



POLITECNICO
MILANO 1863

OLA Project - Report

Mattia Portanti - 10571436

Matteo Fabris – 10623973

Luca Venturini - 10609456

Samuele Portanti - 10571388

Luca Alessandrini - 10569363

Social Influence
and
Advertising

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm with all the parameters known

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 1: Environment

Motivating Application

Online E-Commerce Website that sells clothes

It could sell clothes to:

Adults / Children
Male / Female

These are our **two Binary Features**

We can divide them in the **3 classes**:

Adult / Male

Adult / Female

Children / Male
Children / Female

Adult / Male	00
Adult / Female	01
Children / Male	10
Children / Female	11

Step 1: Environment

Motivating Application



Children / Male
Children / Female

Children Male and Children Female could be merged to

“CHILDREN”

since our children business is basically the same for both the classes!

Step 1: Environment

Probability Distributions

Reservation Prices

for each product and for each class

each class has its reservation price for each product

It is a **Gaussian Distribution** with a Certain Mean and a Certain Standard Deviation

Quantity Bought

for each product and for each class

each class will buy a certain amount of each product

It is a **Gaussian Distribution** with a Certain Mean and a Certain Standard Deviation

Number of Users per Class

for each class how many users visits our website

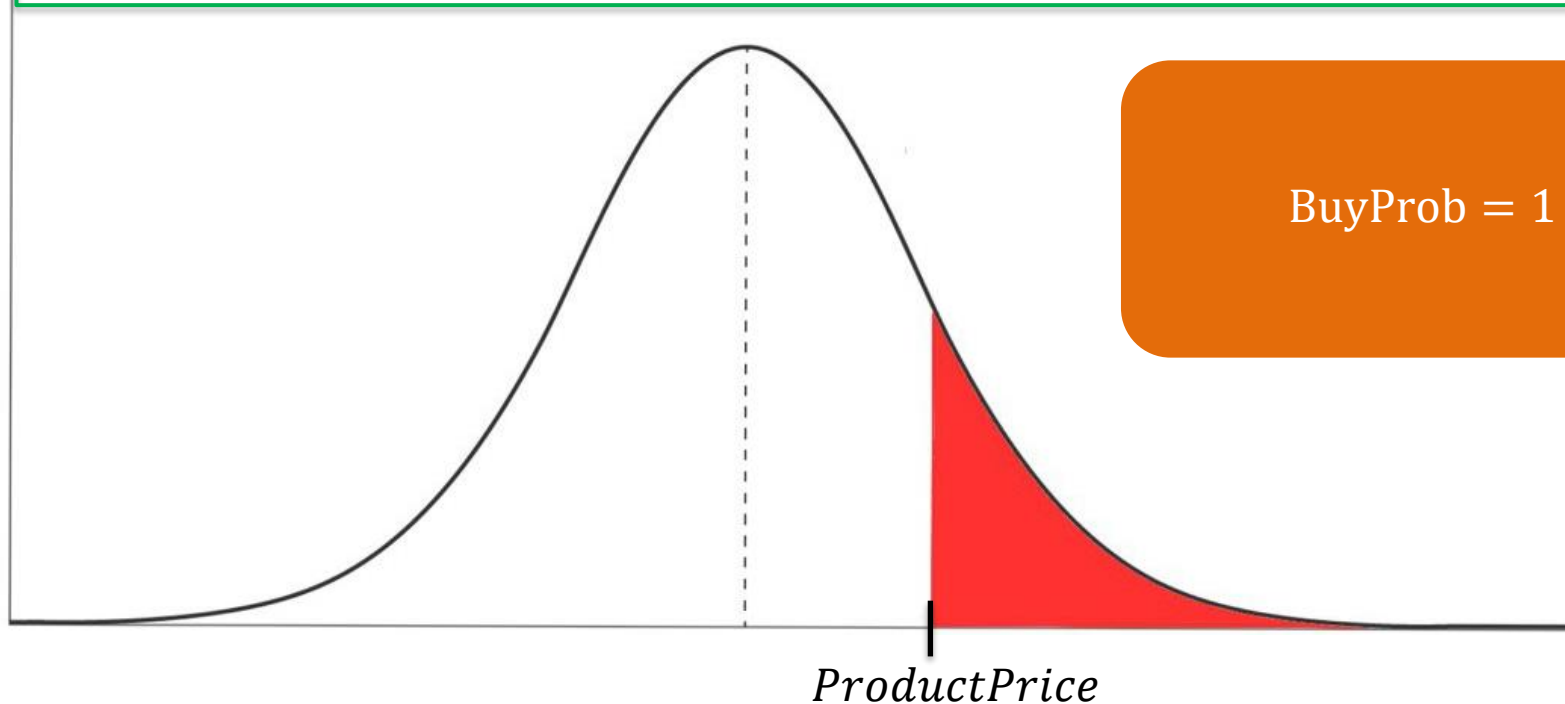
It is a **Gaussian Distribution** with a Certain Mean and a Certain Standard Deviation

Step 1: Environment

Probability Distributions – Example on the Reservation Price

Reservation Prices

The **RESERVATION PRICE** is used to calculate the buy probability, that is the willingness of the user to buy a product at a determined price.

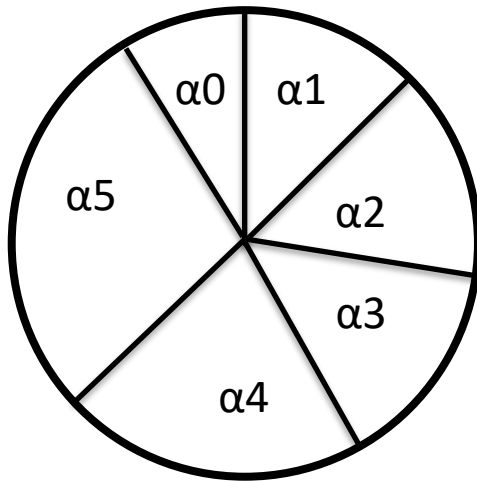


$$\text{BuyProb} = 1 - \left[\int_{-\infty}^x N(\mu, \sigma) dx \right]_{x=\text{ProductPrice}}$$

Step 1: Environment

α - ratios

Ratio of customers Landing on a Webpage in which Product i is primary



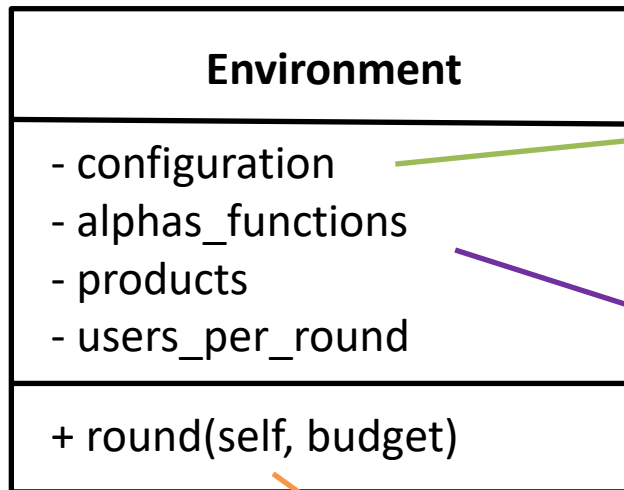
α_i ; $i \in [0, \dots, 4]$ \rightarrow product $i+1$ (1...5)
 α_5 \rightarrow competitors

Every class is characterized by a profile of α functions one per campaign

so, alpha ratios will be: **6 x (number of classes)**

Step 1: Environment

The Environment class



configuration
contains all the parameters read from a configuration file needed for the simulation

alphas functions
They are a list of *functions* that return the increment on the weights, that will impact on the alpha ratio (percentage of users landing on a product) given the allocated budget of that round

round()
This method executed a round of the simulation corresponding to one day in the real world

Step 1: Environment

The Environment class

configuration

contains all the parameters read from a configuration file needed for the simulation

- number of users per class mean
- number of users per class standard deviation
- base alpha functions (starting point)
- products
 - name
 - price
 - number
 - first secondary & second secondary
- lambda value
- graph click matrix 5x5
- reservation price mean
- reservation price standard deviation
- quantity of product bought per class mean
- quantity of product bought per class standard deviation

`/configurations/environment/static_conf_xxx.json`

Step 1: Environment

The Environment class

alphas_functions

They are a list of *functions* that return the increment on the weights, that will impact on the alpha ratio (percentage of users landing on a product) given the allocated budget of that round

$$\gamma_1 \cdot (1 - e^{-\gamma_2 \cdot budget})$$

Where: γ_1 and γ_2
are parameters to be tuned for the simulation

We will have a pair of γ_1, γ_2 for each pair of **(product, user class)**

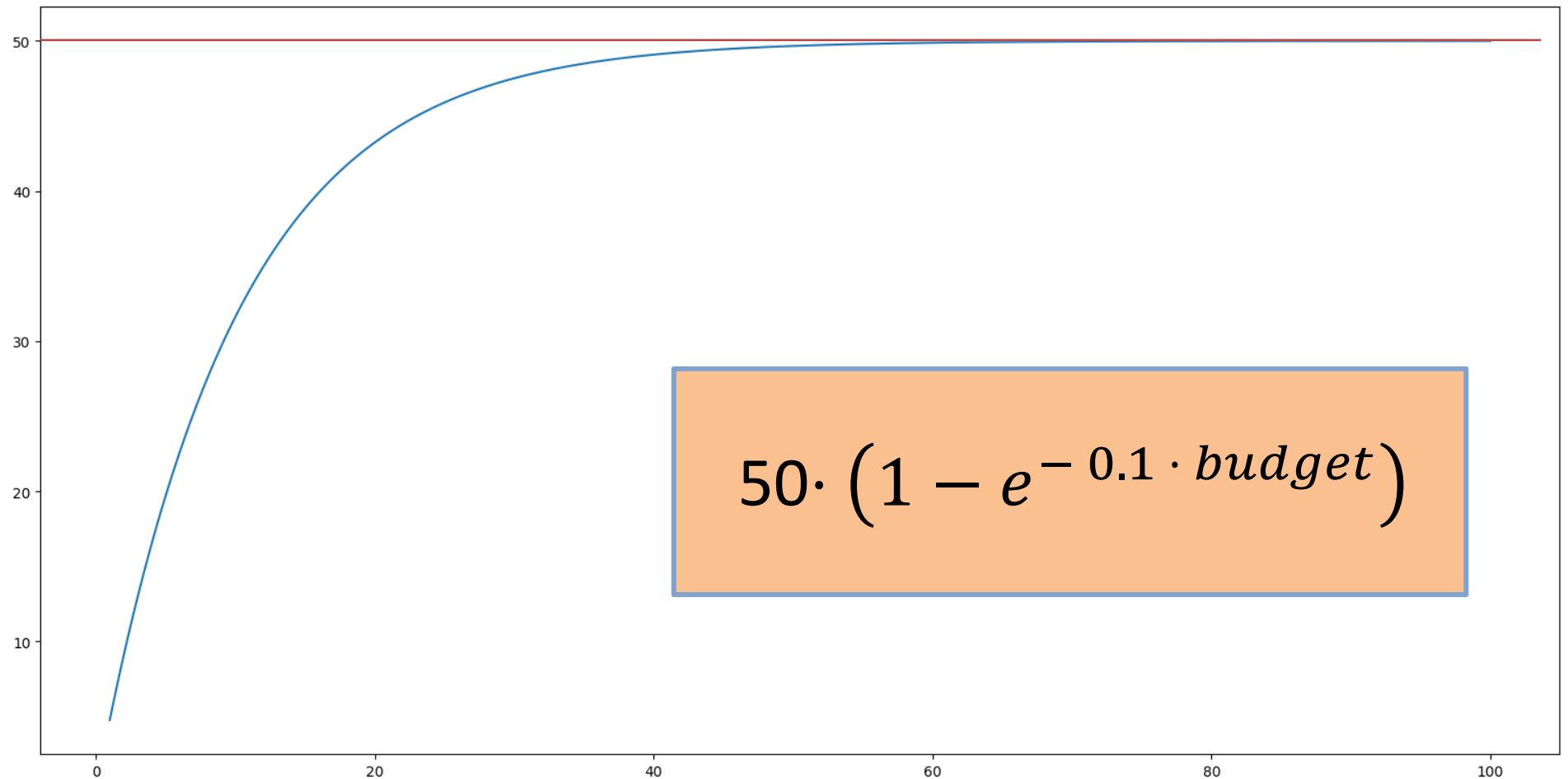
These **weights** will be used to create the Dirichlet distribution.

The sum of these weights will start from a certain value (fixed in the configuration file) and will be constant over the simulation.

Step 1: Environment

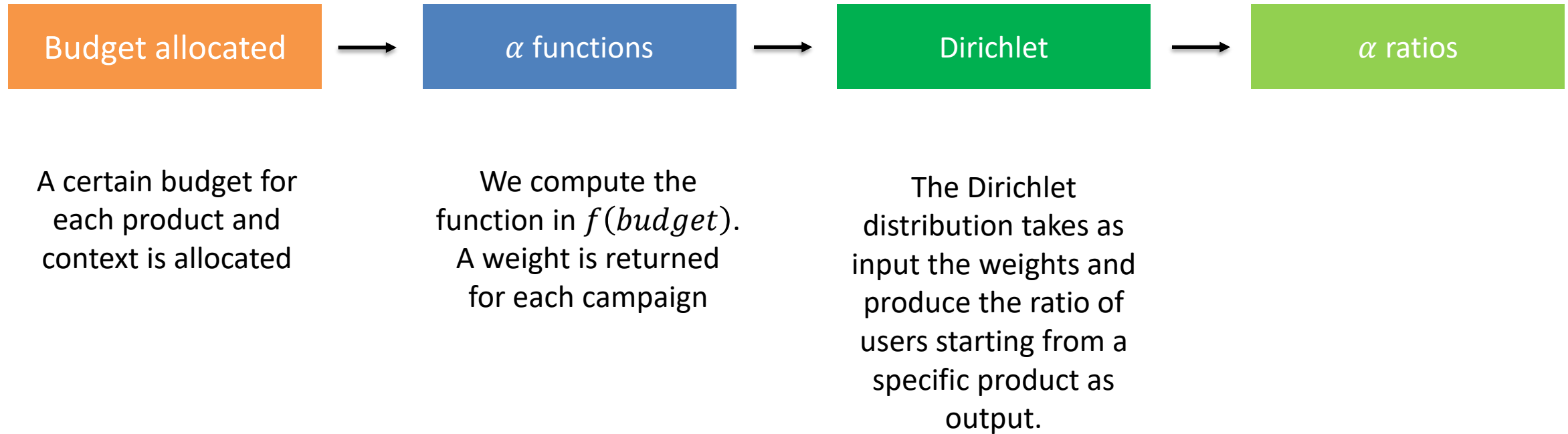
The Environment class

EXAMPLE
of Alpha Function



Step 1: Environment

Alpha functions Explanation



We can notice a relation between the shape of the α functions and the α ratios, through the Dirichlet distribution

Step 1: Environment

The Environment class

round()

This method executed a round of the simulation
corresponding to one day in the real world

It performs these things:

1. Sample the total number of users per class on the website that day
2. Compute the new Alpha Ratios of the new day
 1. Compute how the budget is distributed per class of users proportional to the average percentage of the number of users in that class. (i.e., if the Average per class of user is: [20, 20, 20] the budget will be distributed 1/3 per class)
 2. Compute the increment of the alpha weights using the formula $\gamma_1 \cdot (1 - e^{-\gamma_2 \cdot budget})$ explained in previous slide
 3. Compute the new weights (making sure that the sum is kept constant)
 4. Sample the alphas of this round from the Dirichlet distribution with the weights of step before
3. Adjust the users according to the computed Alpha Ratios
4. Instantiate all the users of the day
5. Simulate the behavior of the users
 1. Sample the reservation price to see if it has bought the product
 2. Check if it has bought the product
 3. Sample if the users clicks on the first primary and/or on the second secondary product
 4. And so on... Until he does not see any more products

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 2: Optimization algorithm

Overview

KNOWN

α functions

Number of items sold

Graph Weights

Binary features

UNKNOWN

UNCERTAIN

Step 2: Optimization algorithm

Formal model

$$\max_{y_{j,t}} \sum_{j=1}^N v_j \cdot n_j(y_{j,t})$$

$$s. t. \sum_{j=1}^N y_{j,t} \leq B_{tot}$$

Considering the formulas above, we have that:

v_j → is the **value** that each user starting from product j produce

$y_{j,t}$ → is the **budget allocated** to the sub-campaign of product j at time t

$n_j(y_{j,t})$ → is the **number of users** starting from the j^{th} product, which is function of the budget allocated for that sub-campaign

B_{tot} → is the **total budget** we can spend in a day.

