BLUECOW

L'ENERGY DRINK.

Bressan Manuel – Carrara Davide – Tombari Filippo – Zanotti Daniela

Data Intelligence Applications – A.Y. 20/21

POLITECNICO
MILANO 1863

Choice of the product:

In order to choose the product, we first analysed the specific requirements of our problem. We need a **consumable** item that can be bought **online**, has enough **margin** to allow for a targeted advertising and should be bought again in the future. The main category in which we found these characteristics is the one of **food and beverages**. In particular we focused on all the **premium range** of these products that can't be normally be bought in a supermarket, have a higher price point among other competitors and focus a lot on advertising in order to justify their exclusivity and value.

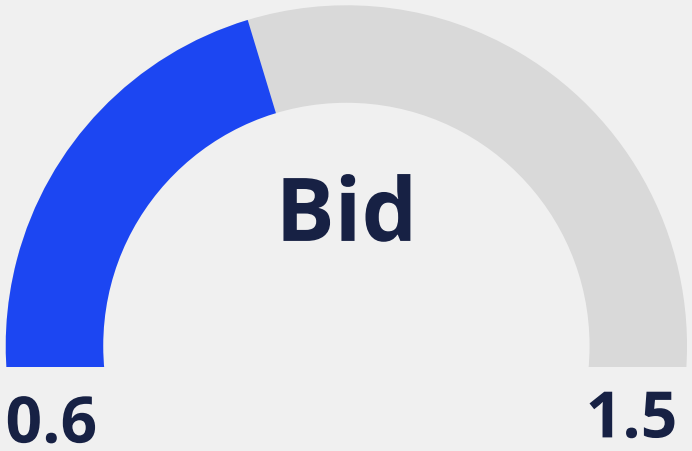
COMPETITOR

A notable example of a product that gives much of its success to their omnichannel and pervasive advertising is **RedBull**.
This brand, over the years, has sponsored all type of adrenaline-filled events and **sport competitions**, but, on the other hand, has also produced marketing campaign entirely devoted to **students** to enhance their study productivity.

This has allowed the brand to have a fairly higher price point with respect to their other competitors and, also, to dominate the market segment. So, we devised a fierce competitor of RedBull, BLUECOW.

We decided to sell a **box of 6 cans** of our energy drink, in order to have a product that could be bought up to 4 times in a month. We decided that our **price** could range in the interval **[5, 14]** whilst our **bid** could range in the interval **[0.6, 1.5]**.

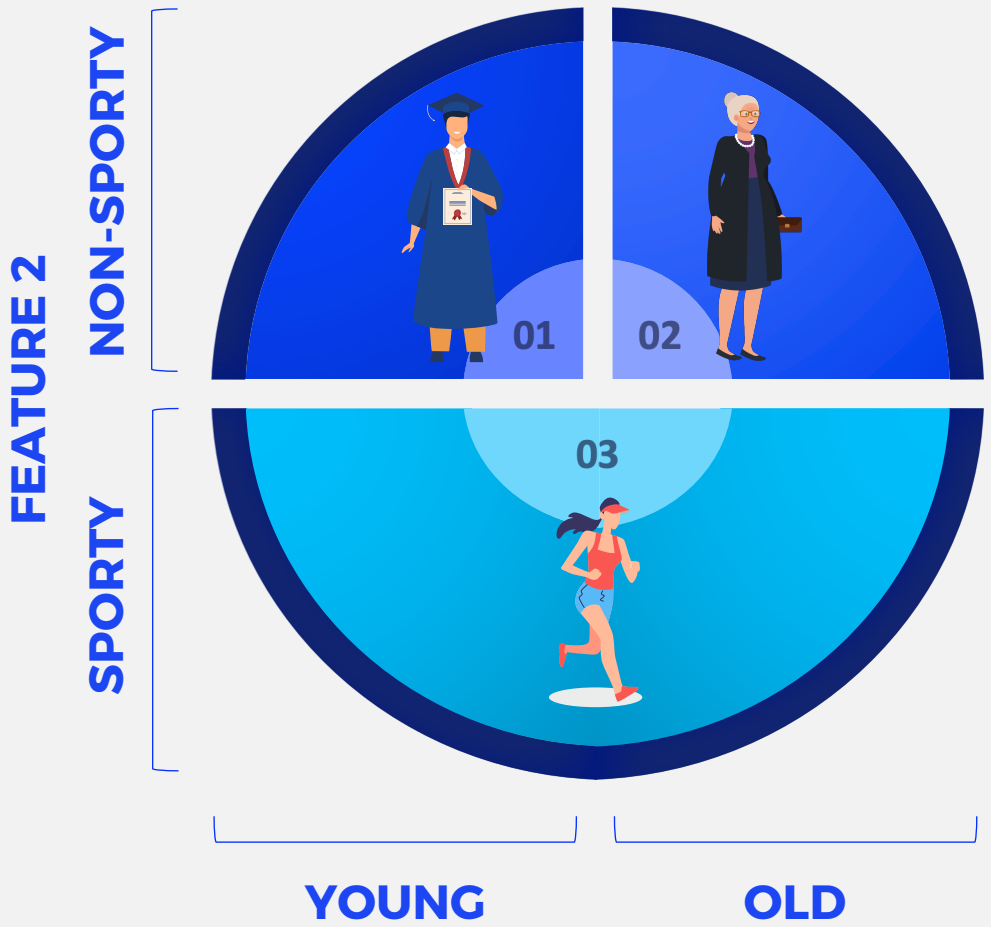
The price of production is set to be 4€ for box so our **margin** is given by $m(p) = p - 4$



