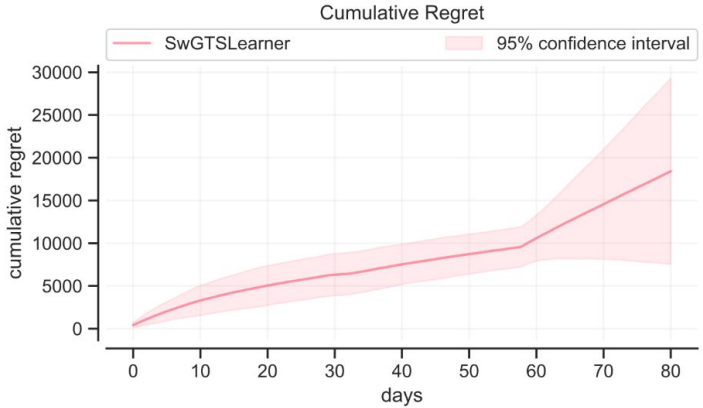


Step 6 - Confidence Intervals > SW-GTS

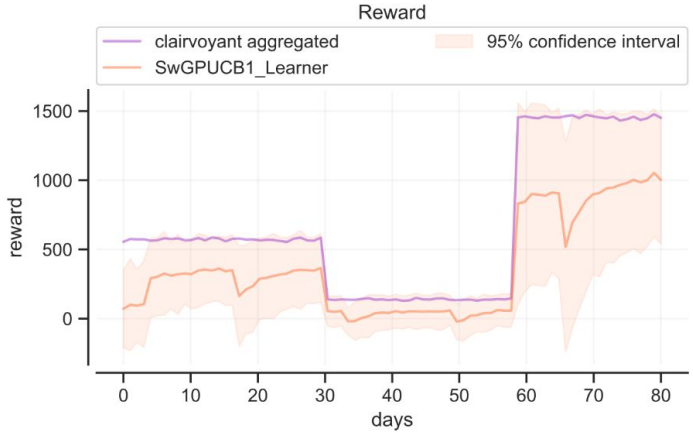


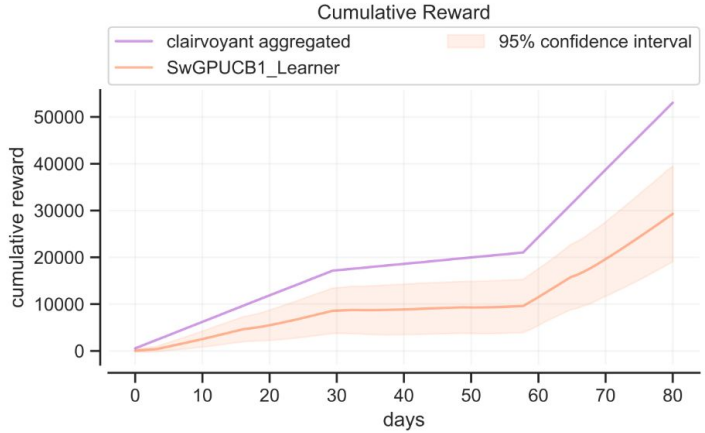


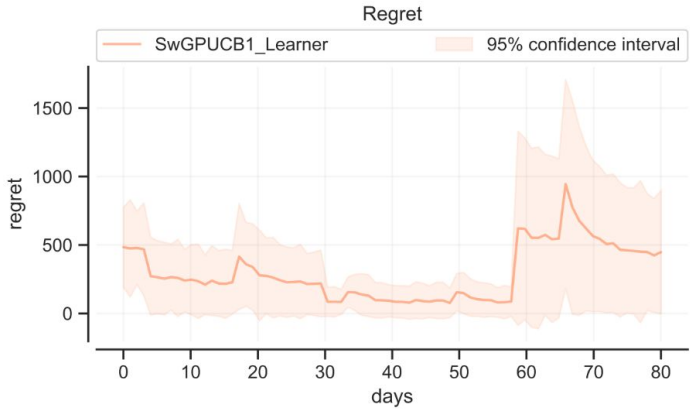


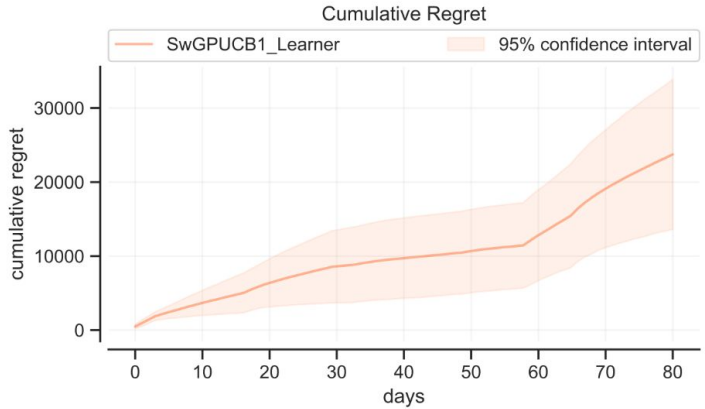