Step 2: Optimization algorithm

Formal model

$$v_j = \sum_{i=1}^P b_{j \rightarrow i} \cdot p_i \cdot \bar{q}_i$$

Considering the formula above, we have that:

- v_j → is the **value** that each user starting from product j produce
- $b_{j \rightarrow i}$ → is the **probability of buying** product i given that the user started from product j
- p_i → is the **price** of product i
- \bar{q}_i → is the **quantity** of product i sold in average to that user class.

Step 2: Optimization algorithm

Estimator

Since v_j is function of $b_{j \rightarrow i}$, we need to estimate it. We followed two different approaches:

- **Exact Computation** (feasible only in our case, since the number of product is small)
- **Monte Carlo simulation** (less precise, but more scalable)

Step 2: Optimization algorithm

Exact Computation - 1

The **Exact Computation** is based on the following *assumption*:

For each landing page a primary product is shown. When it is bought, two other products (secondaries) are shown. They are bought with a certain probability (in addition to the fact that the second secondary has a λ probability to be seen).

In this way, for each landing product we can build a tree with the landing product as root, and the other products as subsequent nodes. Arcs represent the probability of moving from one node to another and buying the product associated to that node. Once the tree has been built, we visit it and we compute in this way the “buy probability matrix”.

A short example will be presented in the following slide

Step 2: Optimization algorithm

Exact Computation - 2

Consider P1 as landing product. Its first secondary product is (by assumption) P2 whereas its second secondary product is (by assumption) P3. Notice that P2 and P3 will be displayed only once P1 has been bought:

- the probability of buying P2, which is in the first secondary slot, is given by:
 - the probability of buying the primary product P1 multiplied by
 - the “click graph” value [from P1 -> P2] multiplied by
 - **the “buy probability” of buying P2**
- the probability of buying P3 that is in the second secondary slot is given by:
 - the probability of buying the primary product P1 multiplied by
 - λ * the “click graph” value [from P1 -> P3] multiplied by
 - **the “buy probability” of buying P3**

Step 2: Optimization algorithm

Monte Carlo estimation

The second approach we used is **Monte Carlo simulation**.

We perform different Monte Carlo simulation, considering each time a different landing product as seed.

We **simulate the behaviour** of the **user**, which will move around the possible products according to their actions (as a comparison, this can be considered as moving down in the tree of the exact computation according to the user behaviour).

To compute the probabilities we **average** the number of times a product has been **bought** with the total **number of iterations** (users) performed.

Step 2: Optimization algorithm

Monte Carlo estimation

Each Monte Carlo simulation requires a number of iterations sufficient to provide a good estimate.

We computed a lower bound on the number of iterations using the following formula:

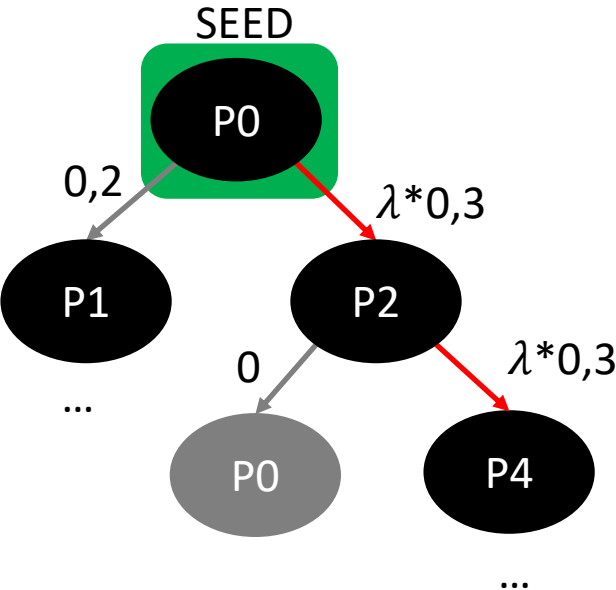
$$R = O\left(\frac{1}{\varepsilon^2} \log(|S|) \log\left(\frac{1}{\delta}\right)\right)$$

That will produce a result with an error of $\pm \varepsilon n$ (where n is the number of nodes), with a probability at least of $1 - \delta$

In order to obtain an acceptable additive error, we need at least around 3000 iterations.

Step 2: Optimization algorithm

Monte Carlo estimation - Example



Example of simulation with P0 as seed:

- 1 The user starts from the landing product P0
- 2 He buys the product P0
- 3 He sees product P1 and P2
- 4 He clicks and buy product P2
- 5 He sees product P4

	P0	P1	P2	P4
Simulations Counter	1	1	1	1
Buy Counter	1	0	1	0
Probability estimated (Buy/Sim)	1	0	1	0

Step 2: Optimization algorithm

Optimizer

The optimizer role is to solve the optimization problem described in the formal model. It has to **maximize the profit** allocating different budgets to each sub-campaign taking into account the total budget constraint.

To do this we adopted the **dynamic programming technique** seen during the course, which is a variation of the knapsack problem.

The method implemented in the code simply returns **the optimal allocation of the budget**, with the expected profit for it.

Once every v_j has been computed for every product, the value $n_j(y_{j,t})$ can be simply computed by exploiting the known relation between the budget allocated and how the Dirichlet's weights will increase.

Step 2: Optimization algorithm

Optimizer

We will use the result computed by the **optimizer, which knows everything**, as the result of a **clairvoyant** algorithm.

We will compare, in the following steps, the optimal result and the values produced using the MABs which will have to estimate some uncertain parameters.

The **Regret**, at each round, will be computed with the following formula:

$$R_t = \mu^* - r_t$$

Where μ^* is the optimal reward (in average), while r_t is the reward we have at round t during the simulation.

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 3: Optimization with uncertain α functions

Overview

KNOWN

UNKNOWN

UNCERTAIN

α functions

Number of items sold

Graph Weights

Binary features

Step 3: Optimization with uncertain α functions

Approach to the Problem

We chose to

estimate the α ratio runtime using a MAB.

So, the MAB will learn the functions that relate the Budget Spent and the Alpha Ratio.

In this way we will have:

5 MABs:

5 campaign x 1 aggregate class

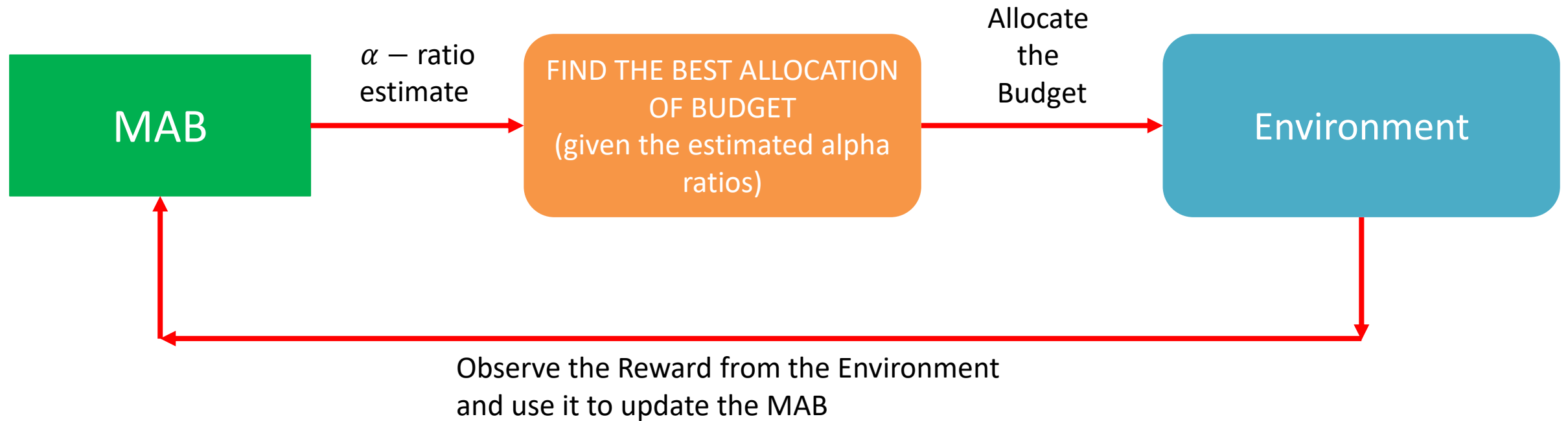
the users behave differently according to their class, which is not observable

The optimizer will allocate the budget.

The observations will be used to update the MAB, which will be used to retrieve an estimation of the alpha ratios.

Step 3: Optimization with uncertain α functions

Approach to the Problem



Step 3: Optimization with uncertain α functions

REWARDS

REWARDS

Each user will start from a primary product according to the budget allocated, in general, **more budget** for a product is spent, **more users will start from it**.

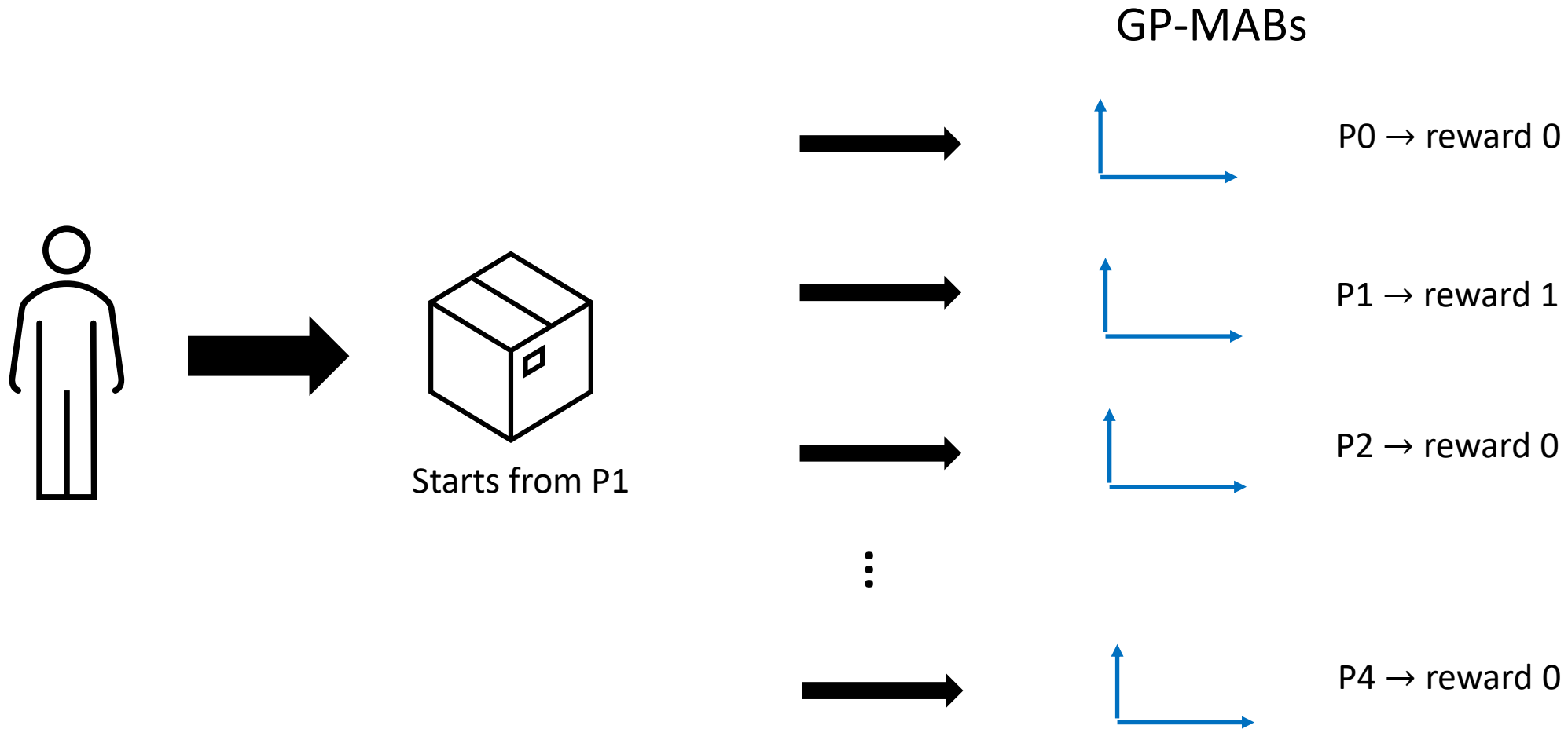
Each user will produce a reward that we will use to update our **5 MABs**, in correspondence with the used arm.

The MAB corresponding to the primary product from which the user started will receive a reward of 1.

The others MAB will receive a reward of 0.