We then chose two binary features: **young vs. old** and **sporty vs. not sporty**. The three possible categories that we kept were:

Young and NON-Sporty people:

characterized by a **high use of internet**, with a **lower propensity to spend money** but **more interested** into our product



Old and NON-Sporty people:

characterized by a **low use of internet**, with a **higher propensity to spend money** but **less interested** into our product;

Sporty people:

here there are **both young and old people**, so there is an overall medium use of internet, a medium propensity to spend money, but still a **high interest in our product**.

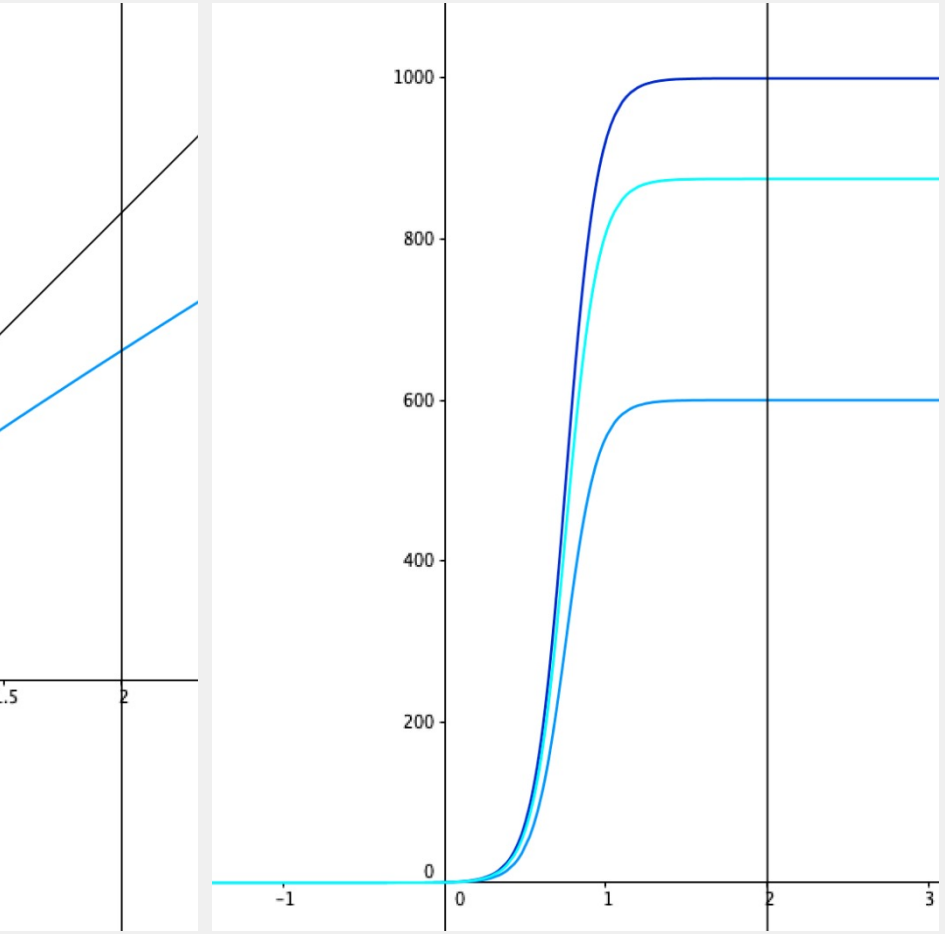
In the following slides we are going to present how the different classes are characterized.
The generic customers' class is in this way constituted:

A stochastic number of **daily clicks** of new users depending on the **bid**

A stochastic **cost per click** as a function of the **bid**

A **conversion rate** function given a **price**

A distribution probability over the **number of times** the user will come back **30 days** after the first purchase



A stochastic number of **daily clicks** of new users depending on the **bid**

For this function we opted for a **sigmoid** that saturates at a **different value for each class**.

$$\frac{e^{10*(x-0.75)}}{1+e^{10*(x-0.75)}} * NoA$$

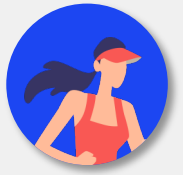
We thought that this made sense because there is somewhat a **competitive value** for the bid in which you start winning the auctions.

Number of Accesses

1000

Number of Accesses

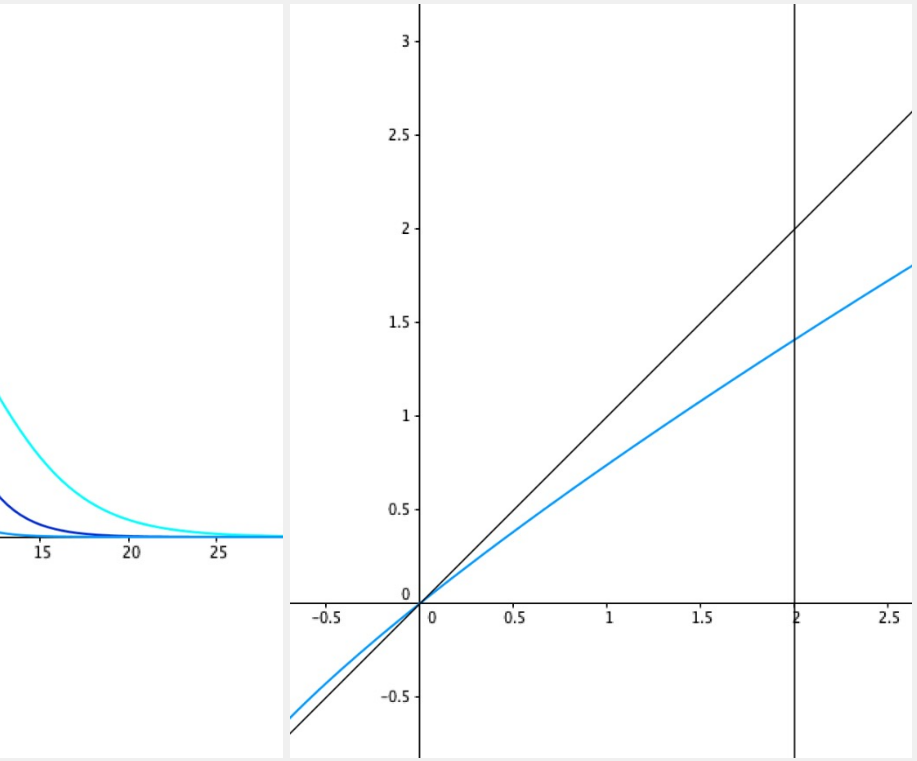
600



Number of Accesses

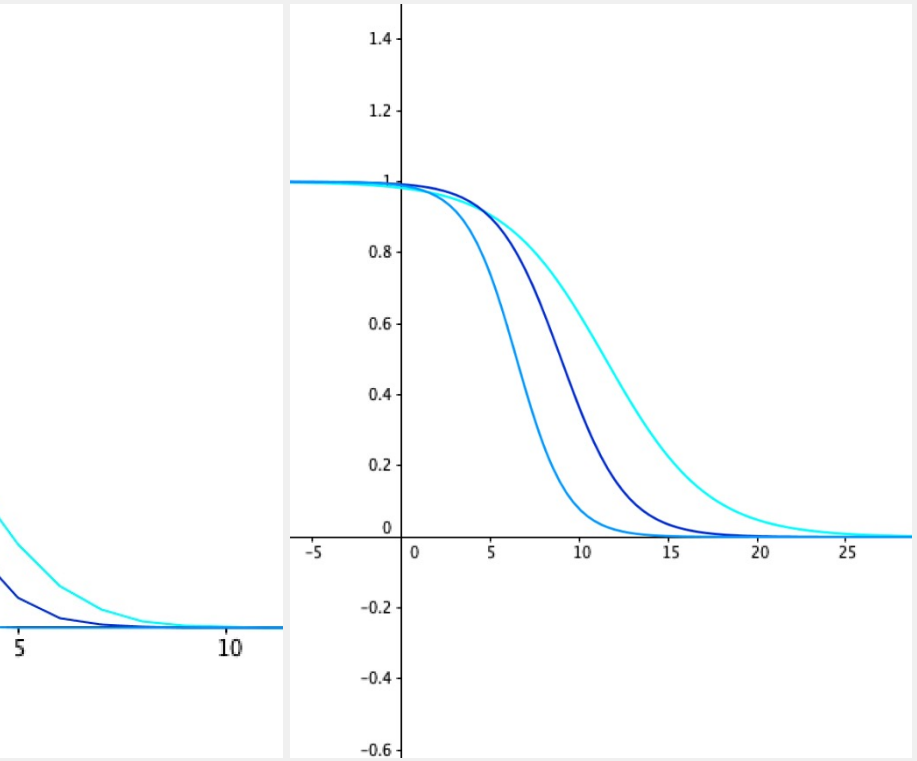
875

A stochastic **cost per click** as a function of the **bid**



In this case we started with the information that the cost per click is **always smaller** than the winning bid.
To evaluate how much “smaller” it is, we looked online at **real cases** of bidding done by other people on AdSense. In those cases, the trend didn’t appear to be linear but, instead, somewhat **sublinear**, so we opted for the function depicted.