Step 6 - Confidence Intervals > SW-GP-UCB1

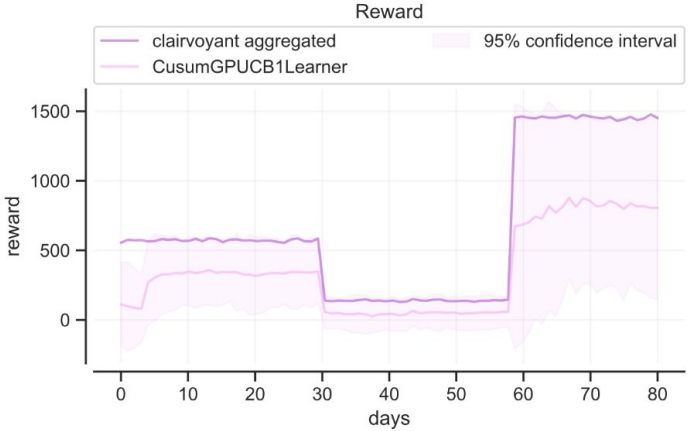


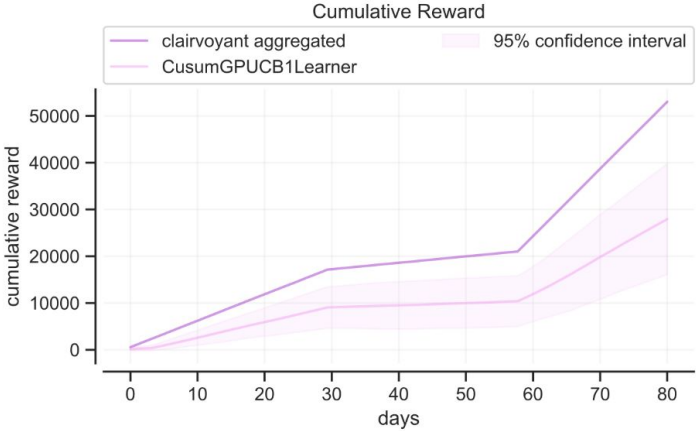


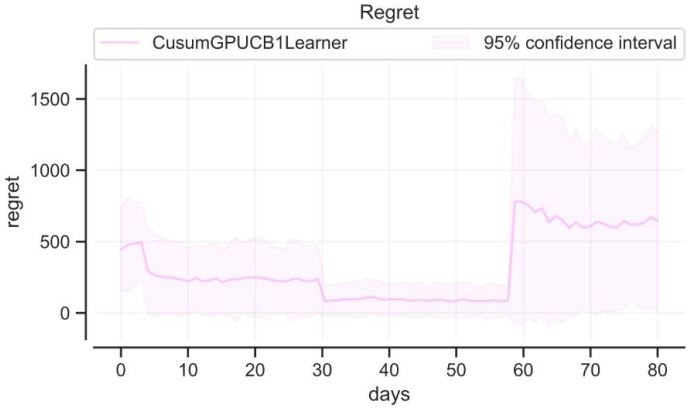


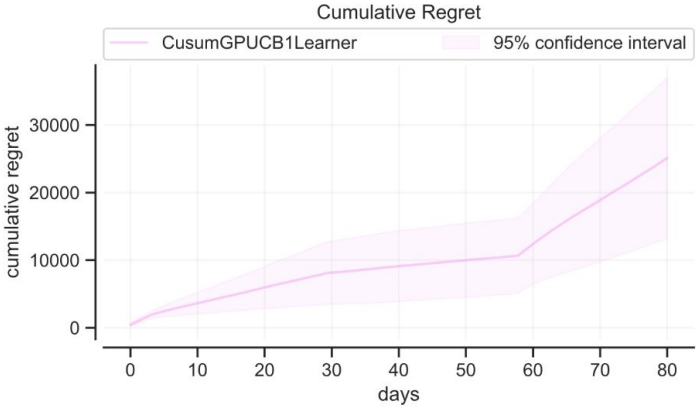