Step 3: Optimization with uncertain α functions

REWARDS – Example binary rewards



Step 3: Optimization with uncertain α functions

REWARDS - Mean Value

REWARDS (Mean Value)

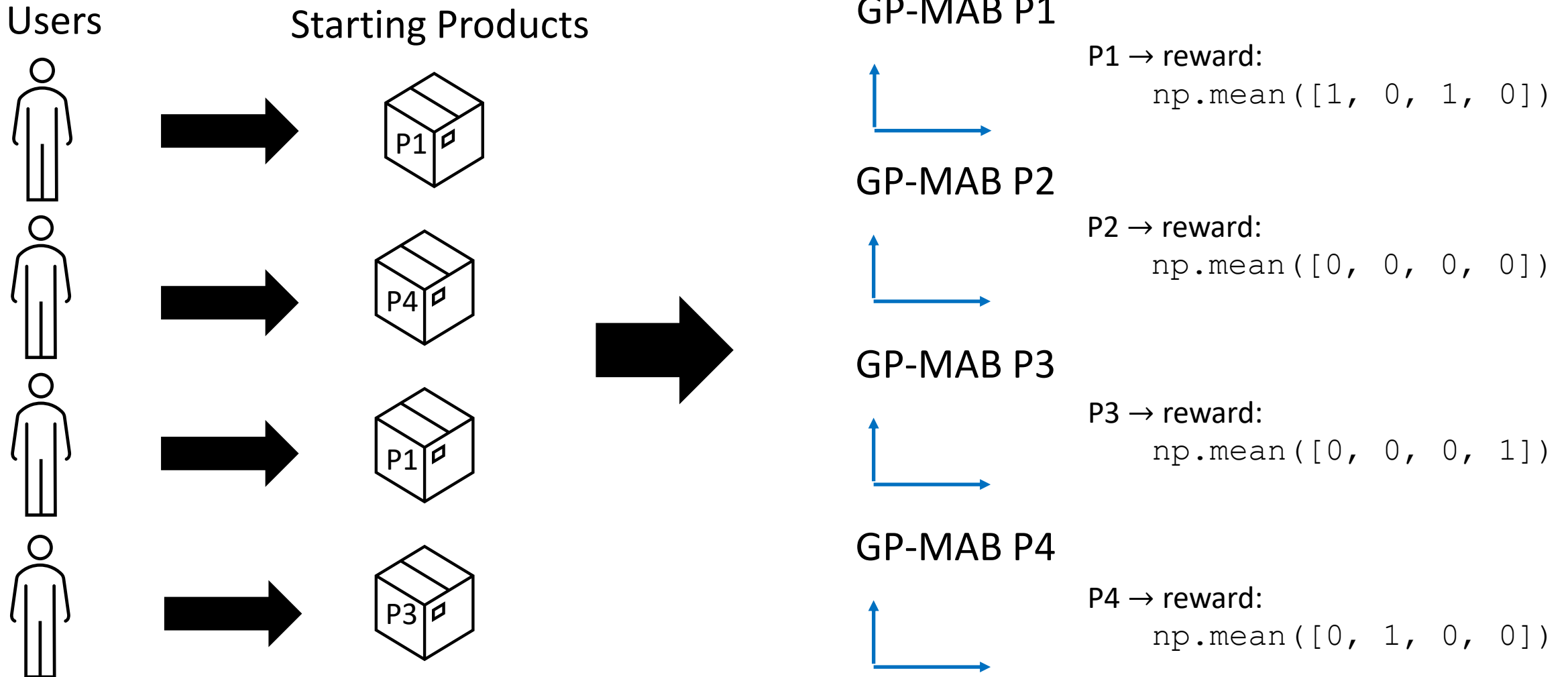
We also tried a second version of rewards, in which, instead of updating the MABs for each sample we update it once per round with the percentage of the users starting from that product in that round (e.g., if 3 users started from product 1, and 20 from other products or competitor, it is updated with $\frac{3}{23}$)

This may lead to loss of some information (we neglect the number of user of a specific round). The performances resulted to be similar to the first version, hence we followed the first version.

The only advantage is the computational time that resulted to be faster, and so could be an approach to follow to be real-time even if as mentioned before we lose some information.

Step 3: Optimization with uncertain α functions

REWARDS – Mean Value example



Step 3: Optimization with uncertain α functions

The MAB

Basically we need to estimate the α ratios using a MAB

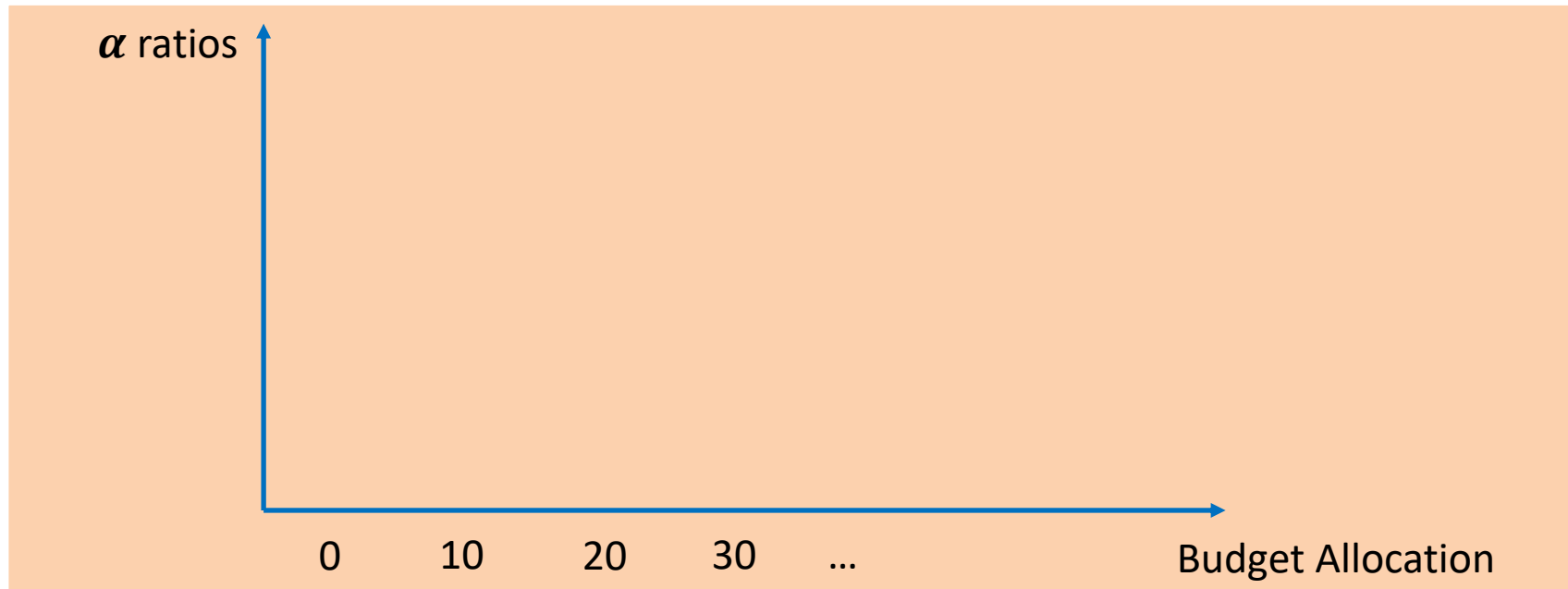
We have developed it with both

UCB

and

Thompson Sampling

Probability of a user to start from a certain product given a certain budget



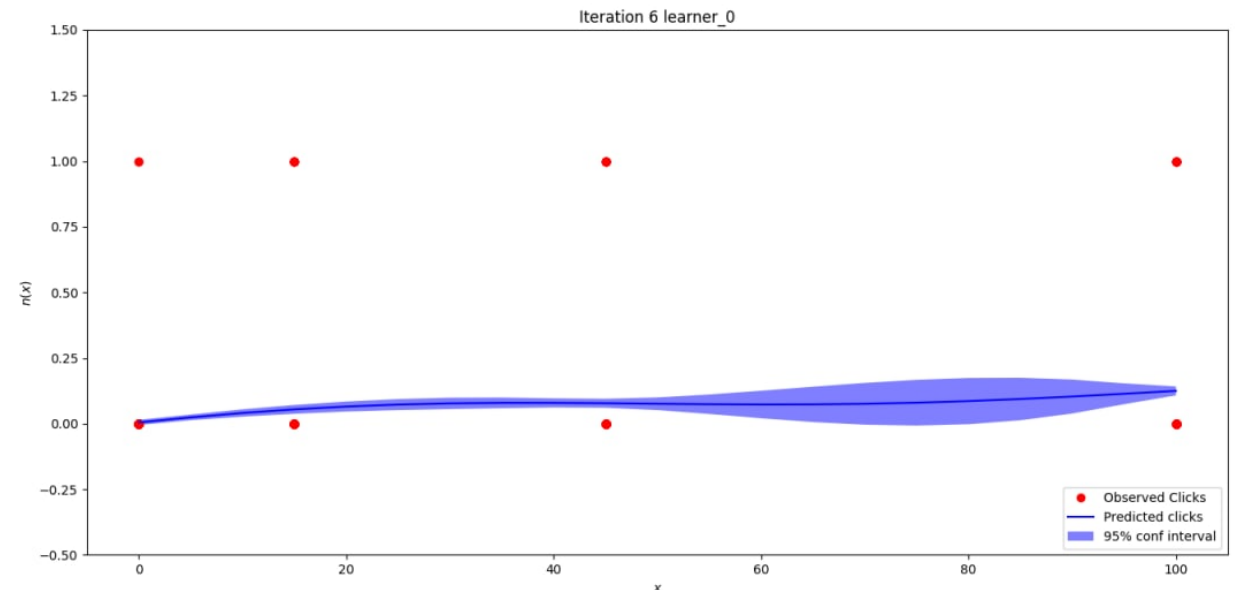
Step 3: Optimization with uncertain α functions

Gaussian Process

Since we have a lot of arms, we exploited **Gaussian Process** to estimate values of never pulled arms but near to the ones that have been pulled.

We use a Gaussian Process with a kernel that is a combination of a **Costant** and a **Radial Basis Function**, where we have fine tuned the parameters, to get acceptable results.

$$K(X_1, X_2) = e^{-\frac{\|X_1 - X_2\|^2}{2\sigma^2}}$$



Step 3: Optimization with uncertain α functions

MAB + GP - Sampling Process

We can use two different implementations of MABs:

UCB and **TS**, adapted to include the **Gaussian Processes**.

The Sampling process:

GP UCB

1. Retrieve Mean and Variance for each arm exploiting the Gaussian Process
2. Get a new sample generated with the formula in the following slide

GP TS

1. Retrieve Mean and Variance for each arm exploiting the Gaussian Process
2. Sample from a Normal with mean and variance of step 1

Step 3: Optimization with uncertain α functions

GP-UCB

GP-UCB Sampling Process:

Return an **Upper Confidence Bound** as:

$$\mu_{t-1}(x) + \sqrt{\beta_t} \cdot \sigma_{t-1}(x)$$

with:

$$\beta_t = 2 \cdot \log \left(\frac{|D| \cdot t^2 \cdot \pi^2}{6 \cdot \delta} \right)$$

Cumulative Regret Bound:

$$P \left(R_T < O \left(\sqrt{T \cdot \gamma_t \cdot \log(|D|)} \right) \forall T \geq 1 \right) < 1 - \delta$$

where:

γ_t is the Maximum Information Gain after t rounds

<https://arxiv.org/pdf/0912.3995.pdf>

Step 3: Optimization with uncertain α functions

GP-TS

GP-TS Sampling Process:

Return a **Sample From**:

$$x \sim N(\mu, \sigma^2)$$

with:

$\mu, \sigma \rightarrow$ from the Gaussian Process

Cumulative Regret Bound:

$$P \left(R_T < \sqrt{\frac{2 \cdot \Lambda^2}{\log \left(1 + \frac{1}{\sigma^2} \right)} \cdot C \cdot T \cdot B \cdot \sum_{k=1}^C \gamma_{k,T}} , \forall T \geq 1 \right) < 1 - \delta$$

where:

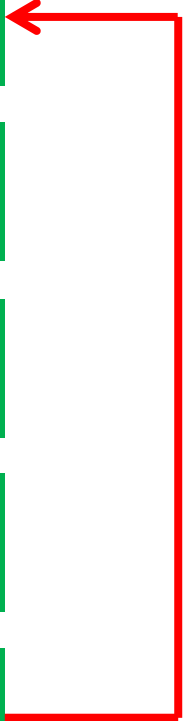
γ_t is the Maximum Information Gain after t rounds

$$B = 8 \cdot \log(2T^2 MC / \delta)$$

from course slides

Step 3: Optimization with uncertain α functions

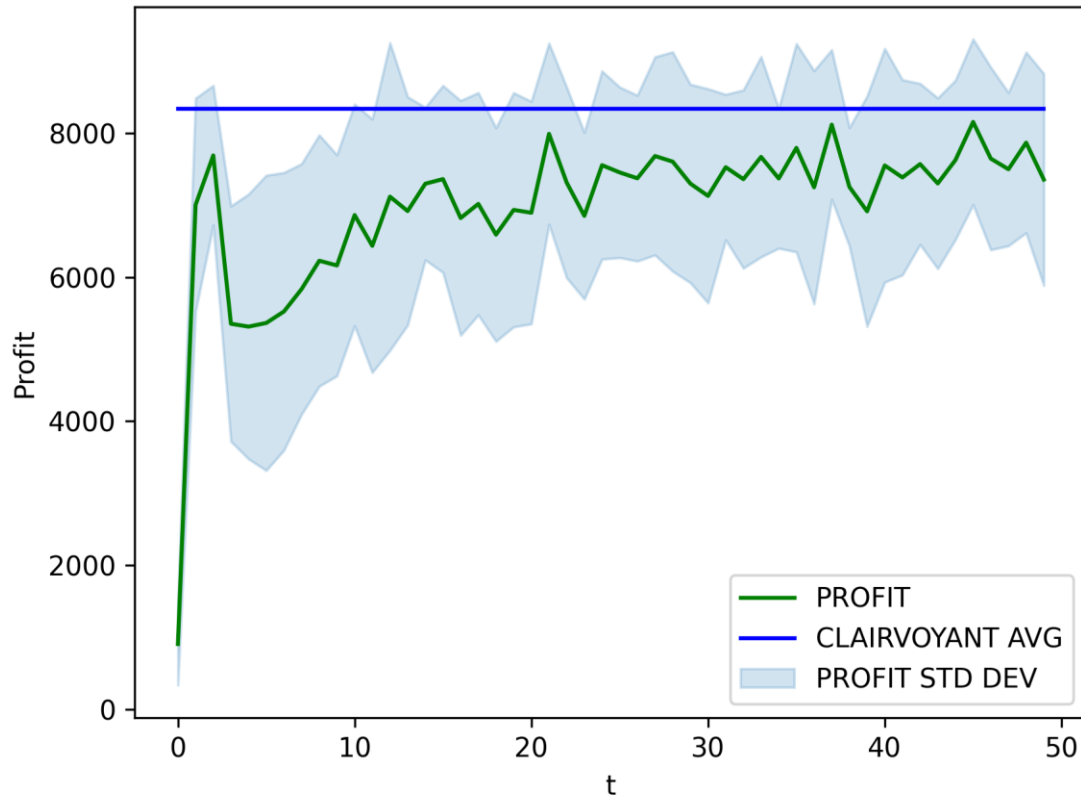
Steps of the Algorithm

- ① Choose the ARMS (possible budget values) and initialize the MABs
 - ② For each arm consider the estimate of the α ratio
(upper confidence bound for UCB or a sample from a normal in case of TS)
 - ③ Give all the estimated α ratios to the Optimizer. It will choose the best budget to allocate
 - ④ Allocate the returned budget and observe at the end of the day the reward from the Environment
 - ⑤ Update the MABs with the Reward observed according to the real budget allocated
 - ⑥ Restart from Step 2
- 

Step 3: Optimization with uncertain α functions

Results

TASK 3 UCB
REWARD PLOT and CLAIRVOYANT PLOT



UCB

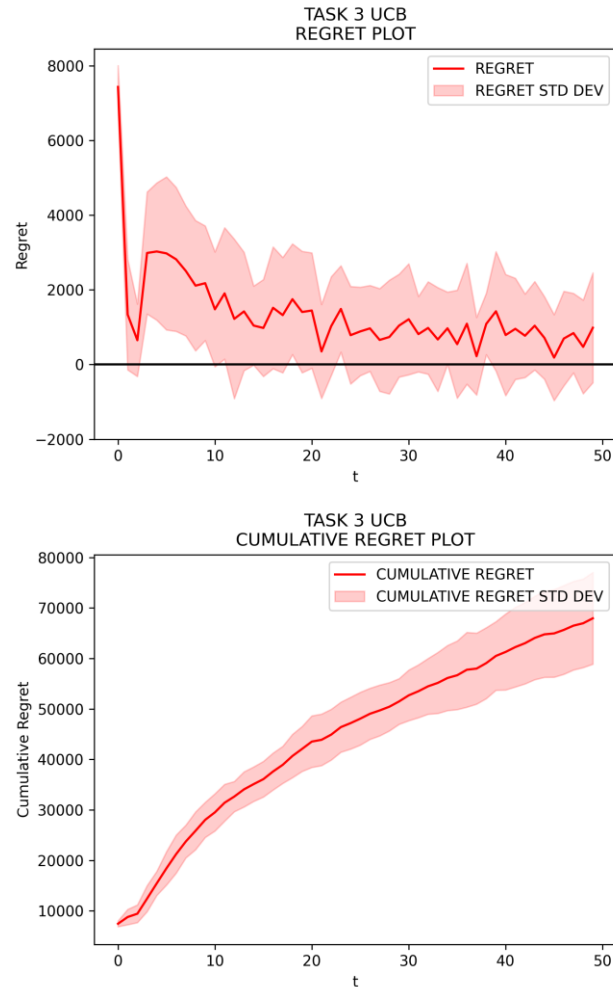
We can notice that, after an initial *learning phase* where we **explore a lot** (the big drop at the beginning is due to exploration) then we tend to the Clairvoyant.