No difference among classes



A conversion rate function given a **price**

Also in this case we used a **sigmoid**, that starts from 1 at price 0 and, according to each customers' class, has a different mean and a different descending slope.

This made sense to us because, obviously, when the price is very low everybody would buy our six-pack and when the price rises every class has its perceived right price.

Mean

9

Velocity

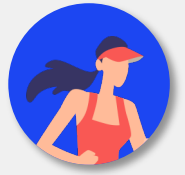
0.55

Mean

6.5

Velocity

0.7

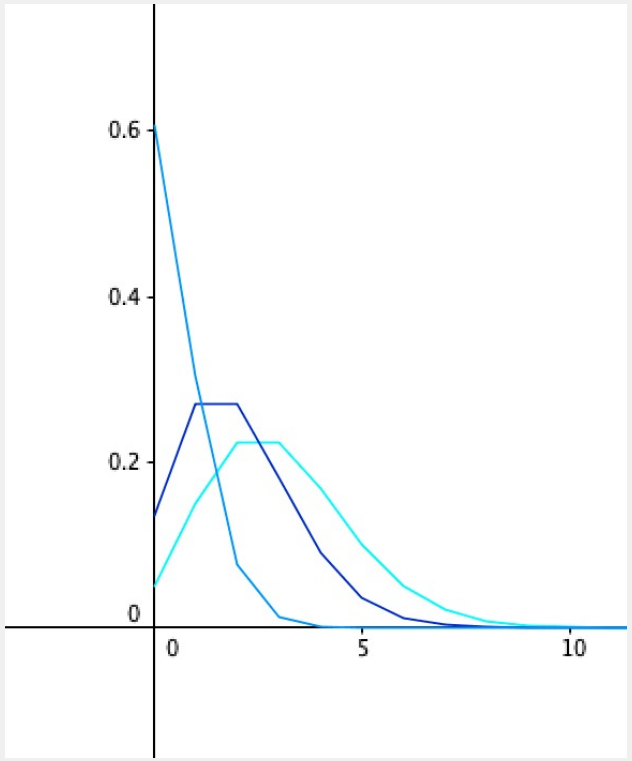


Mean

11.5

Velocity

0.35



A distribution probability over the number of times the user will **come back 30 days after** the first purchase

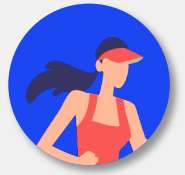
In this case a good mathematical model to represent this phenomenon is the **Poisson distribution** with a parameter **λ** that is the mean of returns and is **different for each class**.

Lambda

2

Lambda

0.5



Lambda

3

Objective:

Find through an online learning method the optimal (price, bid) that maximize the reward.