Step 6 - Confidence Intervals > CUSUM-GP-UCB1

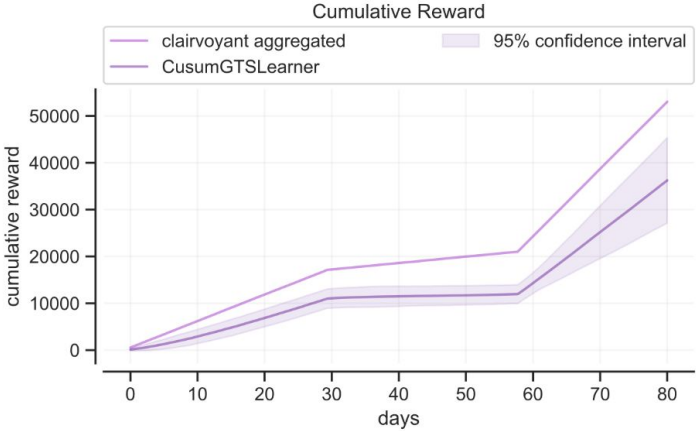


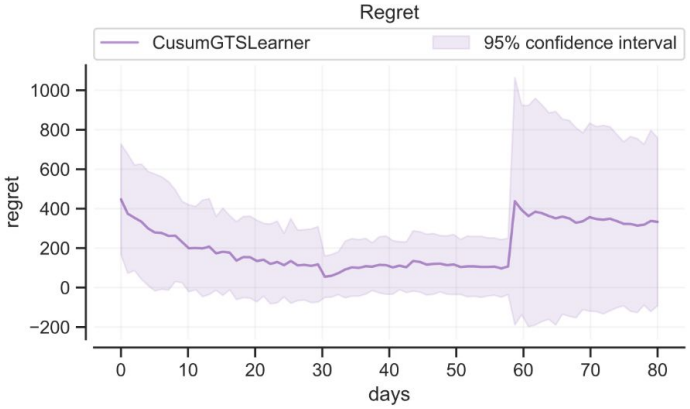


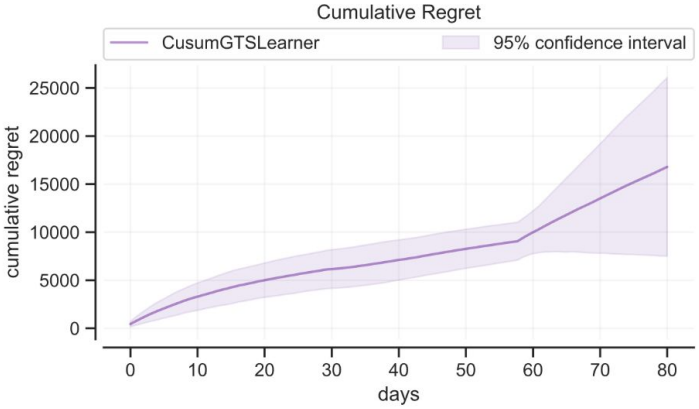




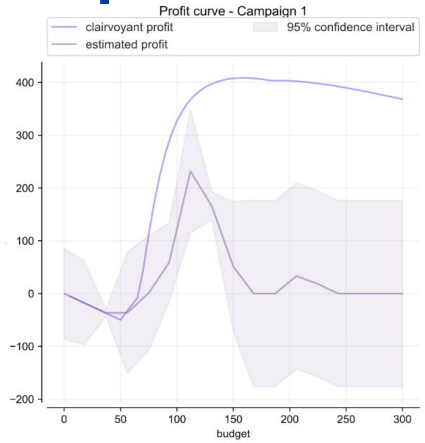
Step 6 - Confidence Intervals > CUSUM-GTS