Note:

20 experiments performed over 150 users
so, nearly 3000 samples collected for the experiment

Step 3: Optimization with uncertain α functions

Results



UCB

We can notice that, after an initial *learning phase* where we **explore a lot** (the big drop at the beginning is due to exploration) then we tend to the Clairvoyant.

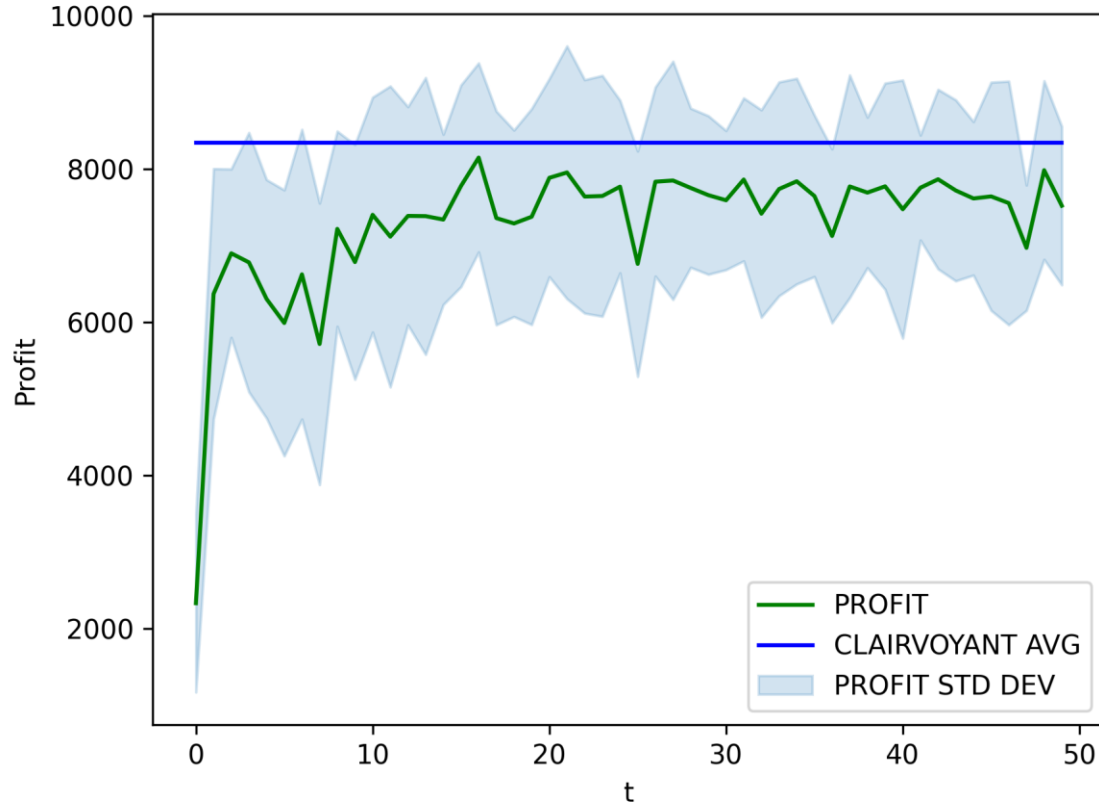
Note:

20 experiments performed over 150 users
so, nearly 3000 samples collected for the experiment

Step 3: Optimization with uncertain α functions

Results

TASK 3 TS
REWARD PLOT and CLAIRVOYANT PLOT

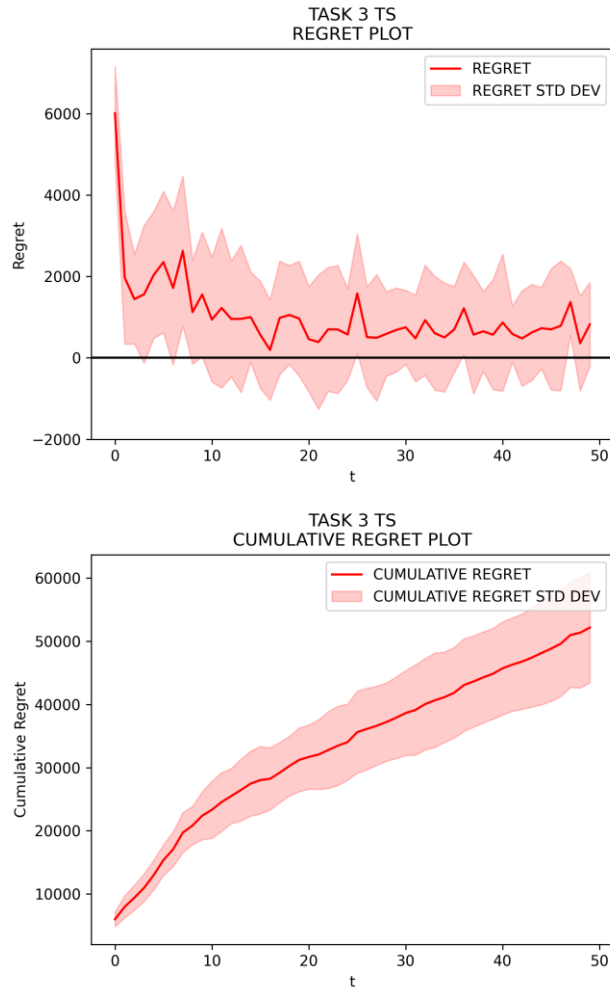


Thompson Sampling

As expected, we can notice that the **Thompson Sampling** variant is performing slightly better than the UCB because of its stochasticity, that is a very good feature!

Step 3: Optimization with uncertain α functions

Results



Thompson Sampling

As expected, we can notice that the **Thompson Sampling** variant is performing slightly better than the UCB because of its stochasticity, that is a very good feature!

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

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

Overview

KNOWN

UNKNOWN

UNCERTAIN

α functions

Number of items sold

Graph Weights

Binary features

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

Approach to the Problem

The **uncertainty in α ratios**

is approached exactly as before, with a **GP-MAB**.

The **uncertainty in the number of items sold**

is solved by performing an **AVERAGE** over all the observations.

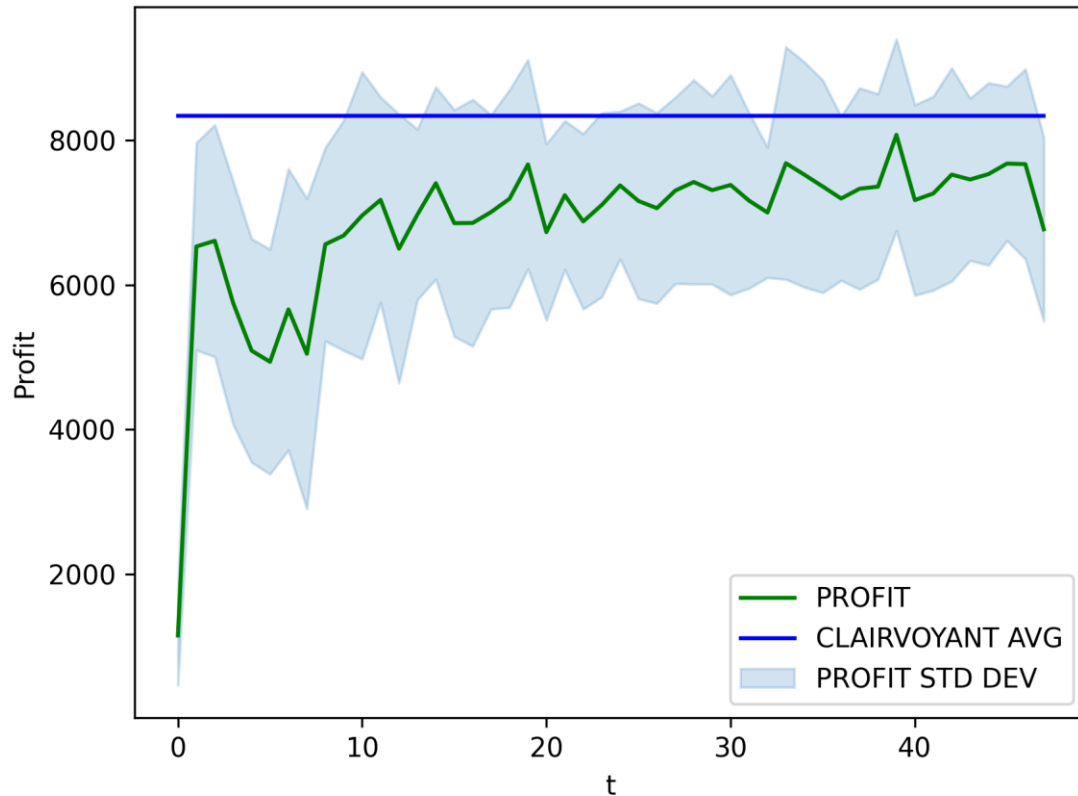
NOTE:

a MAB is not good to estimate the Number of Items Sold since:
the Observation: “number of items sold for a specific product”
does not depend on our Action: “Budget Allocated for each Campaign”

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

Results

TASK 4 UCB
REWARD PLOT and CLAIRVOYANT PLOT



UCB

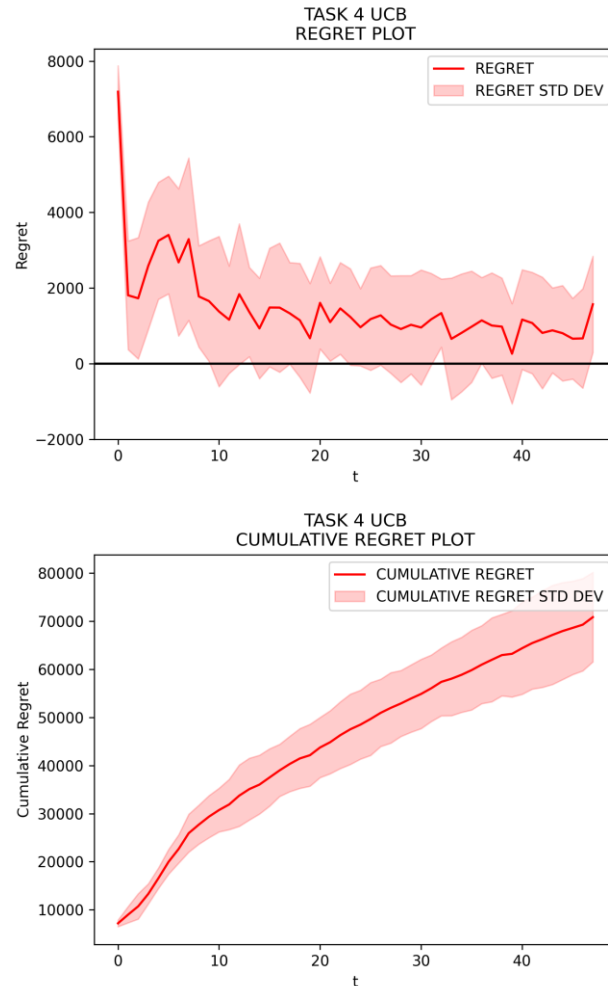
The UCB algorithm has similar performances to step 3, the uncertainty in the number of item sold seems to not influence significantly the performances.

Note:

20 experiments performed over 150 users
so, nearly 3000 samples collected for the experiment

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

Results



UCB

The UCB algorithm has similar performances to step 3, the uncertainty in the number of item sold seems to not influence significantly the performances.

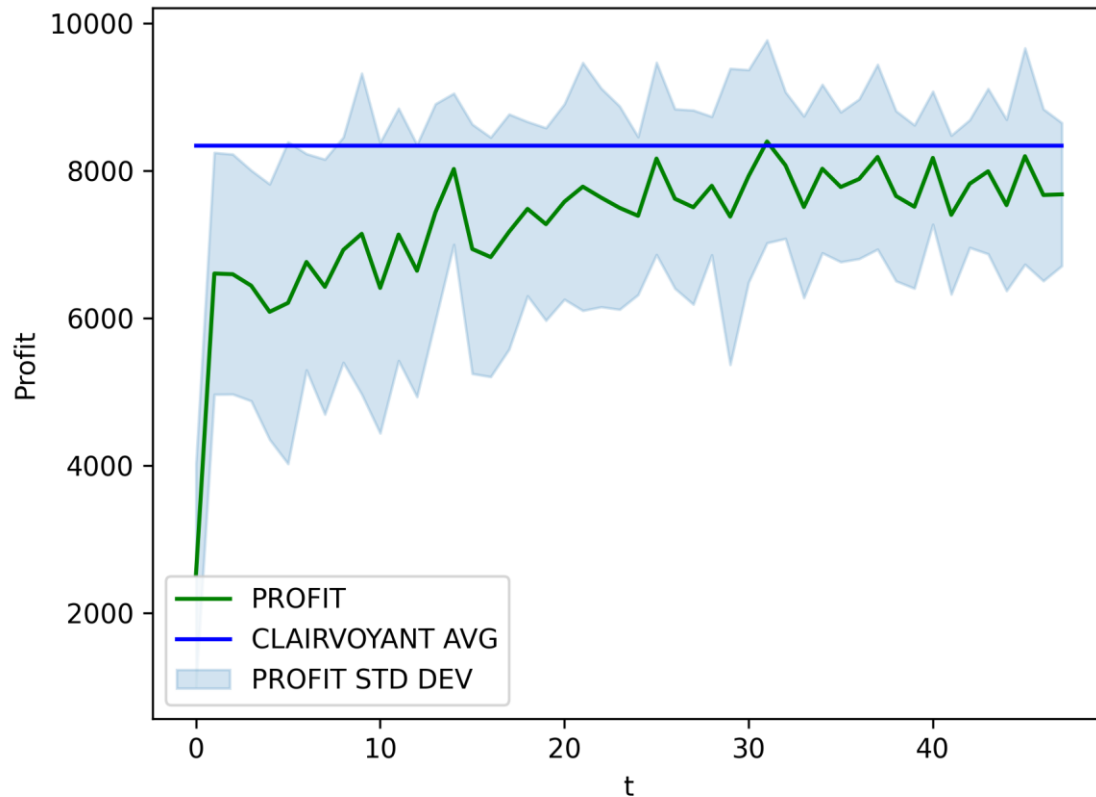
Note:

20 experiments performed over 150 users
so, nearly 3000 samples collected for the experiment

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

Results

TASK 4 TS
REWARD PLOT and CLAIRVOYANT PLOT

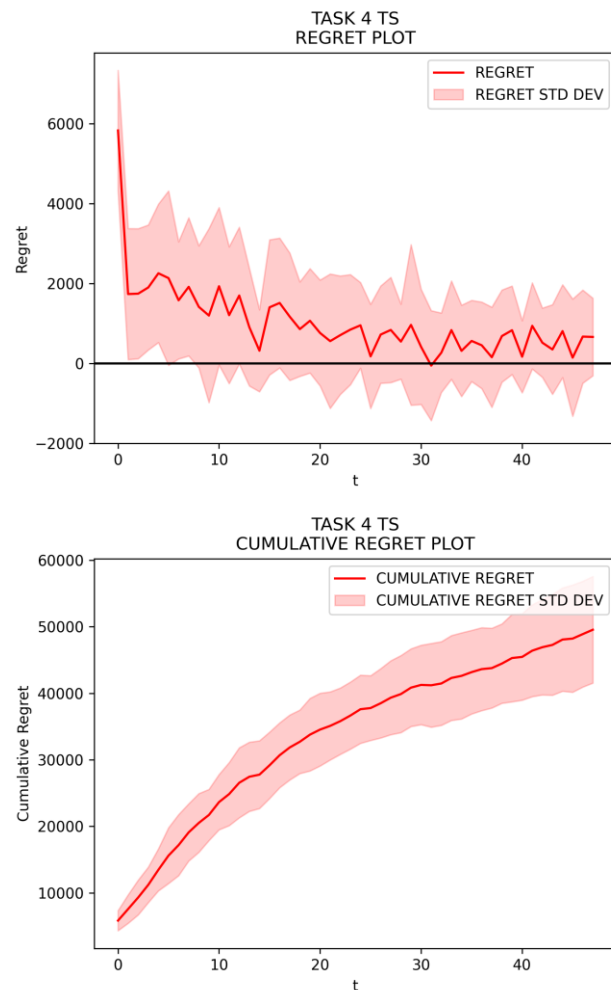


TS

As expected, we can notice that the **Thompson Sampling** variant is performing slightly better than the UCB because of its stochasticity also with unknown quantities.

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

Results



TS

As expected, we can notice that the **Thompson Sampling** variant is performing slightly better than the UCB because of its stochasticity also with unknown quantities.

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 5: Optimization with uncertain Graph Weights

Overview

KNOWN

α functions

Number of items sold

UNKNOWN

Binary features

UNCERTAIN

Graph Weights

Step 5: Optimization with uncertain Graph Weights

Approach

We can split the problem in two components:

- We need to **estimate the transition probability from one product to another**, since in this case it is not known
- We need to maximize our profit **acting on the budget allocation**: this problem can be solved exploiting influence maximization techniques.

Step 5: Optimization with uncertain Graph Weights

Approach – Transition Probabilities

The first component consists of the transition probabilities. Since we cannot control the users' transition probabilities, we need to resort to the **AVERAGE**, as we did in step 4.

We collect the samples during each round: every time a user sees a product, we record if he clicked on it or not.

Then, we can simply estimate the transition probabilities averaging the number of clicks over the number of times a product has been seen.

Remember:

a MAB is not a good choice to estimate the transition probabilities:
the Observation: “The transition between two products”
does not depend on our Action: “Budget Allocated for each Campaign”

Step 5: Optimization with uncertain Graph Weights

Approach – Influence Maximization

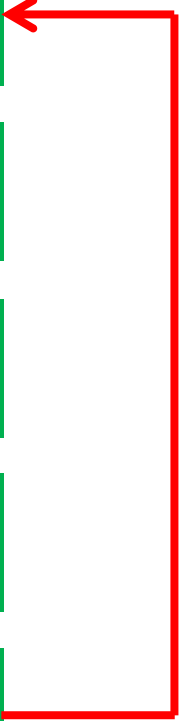
Exploiting **influence maximization**, we can act on the **budget allocation** to maximize our profit.

We can act similarly to what we did in step 2: we compute v_j running different **Monte Carlo** simulations (one for each product set as *seed*). The main difference with respect to step 2 is that here the transition probability is unknown, and we estimate it as described in the slide above.

An overview of the complete procedure is visible in the following slide.

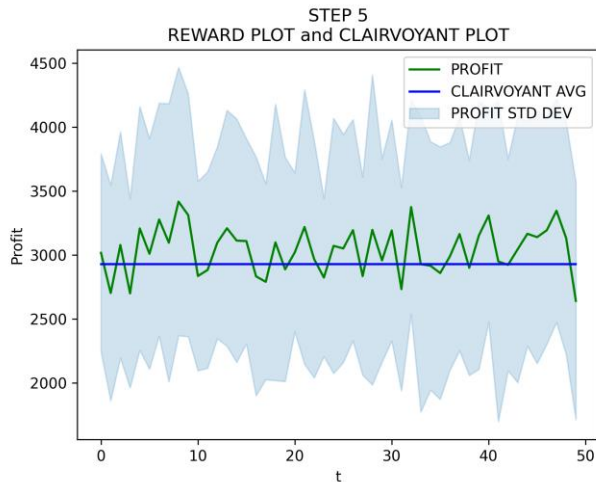
Step 5: Optimization with uncertain Graph Weights

Steps of the Algorithm

- ① Consider the “transition probability” matrix estimated with the average
 - ② Run Monte Carlo simulations considering each product as landing product (*seed*), using the estimated “transition probability”.
 - ③ Run the optimizer, which will return the best allocation with its expected profit exploiting the results of the Monte Carlo simulations
 - ④ Allocate the returned budget and observe at the end of the day the reward from the Environment
 - ⑤ Update the transition probability using the average
 - ⑥ Go to step 2 and start another round
- 

Step 5: Optimization with uncertain Graph Weights

Results



As we can notice we directly end up to the nearly optimal values since we don't have a «Learning Phase», but we estimate the values using an average. This is possible because, as written in the text, the **only** uncertain parameters are the Graph Weights.

We are very satisfied by this result, and we can say that probably the convergence of the transition probability estimation is much faster than the one of the alpha ratios estimation.

In this case we don't need to choose the arm to be pulled, but whatever budget we allocate, we can observe and use all the returned data to update our unique estimation.

Step 5: Optimization with uncertain Graph Weights

VARIANT: Uncertain Weights + Alpha Functions

KNOWN

UNKNOWN

UNCERTAIN

α functions

Number of items sold

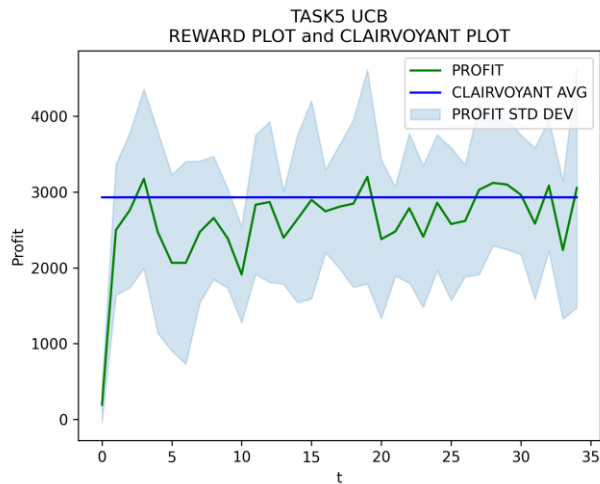
Graph Weights

Binary features

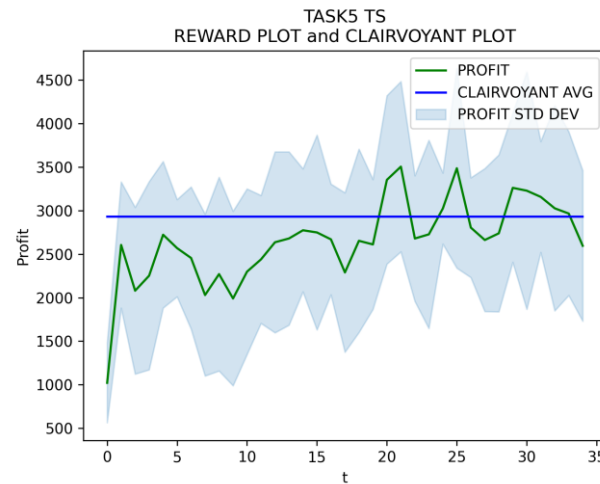
Step 5: Optimization with uncertain Graph Weights

Results

UCB



TS



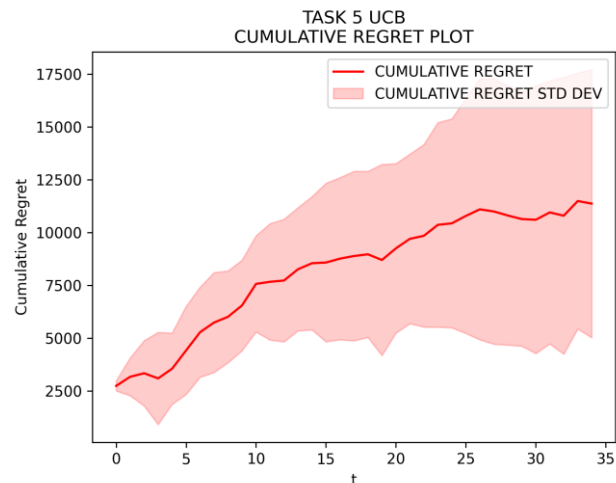
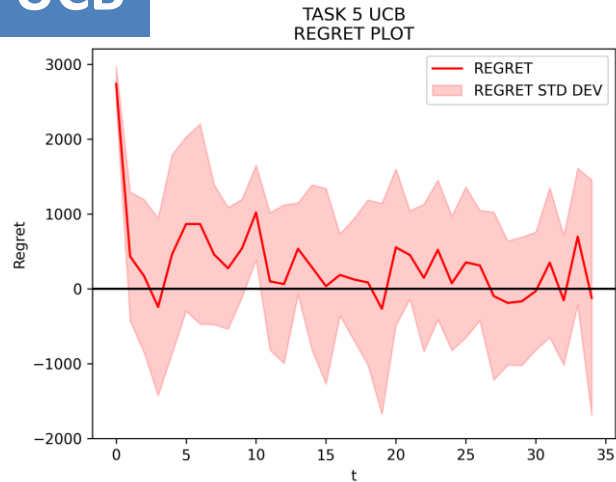
If instead we assume uncertain alpha functions, we also need to estimate them. To do that, we can use GP-UCB or GP-TS approach, as done in the previous steps.

As we can see from the graphs on the left, the uncertainty in the alpha function plays an important role in the initial regret value, making it higher.

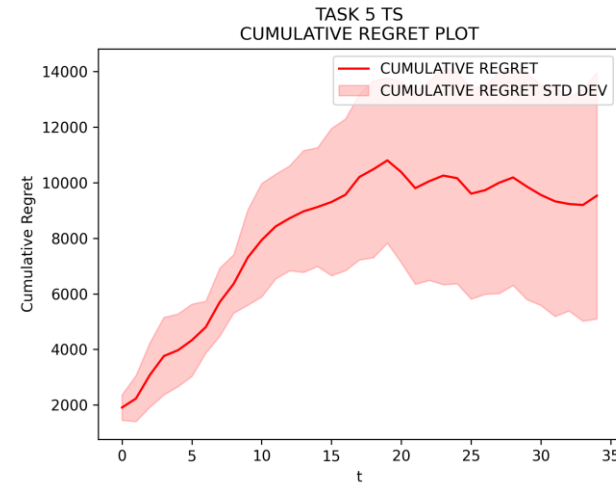
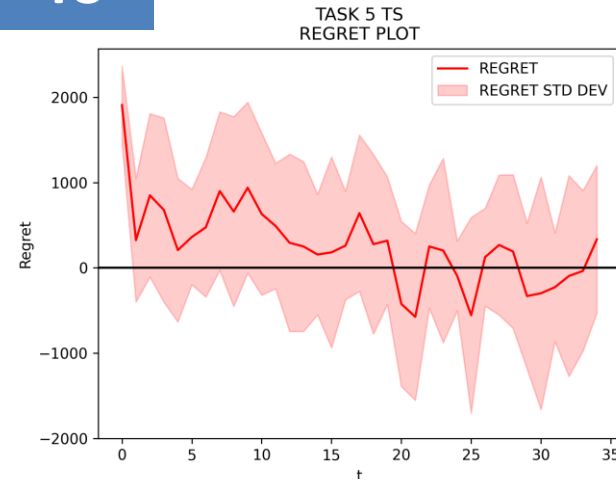
Step 5: Optimization with uncertain Graph Weights

Results

UCB



TS



If instead we assume uncertain alpha functions, we also need to estimate them. To do that, we can use GP-UCB or GP-TS approach, as done in the previous steps.

As we can see from the graphs on the left, the uncertainty in the alpha function plays an important role in the initial regret value, making it higher.

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 6: Non-stationary demand curve

Overview

KNOWN

UNKNOWN

UNCERTAIN

α functions with abrupt changes

Number of items sold

Graph Weights

Binary features

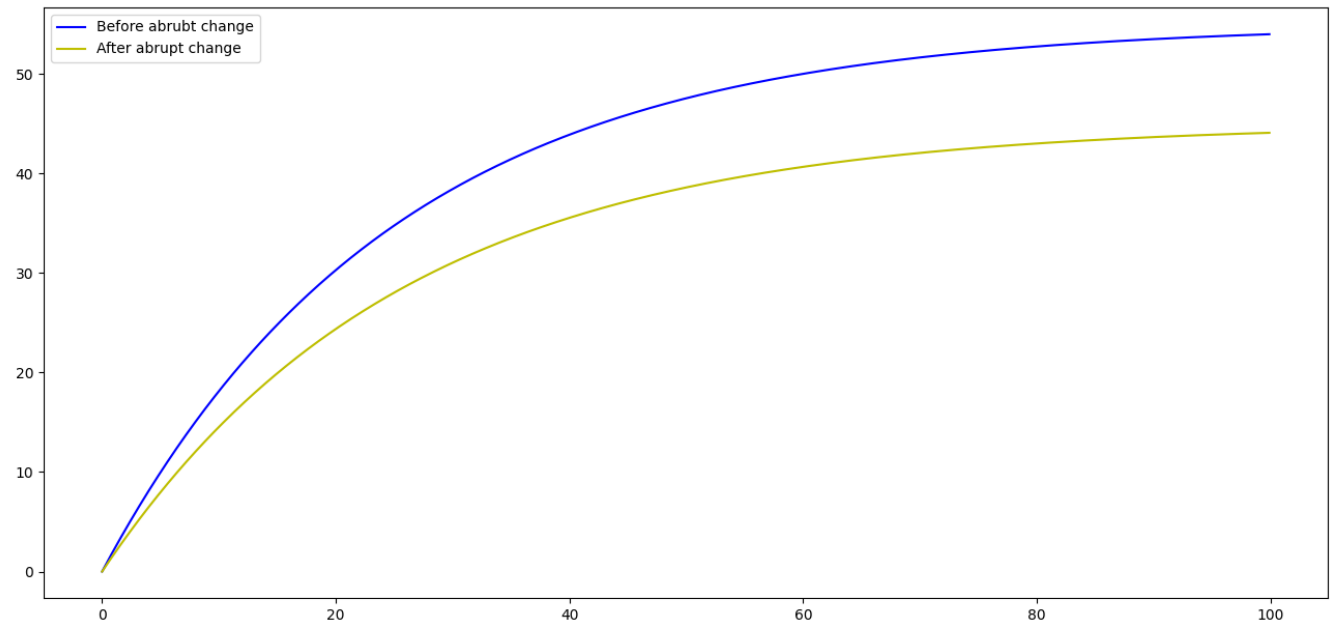
Step 6: Non-stationary demand curve

Note

The request here is to face the situation in which the **demand curve changes** during time. In other words our algorithm needs to cope with an *abrupt concept drift*.