Pricing & Bidding

Solving first the two problems separately with the other parameter fixed

Context Split on Pricing

Create an algorithm able to evaluate whether a certain splitting is better in terms of rewards or not.

All together

Put all the knowledge gathered into a unique algorithm able both to determine the right splitting and all the joint price and bid couple for all the classes.

Formulation of the objective function

STEP 1 - AGGREGATED

We have first considered the **offline** case, in which there are not time varying variables.

The **objective function** we want to maximize in the aggregated case (where there is only a couple of price and bid) is:

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

$n_i(b)$: $number\ of\ accesses\ of\ class\ i$

$q_i(p)$: $conversion\ rate\ of\ class\ i$

$r(p)$: $margin$

N_i: $mean\ number\ of\ returns\ for\ class\ i$

$c(b)$: $cost\ per\ click$

STEP 1 - AGGREGATED

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

This function is **non-linear** and moreover it is **not convex nor quadratic**. For these reasons we cannot use easy optimization algorithm in order to find its maximum.

Given a set of bids and prices, how can we find the best couple?

Since we are considering an aggregated scenario there is no way to divide the three class-related problems. As a consequence we cannot decouple the pricing and bidding problem: indeed, the pricing related term for each class has to be weighted by the number of clicks, which is bid-dependent.

Subsequently, we need to evaluate the objective function for every possible couple (price, bid) and select the one providing the highest reward.

STEP 1 - AGGREGATED

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

$O(n_b n_p)$

This is the **complexity** needed to check every cell of the matrix where n_b and n_p are respectively the number of different arms for the **bids** and for the **prices**.

STEP 1 - DISAGGREGATED

In this new situation we are still considering the **offline** case but the results are sligthly different.

The **objective function** we want to maximize in the disaggregated case (where we can choose a differente (price, bid) for every class) is:

$$\sum_{i=1}^{3} \max_{p_i, b_i} \{n_i(b_i)[\, q_i(p_i) r(p_i)(N_i + 1) - c(b_i)]\}$$

$n_i(b)$: *number of accesses of class i*

$q_i(p)$: *conversion rate of class i*

$r(p)$: *margin*

N_i: *mean number of returns for class i*

$c(b)$: *cost per click*

STEP 1 - DISAGGREGATED

$$\sum_{i=1}^{3} \max_{p_i, b_i} \{n_i(b_i)[\, q_i(p_i) r(p_i)(N_i + 1) - c(b_i)]\}$$

Given a set of bids and prices, how can we find the best couple?

We are maximizing three different problems, one for each class. For each one of them, the price related terms (conversion rate and margins) have a positive impact on the objective function, meaning that an increase in q(p)*r(p) will automatically lead to an increment in the final value we want to maximize.

Thus, we can solve the **pricing problem independently** from the general one. In practice, it suffices to compute the value of q(p)*r(p) for each given price and select the optimal one. At this point we just need to compute the value of the objective function for each possible bid, using the optimal price found above. Lastly, we select the bid corresponding to the highest reward.

STEP 1 - DISAGGREGATED

$$\sum_{i=1}^{3} \max_{p_i, b_i} \{n_i(b_i)[\, q_i(p_i) r(p_i)(N_i + 1) - c(b_i)]\}$$

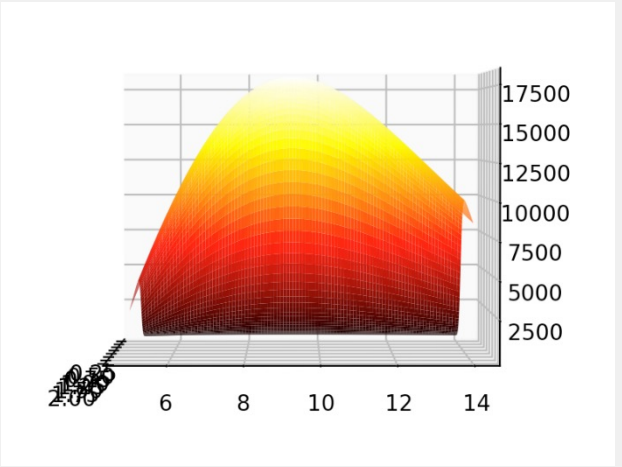
The total reward will be given by the sum of the rewards of the three problems. The optimal strategy will be represented by $\underline{p}$ and $\underline{b}$, composed by optimal prices and bids found.