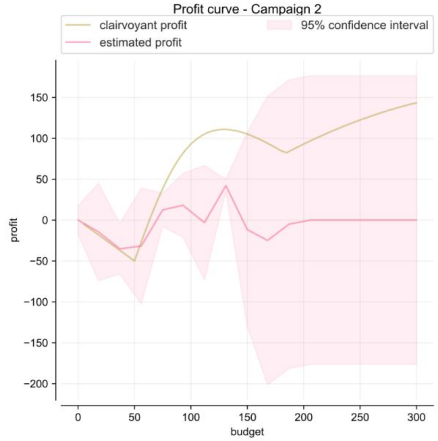


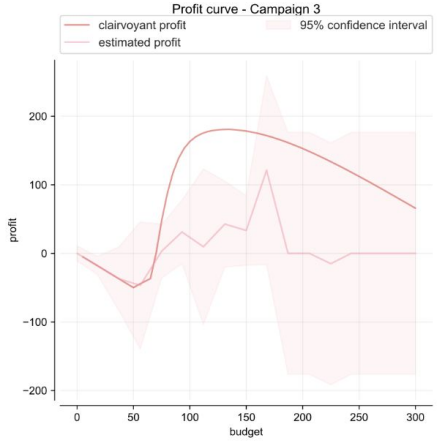


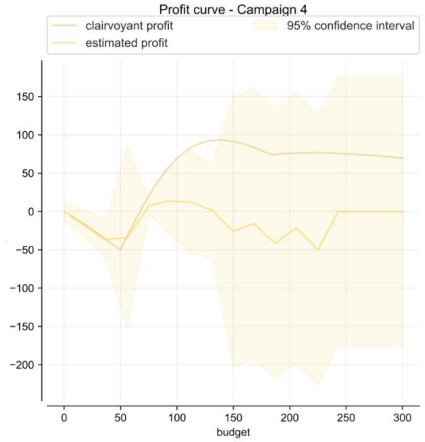


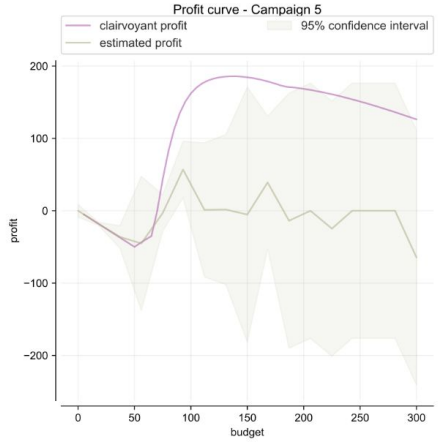
Step 6 - Estimated Profit Curve > SW-GTS











Sliding Window-TS for abrupt changes

In a sliding window TS for abrupt change, with $X_{i,t}$ ~ Be(μ_i,t), for every $\tau \in N$, the pseudo-regret would be:

$$\bar{R}_N(\mathfrak{U}) \leq \sum_{i=1}^{K} \left[\tau B N^{\alpha} + \sum_{\phi=1}^{B_N} \Delta_{i,\phi} \frac{N_\phi}{\tau} \left(\frac{52 \log \tau}{\Delta_{i,\phi}^2} + \log \tau + 5 + \frac{19}{\log \tau} \right) \right]$$

where:
B and α are defined according to the following assumption:

There exists α ∈ [0, 1], independent of N, s.t. the number of breakpoints BN is of order O(Nα). That is, there exist α ∈ [0, 1) and B ∈ R + such that: BN ≤ BNα.

$\Delta_{i,\varphi} := \mu_{i*,\varphi} - \mu_{i,\varphi}$ is the difference between the expected reward $\mu_{i*,\varphi}$ of the best arm $a_{i*,\varphi}$ and the expected reward $\mu_{i,\varphi}$ of arm ai during phase F_φ

Step 6 - Conclusions

- In this experiment we can notice that the sliding window GP-UCB suffers the highest regret, while the GTS is the learners that performs better.
- In general, the two sliding window algorithms (SW-GP-UCB and SW-G-TS) provide worse performance than they could have. The reasons are mainly related to the window size and the Bandit memory reset, which slows the learning process.
- CUSUM-GP-UCB and SW-GP-UCB offer the worst performance, although CUSUM-GP-UCB has also the less uncertain prediction.
  Both CUSUM algorithms can be improved by choosing more accurately the parameters of the change detection algorithm.