Since we are facing an advertising problem (and we cannot act on the price) we interpret the text as if what is really changing are the **alpha ratios**, and so how much effective is to advertise a specific product.

Here an example, where we show how dirichlet's weights change, for the same budgets, before and after an abrupt change



Step 6: Non-stationary demand curve

Motivating Application

By hypothesis, from time to time, α functions may have **abrupt changes**. This means that they are non-stationary.

In our clothes example, this may happen for several reasons:

- Seasonal changes due to the weather
- New competitor enters the market
- A new trend become popular in few days

Our objective is to face this changes quickly in order to avoid a loss in our incomes.

Step 6: Non-stationary demand curve

Approach

Considering step 3, the only difference is that we need to implement some solutions to discard too old samples, since they may be no more significative, because in the meanwhile an abrupt change may have occurred.

In order to face this problem, we followed two different strategies:

1. **Sliding window** approach
2. **Change detection** technique

The deletion of the samples is the **only thing** we need to add to step 3 in order to tackle the problem. For this reason, all the other parts remain untouched.

Step 6: Non-stationary demand curve

Sliding Window Approach

We set a sliding window of length t , whose target is to consider samples drawn, **at most**, in the previous t rounds. Since the window size is fixed, and leads to a drop of samples, we expect to decrease a bit our performances.

The size, in our case, has been chosen empirically, in particular we set the size τ of the window with the following formula, where T is the time horizon of our experiment:

$$\tau = \sqrt{T}$$

We expect this algorithm to perform worse w.r.t. change detection since at a certain time, the window overlaps two different phases (it will contain some samples drawn from the old distribution).

Step 6: Non-stationary demand curve

Change Detection Technique

With change detection technique we have a fundamental difference: in this case we don't have a fixed-size window, but we have an algorithm which decides whether to discard all the old samples or not.

If the algorithm finds that the new samples come from a different distribution, all the previous samples for the specific arm will be discarded.

We implemented the formulas of **CUSUM algorithm** visible in the following slide.

Step 6: Non-stationary demand curve

Change Detection Technique – theoretical background

The first k samples of an arm are collected in order to have a reliable estimate of \bar{X}_a^0

Now we can start to detect possible changes. We can define the **positive** ($s_a^+(t)$) and **negative** ($s_a^-(t)$) **deviation from the reference point** at time t

$$\begin{aligned}s_a^+(t) &= (x_a(t) - \bar{X}_a^0(t)) - \epsilon \\ s_a^-(t) &= -(x_a(t) - \bar{X}_a^0(t)) - \epsilon\end{aligned}$$

We can update the **cumulative positive** ($g_a^+(t)$) and **cumulative negative** ($g_a^-(t)$) **deviation from the reference point** at time t

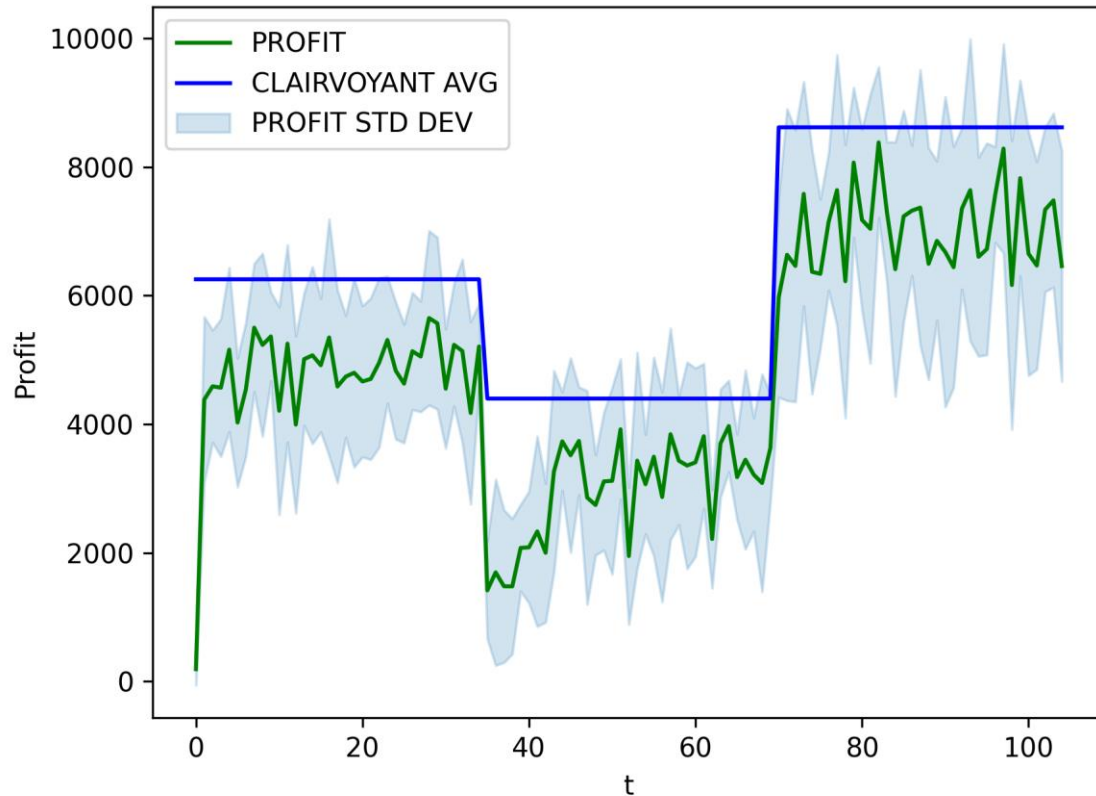
$$\begin{aligned}g_a^+(t) &= \max\{0, g_a^+(t-1) + s_a^+(t)\} \\ g_a^-(t) &= \max\{0, g_a^-(t-1) + s_a^-(t)\}\end{aligned}$$

These two parameters track **how much** samples deviate from the average. When this quantity exceed a fixed threshold, a change is detected

Step 6: Non-stationary demand curve

Results

TASK 6 Sliding Window
REWARD PLOT and CLAIRVOYANT PLOT



Sliding window approach

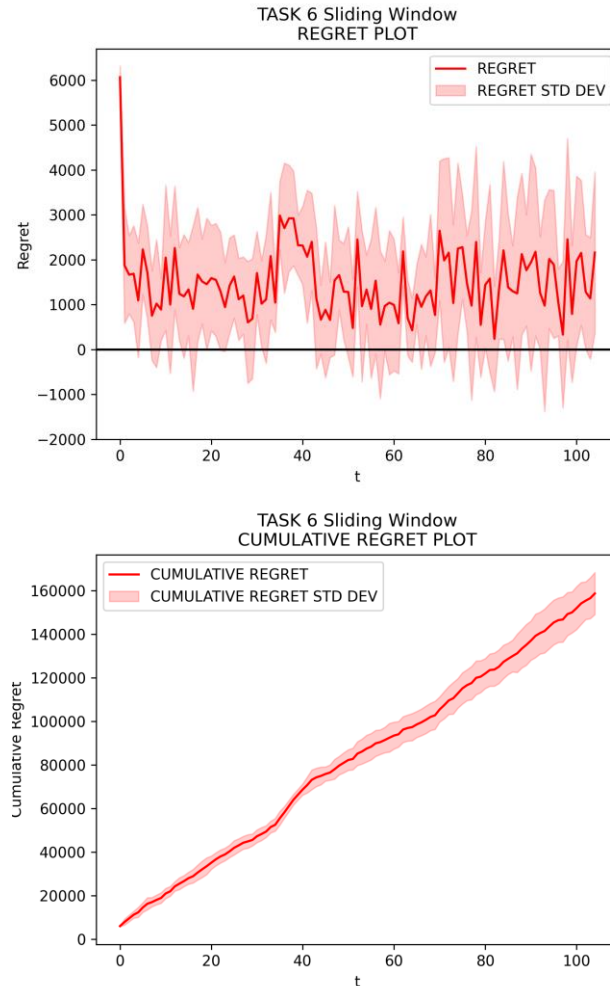
Here we can see the result of a simulation over 105 days, where the alpha functions change three times, modifying the optimal reward we can achieve.

The simulation is done with a sliding window of length 10.

We can see how the responses to changes in environment are quite rapid, however a small sliding window does not allow to learn much from the past samples, that are discarded quickly.

Step 6: Non-stationary demand curve

Results



Sliding window approach

Here we can see the result of a simulation over 105 days, where the alpha functions change three times, modifying the optimal reward we can achieve.

The simulation is done with a sliding window of length 10.

We can see how the responses to changes in environment are quite rapid, however a small sliding window does not allow to learn much from the past samples, that are discarded quickly.

Looking at the cumulative regret we can see clearly that the sliding window impact in a negative way on the performances, since it becomes difficult to obtain a sublinear regret with few samples.

Step 6: Non-stationary demand curve

Results

TASK 6 Change Detection
REWARD PLOT and CLAIRVOYANT PLOT



Change detection approach

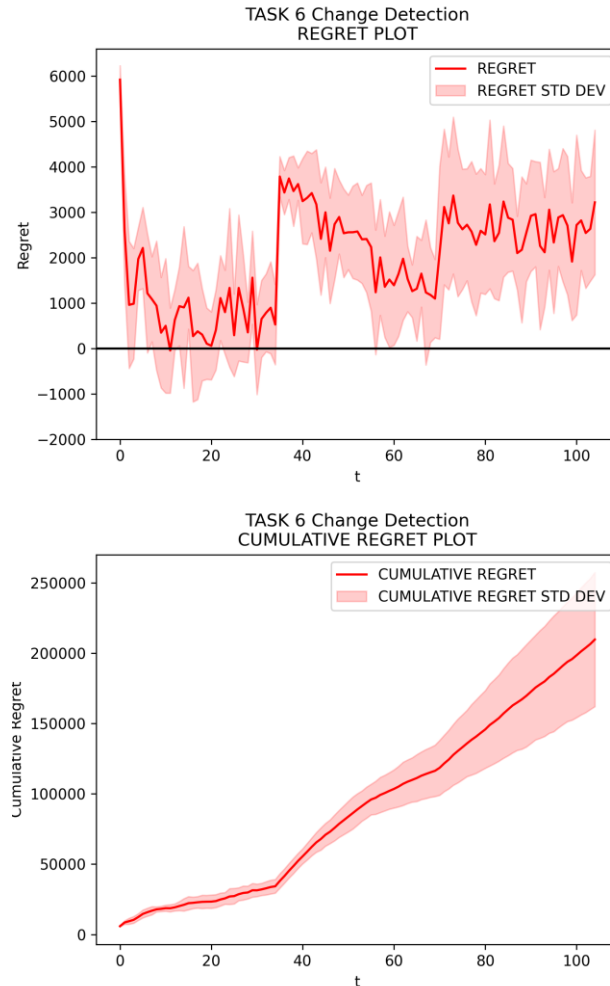
Here we can see the result of a simulation over 105 days, where the alpha functions change three times, modifying the optimal reward we can achieve.

The simulation is done with the change detection algorithm described previously.

We can notice that the change in the alpha function cause a greater regret, and some steps are required before we can learn the correct model after an abrupt change. This, probably, is due to the fact that when a change is detected in correspondence of one arm, only the rewards of the corresponding arm are discarded. For this reason, we implemented a different version of the algorithm.

Step 6: Non-stationary demand curve

Results



Change detection approach

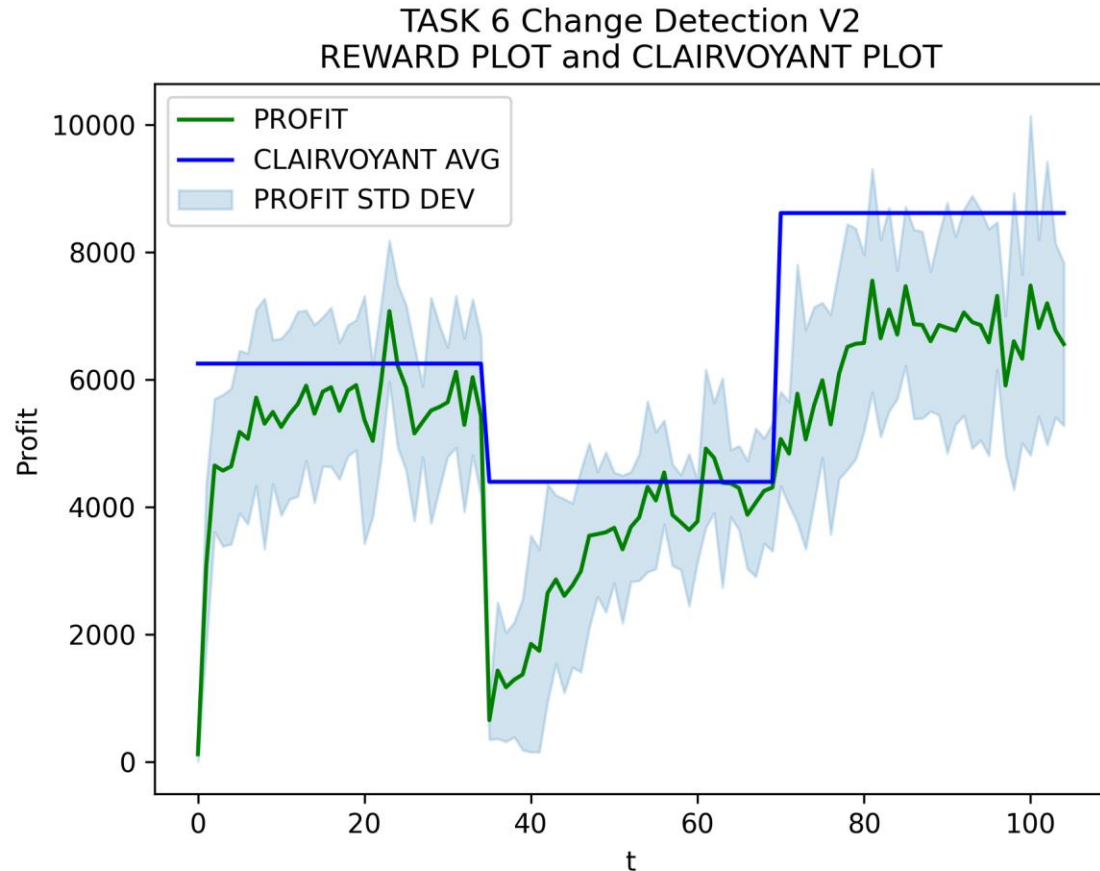
Here we can see the result of a simulation over 105 days, where the alpha functions change three times, modifying the optimal reward we can achieve.

The simulation is done with the change detection algorithm described previously.