$O(n_b + n_p)$

$O(n_p)$ is the complexity due to solving the pricing subproblem by enumeration of $q_i(p_i)*r_i(p_i)$. Once we find the optimal price, the bidding problem can be solved in an analogous way with complexity $O(n_b)$.

STEP 1

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$



AGGREGATED

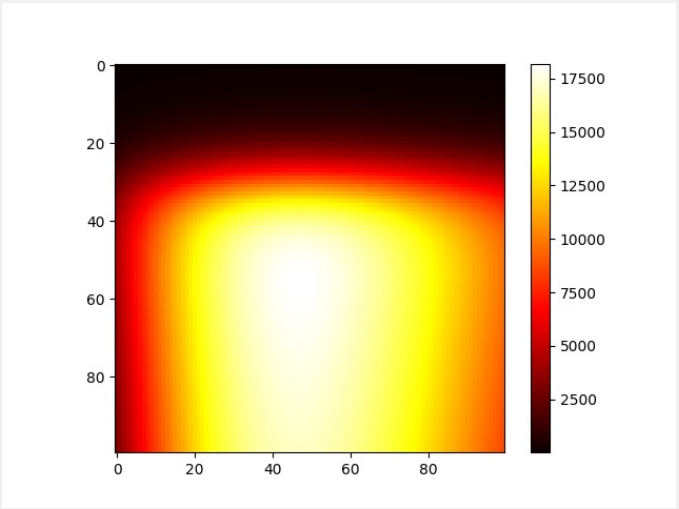
=

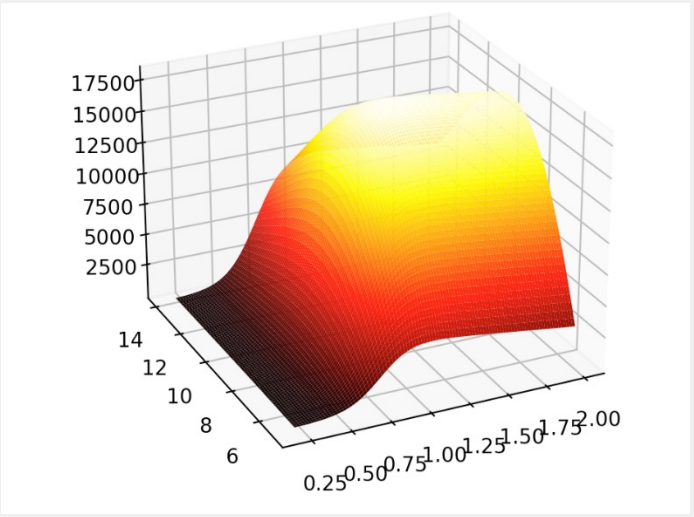
OPTIMAL PRICE-BID

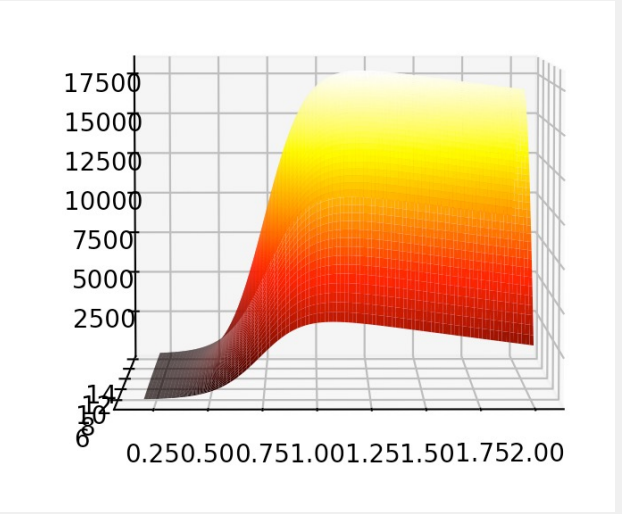
(9, 1.2)