Step 7 - Context generation

During this step we added the possibility to dynamically adapt the simulation to all the contexts that can be generated using the 2 binary features

A context is defined as a list of 4 bits: 1 feature considered, 0 not considered

We could have 4 possible of contexts:

- [[1, 1, 1, 1]] (fully aggregated)
- [[1, 1, 0, 0], [0, 0, 1, 1]] or [[1, 0, 1, 0], [0, 1, 0, 1]] (family or worker split)
- [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]] (fully disaggregated)

The context generation algorithm has been developed has flexible as possible to try easily all possible splits

Step 7 - Context generation

The same structure of the combinatorial learner has been kept, the only difference was in the number of learners employed:

- 5 learners (previous steps learners)
- 10 learners (1 split)
- 20 learners (2 split)

The number of learners grows exponentially as 5^n_splits

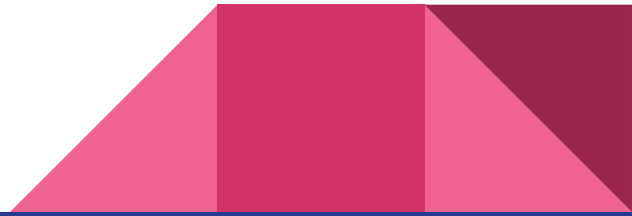
In the simulation the breakpoints activating the splits can be setted

The main advantage of performing the split is that the budget can directed with more accuracy over the best users, the drawback is that convergence is slower since the problem is harder

Step 7 - Context generation

Split techniques developed:

- Copying the learners instances of the previous context split and doubling them: it didn’t work because the exploration was too little since the learners where already exploring less and the exploitation what learned so far, the optimal budget array was usually more and more sparse after every split
- Doubling and restarting the learning process gave the better results especially with GP learners since the learners where really quick in spotting the bad campaigns giving a negative slope profit



Step 7 - Context generation

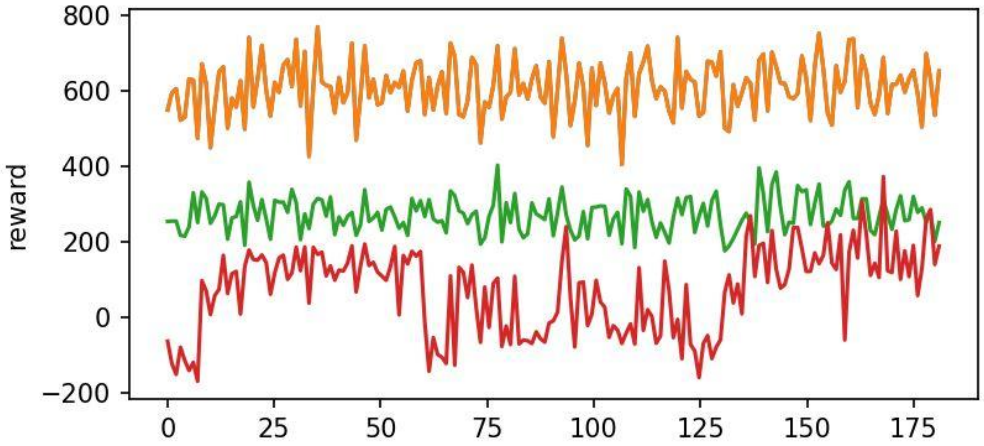
The decision of the split yes/no has been performed using the Hoeffding lower bound of the 2 cases, where x is the mean profit value

$$\bar{x} - \sqrt{-\frac{\log(\delta)}{2|Z|}}$$

where δ is the confidence and Z is the set of data.
The confidence used was 0.8 since the learners with more contexts to consider were slower to converge using too high confidence interval would cause the split case to be always worse or the need to spend too much time testing a contextual split.

Step 7 - Context generation

Average behavior observed using GPTS with context generation length 60 days with breakpoints at 60 and 120.



The second split passed the Hoeffding lower bound test with 125.41 > 92.51

Step 7 - Context generation

The spotted pattern was that the disaggregated case was the best, the convergence slower but better on the long run, however in real scenarios we may not be able to afford such long periods of training.

The first splitting was less likely to pass the the splitting condition, the initial regret was not worth the gain.