We can notice that the change in the alpha function cause a greater regret, and some steps are required before we can learn the correct model after an abrupt change. This, probably, is due to the fact that when a change is detected in correspondence of one arm, only the rewards of the corresponding arm are discarded. For this reason, we implemented a different version of the algorithm.

Step 6: Non-stationary demand curve

Results



Change detection approach – version 2

We did a different attempt on the change detection mechanism, here, when a change is detected for an arm, all the samples of the corresponding GP-MAB are discarded.

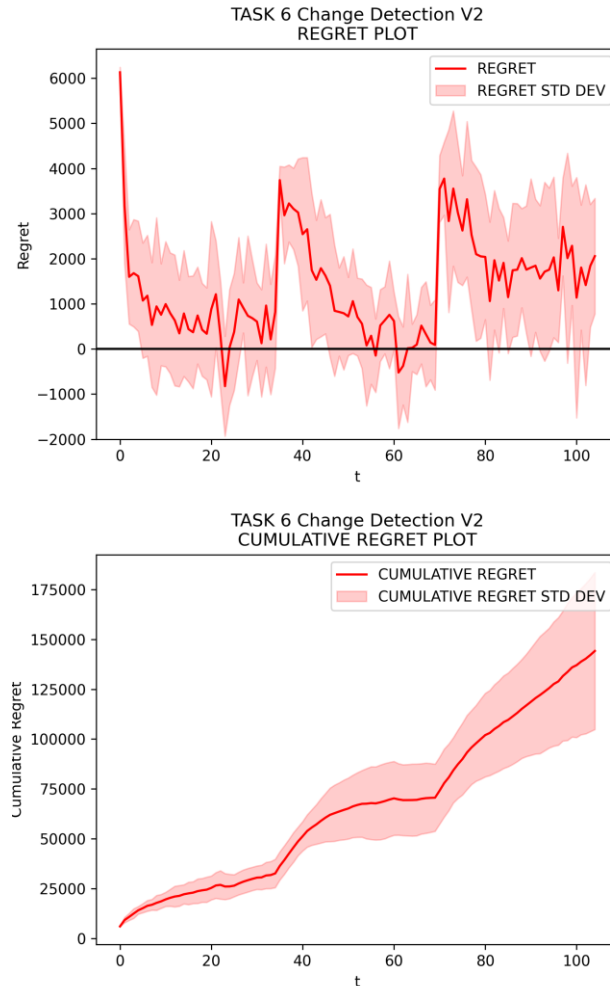
This mechanism can be better in our case since the alpha functions changes in all the points at the same time, but it could be a drawback if the function changes only for specific arms, since we would discard samples that would be instead still valid.

We can notice that, in particular for the first change, the model reacts faster than the previous one.

In general this last version performs better, as visible looking at the graphs, w.r.t. the previous two approaches.

Step 6: Non-stationary demand curve

Results



Change detection approach – version 2

We did a different attempt on the change detection mechanism, here, when a change is detected for an arm, all the samples of the corresponding GP-MAB are discarded.

This mechanism can be better in our case since the alpha functions changes in all the points at the same time, but it could be a drawback if the function changes only for specific arms, since we would discard samples that would be instead still valid.

We can notice that, in particular for the first change, the model reacts faster than the previous one.

In general this last version performs better, as visible looking at the graphs, w.r.t. the previous two approaches.

Roadmap

1

Environment

3

Optimization Algorithm with uncertain α functions

5

Optimization Algorithm with uncertain graph weights

7

Context generation

2

Optimization Algorithm

4

Optimization Algorithm with uncertain α functions and number of items sold

6

Optimization Algorithm with non-stationary α functions

Step 7: Context generation

Overview

KNOWN

UNKNOWN

UNCERTAIN

α functions

Number of items sold

Graph Weights

Binary features

Step 7: Context generation

Approach

The main difference, with respect to all the previous steps, is that in this case we **can observe the features**, but we don't know how to partition the users in different classes.

By assumption, the context generation problem is tackled every two weeks. This is done in order to collect a significative number of samples before dividing the users in different classes.

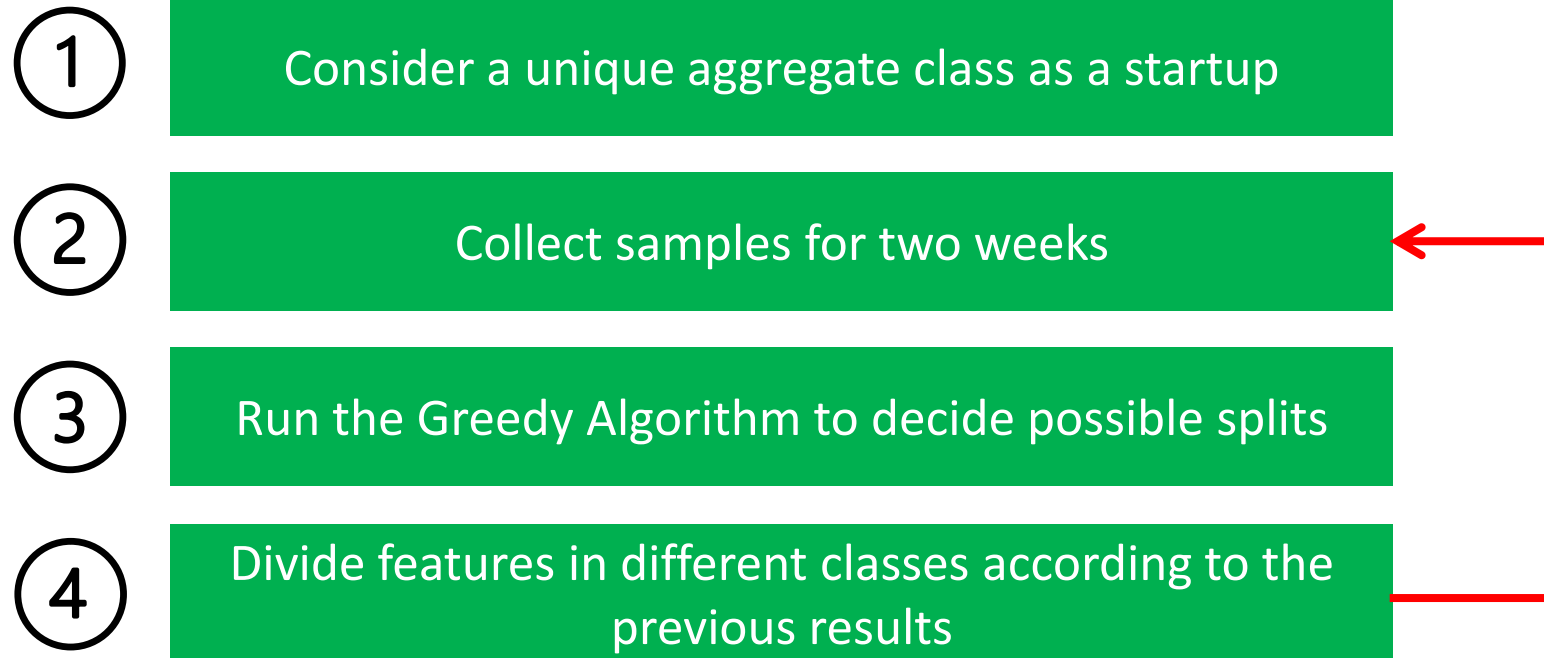
In our example, the two features are:

1. Age
2. Gender

We encoded them as binary features.

Step 7: Context generation

Steps of the algorithm:



Step 7: Context generation

Greedy algorithm

Our greedy algorithm exploits all data collected during the experiment, from the beginning.

For each user we record:

- The features
- The landing product (reward obtained according to the budget)

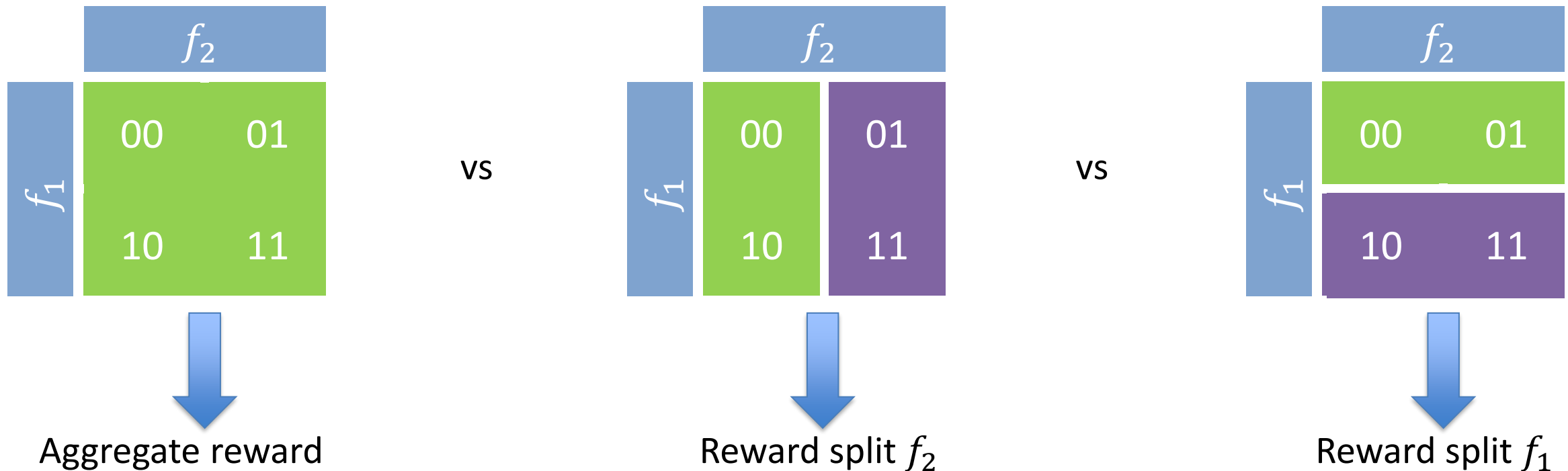
Every two weeks the algorithm is run in order to **find the best split** (according to data collected so far). Every two weeks the algorithm decides the best potential feature to be split. If the split is convenient, then it is performed, and the same evaluation is done also with the remaining feature.

We always start from a unique aggregate class, and we evaluate all the possible split options, because the splitting doesn't have to be **monotonic**.

Step 7: Context generation

Greedy algorithm - Example

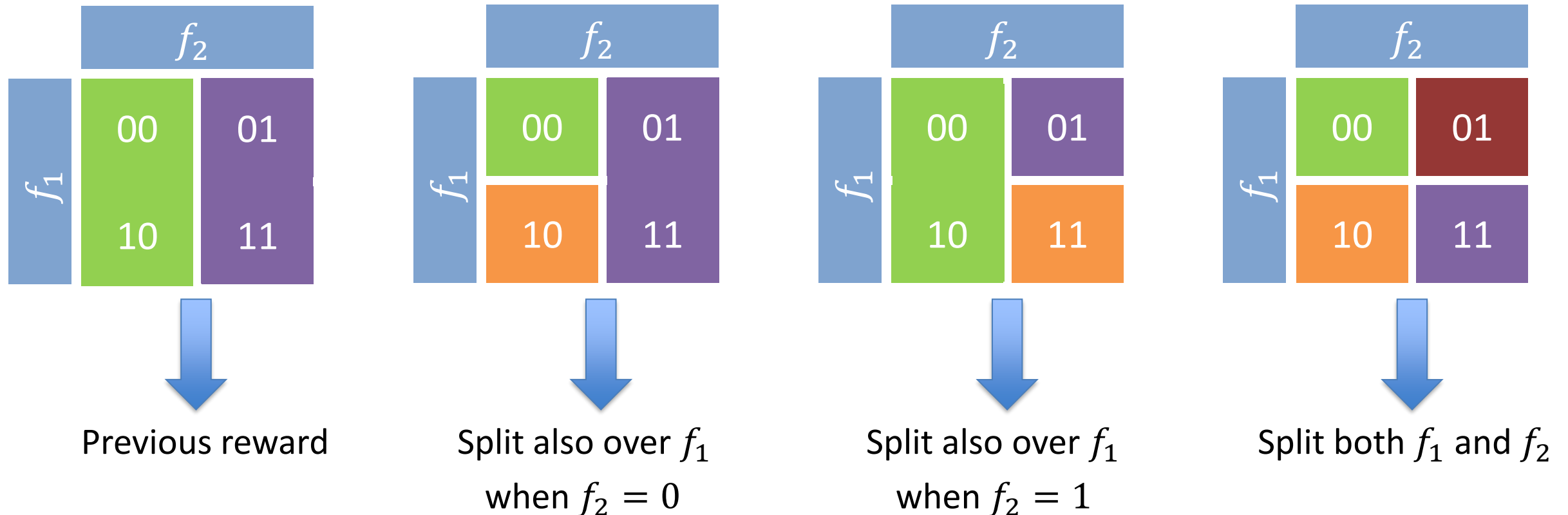
At the beginning we start evaluating whether it is more convenient to keep all aggregate, or split according to feature f_1 , or feature f_2 :



Step 7: Context generation

Greedy algorithm - Example

Suppose that the greatest reward we can obtain, is the one obtained splitting feature f_2 . We check whether it can be worth to split also with respect to f_1 inside each one of the two contexts:



Step 7: Context generation

Greedy algorithm – The split

In order to decide whether it is convenient to split or not, we started using the formula seen during the lessons:

$$\underline{p}_{c_1} \underline{\mu}_{a_{c_1}^*, c_1} + \underline{p}_{c_2} \underline{\mu}_{a_{c_2}^*, c_2} \geq \underline{\mu}_{a_{c_0}^*, c_0}$$

In our case, the best expected reward for each context (c_0, c_1, c_2) is returned by an optimizer, which receive as input the discretized values of the alpha ratios.

The estimated values for the alpha ratios of a context are computed using only samples of that context. Once we have the estimation of the alpha ratios we put them inside the optimizer, which will return the best expected reward for that context.

For what concerns the probabilities, we can keep track of the number of users for each feature and compute them accordingly.

Step 7: Context generation

Greedy algorithm – The split

Where:

$$\underline{p}_{c_1} \cdot \underline{\mu}_{a_{c_1}^*, c_1} + \underline{p}_{c_2} \cdot \underline{\mu}_{a_{c_2}^*, c_2} \geq \underline{\mu}_{a_{c_0}^*, c_0}$$

We decide to split the **context C0** into the two **subcontexts C1 and C2**

if the sum of the weighted lower bound on the best expected reward of the two subcontexts is greater or equal to the aggregate one.

Step 7: Context generation

Greedy algorithm – Empty arms and Lower Bound

The main problem of the greedy algorithm approach is that we **may have some arms** (budget's values) with **no samples**. For example, we may have exploited the arms 10, 15, 35, but not 5, 20, 25 and 30. Since we know that the alpha functions have a certain regularity, nearby values are somehow correlated. For this reason, in order to cope with the problem of arms with no samples, we decided to **exploit Gaussian Processes'** properties.