OPTIMAL REWARD

18140.28







STEP 1

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

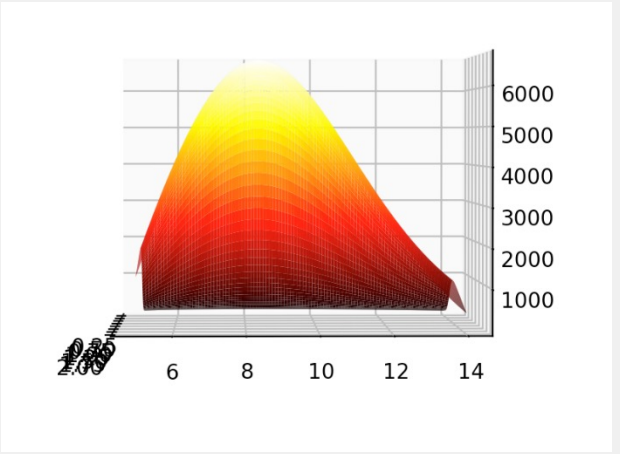
=

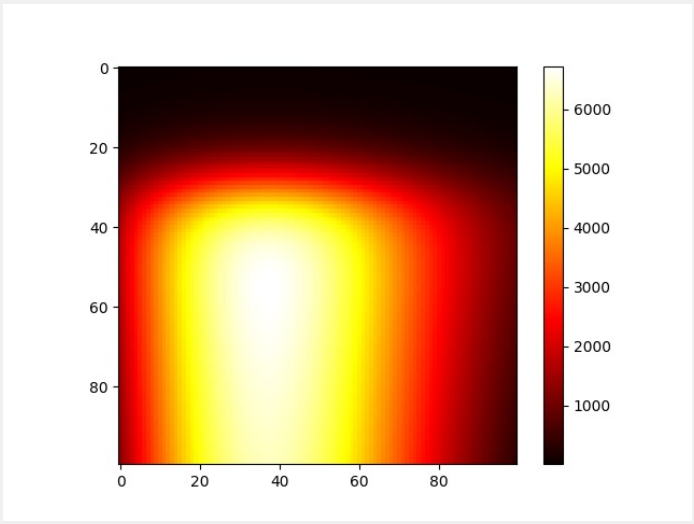
OPTIMAL PRICE-BID

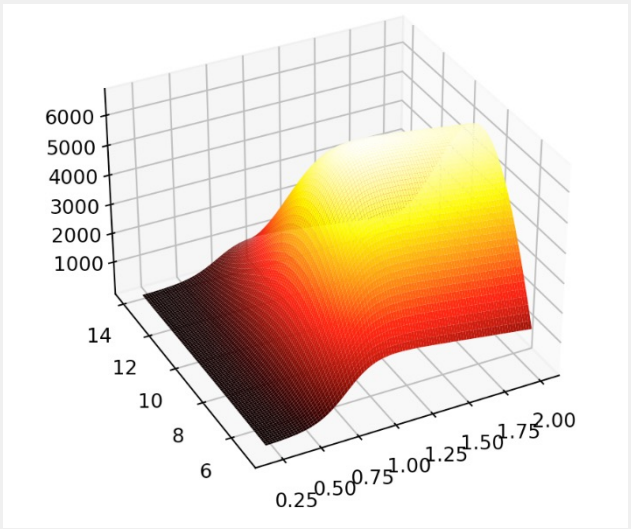
(8, 1.2)