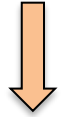
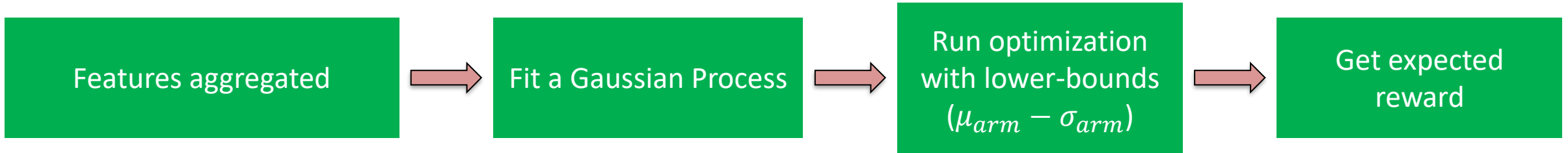
We need to pass to the optimizer the estimation of the alpha ratios computed in **each discretization** of the budget (to obtain $\underline{\mu}_{a_{c_1}^*, c_1}$), hence we fit a GP.

In order to obtain $\underline{\mu}_{a_{c_1}^*, c_1}$ (the **lower bound**) from the optimizer, and not $\mu_{a_{c_1}^*, c_1}$, we pass to the optimizer the GP's lower bound, obtained as the **subtraction** between the **mean** and the **variance**.

Step 7: Context generation

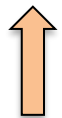
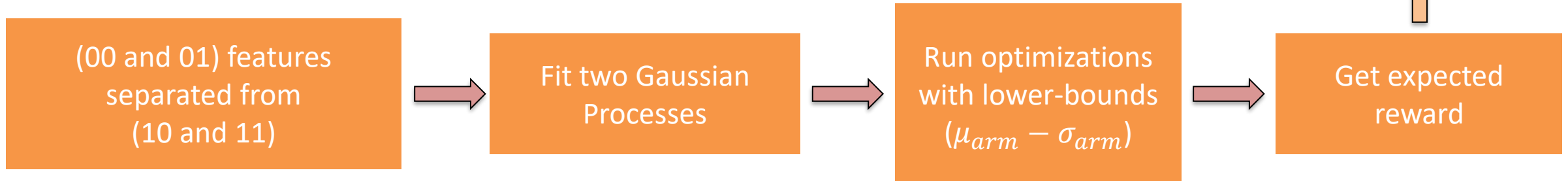
Greedy algorithm – example of one step

Aggregated model



Compare rewards and decide if splitting can be worth

Split according first feature (in first position)



Step 7: Context generation

MAB update – challenging problems

The most challenging problem we need to solve is to find a way to **re-use correctly samples from past splits** when we want to initialize a new context, which may be different from the current one.

Consider for example the case in **which all the features are aggregated**, and the total budget allocated for product P1 is 60.

The budget needs to be divided within the classes, hence, for each class of user, we are not investing 60, but a smaller value. However, we will update our observation on the arm 60 of the MAB of the “aggregated context”.

Suppose then a split is performed, from which we obtain two contexts (c1, c2), and suppose the budget allocated for product P1 in c1 is 60: this amount will be split within the classes in c1.

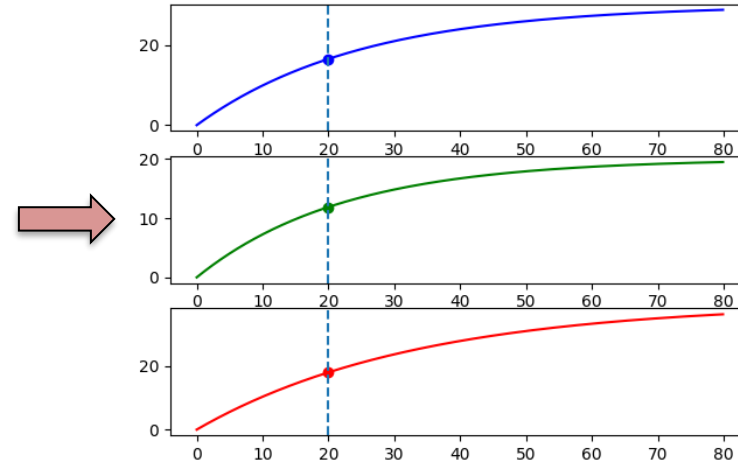
Step 7: Context generation

MAB update – challenging problems example

Aggregate case

Play arm 60 for product P1

The Environment **splits** the budget among the **three classes**
Ex. [20, 20, 20]



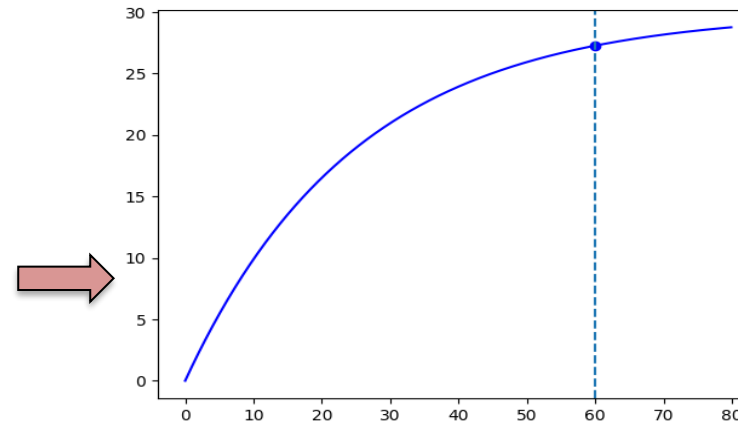
If we change context the **same arm** refers to **values sampled in different points!**

Update arm 60 on the **MAB of P1**

Context with feature 00 only (class1)

Play arm 60 for product P1

Environment **does not split** budget since in the context we have only one feature



We need to pay attention when using data collected with old different splits

Step 7: Context generation

MAB update – challenging problems

Since our objective is to re-use data from the previous weeks, we found a way to **re-balance this inequality**:

When we decide **to initialize the new created MABs** for the new **context, with old samples, we scale the value of the arm** of the old samples in order to make it coherent with the new one.

For example, if we need to use an old sample corresponding to arm 50 in a context that was “all aggregated”, and now our context contains only the features with value “00”, corresponding to class 1, we will divide 50 by the number of users that we had in the old context, and we multiply it by the number of users with feature “00” that we have in the current.

This may lead to update arms that actually do not exist (the scaling factor may produce intermediate values between two arms). We still can do the update if we exploit Gaussian Processes: we will update the function in an intermediate value that will never be used, but this contributes to improve our estimate on arms actually used thanks to the properties induced by the kernel function.

Step 7: Context generation

MAB update – Reusing Old Samples

Aggregate context [00, 01, 10, 11]

Play arm 60 for product P1

The Environment **splits** the budget among the **three classes**
Ex. [20, 20, 20]

Re-Use these samples in the context [00, 01]

Context [00, 01]

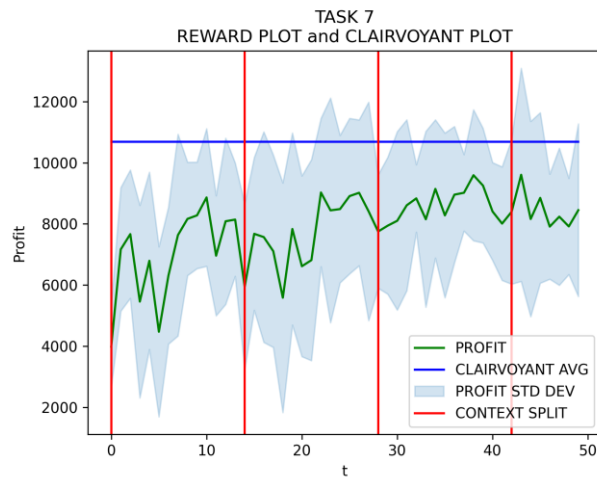
Fit a NEW MAB

Update Arm 40 with the reward of the collected sample

Step 7: Context generation

Results – considerations

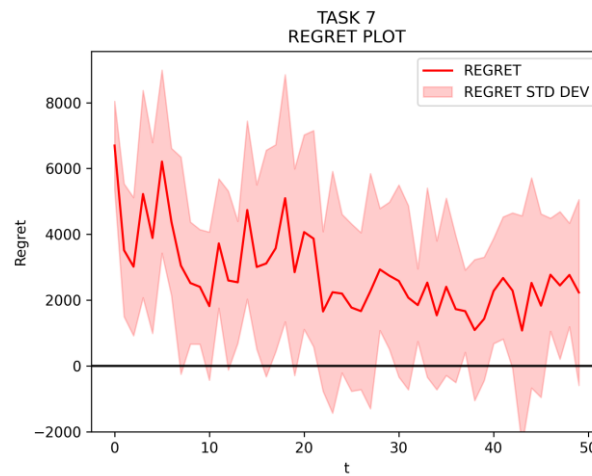
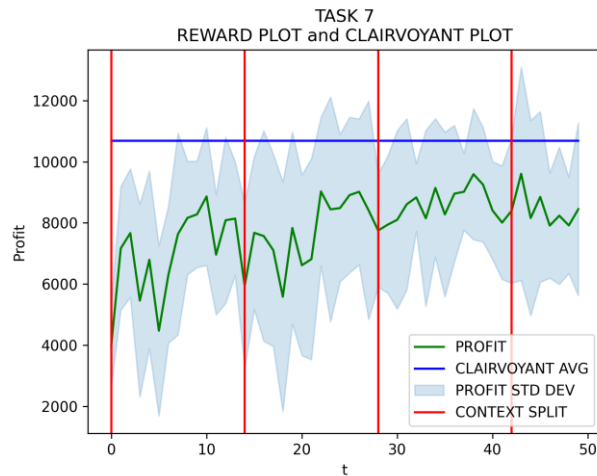
Unfortunately, we noticed that the split is not performed correctly every time. Hence, on average, it will not tend to the optimal value. Anyway, as we can see from the graph, after the first two weeks in which we have an aggregated context, the algorithm seems to learn a bit.



Step 7: Context generation

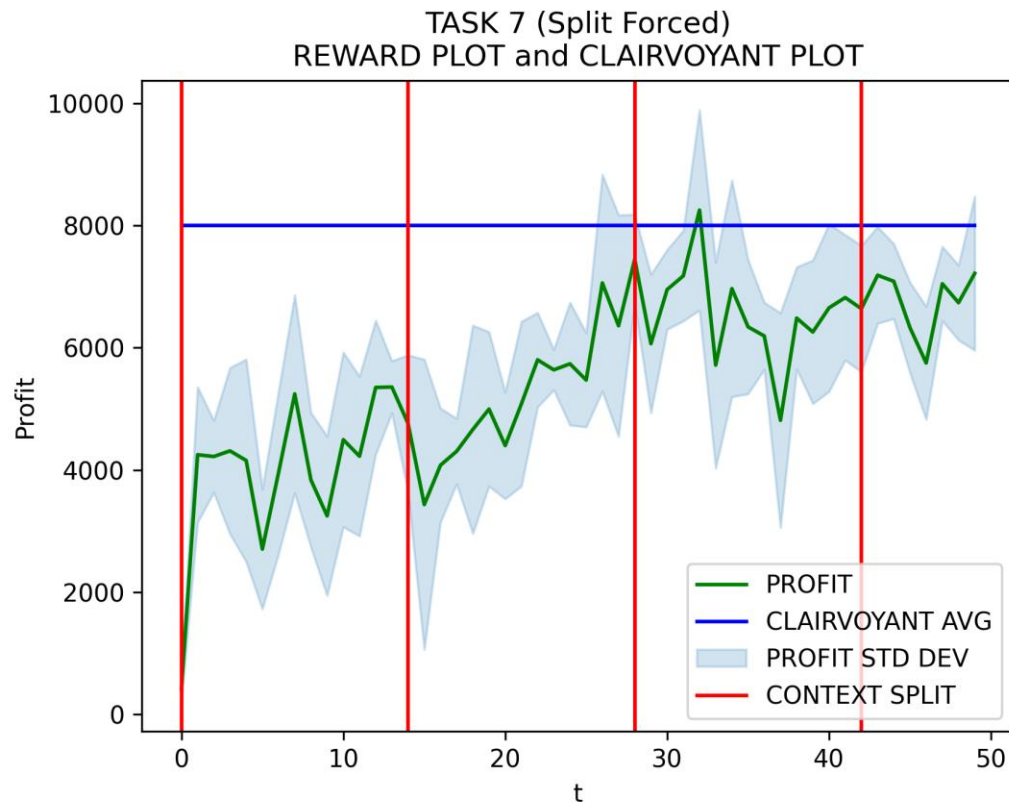
Results – considerations, further tests

A positive result we have obtained is that old samples are re-used in a correct way. We deduced this from the fact that we don't have drops in performances when we call the split algorithm starting a new MAB with old users' contributions.



Step 7: Context generation

Results – considerations, further tests

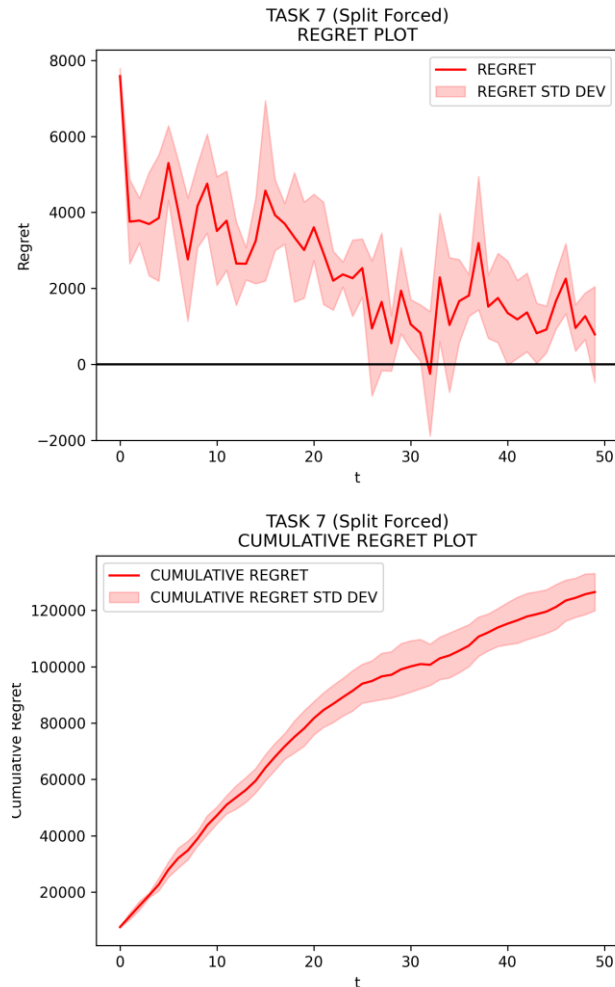


We tried (by modifying conditions in the code) to force splits in the right way. The output was promising, because as visible, after the first split (14^{th} day), it tends clearly to the optimal value. This led us to make some considerations: the main problem could probably be in the split algorithm. This could be due to several options, including:

- The fact that the algorithm is greedy (hence, it may stuck in a local optima generated by the decision of the split order)
- A lack of exploration that makes our estimation of alpha ratios bad or wrong for the subclasses
- We still do not exclude a bug in our code.

Step 7: Context generation

Results – considerations, further tests



We tried (by modifying conditions in the code) to force splits in the right way. The output was promising, because as visible, after the first split (14^{th} day), it tends clearly to the optimal value. This led us to make some considerations: the main problem could probably be in the split algorithm. This could be due to several options, including:

- The fact that the algorithm is greedy (hence, it may stuck in a local optima generated by the decision of the split order)
- A lack of exploration that makes our estimation of alpha ratios bad or wrong for the subclasses
- We still do not exclude a bug in our code.



Alessandrini Luca

Fabris Matteo

Portanti Mattia

Portanti Samuele

Venturini Luca