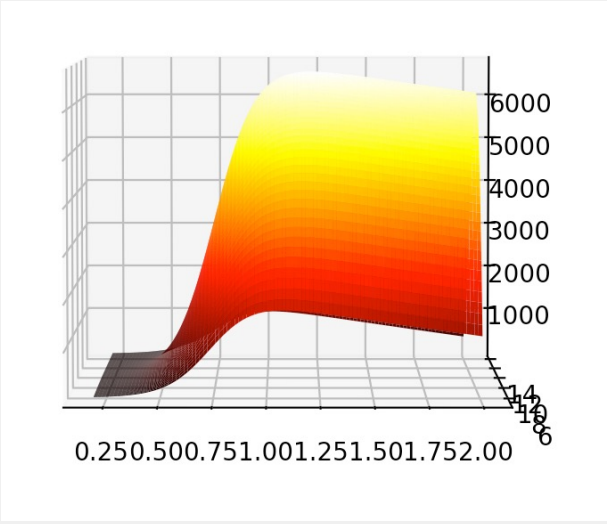
OPTIMAL REWARD

6656.63









STEP 1

$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

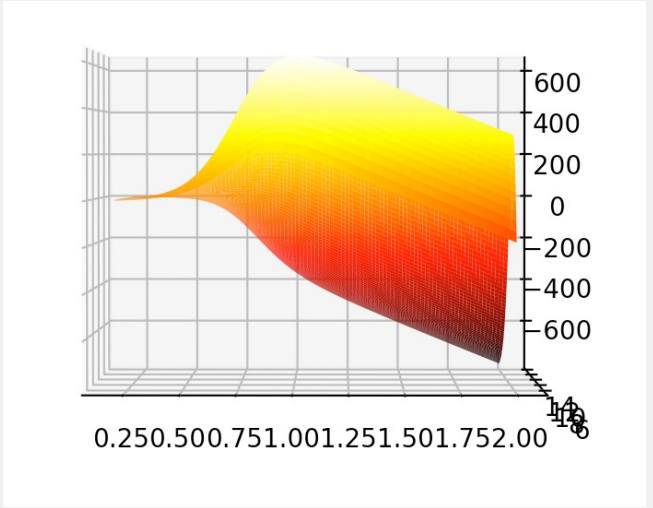
=

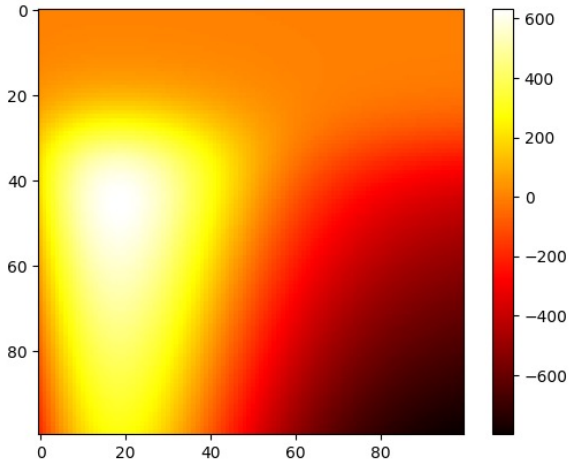
OPTIMAL PRICE-BID

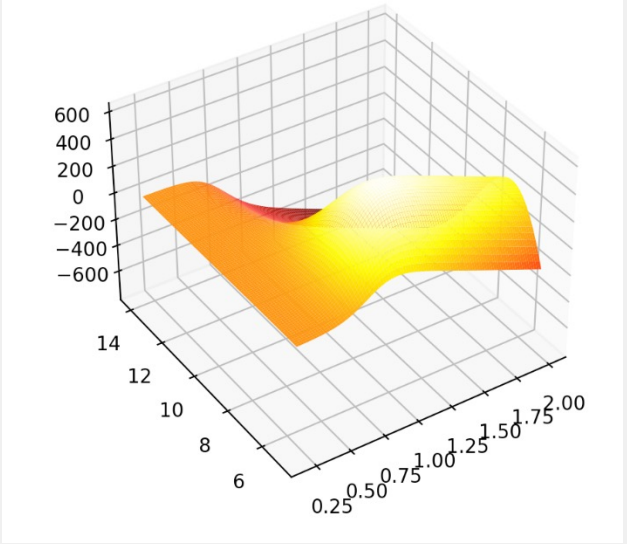
(7, 1)

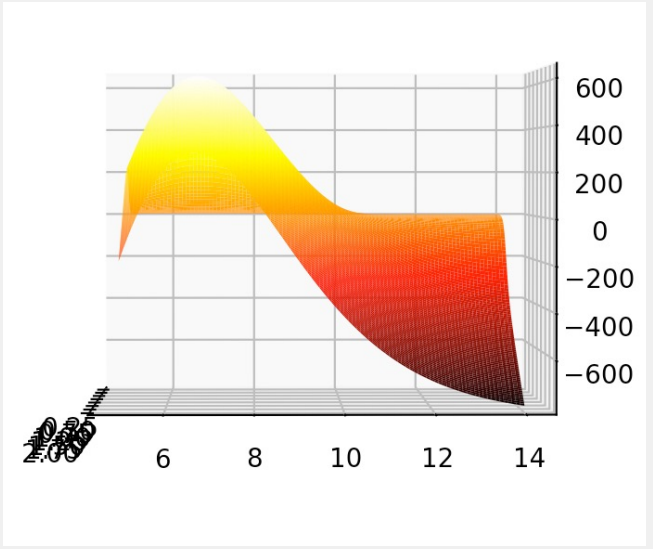
OPTIMAL REWARD

620.54



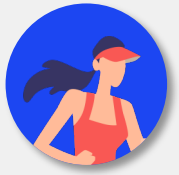






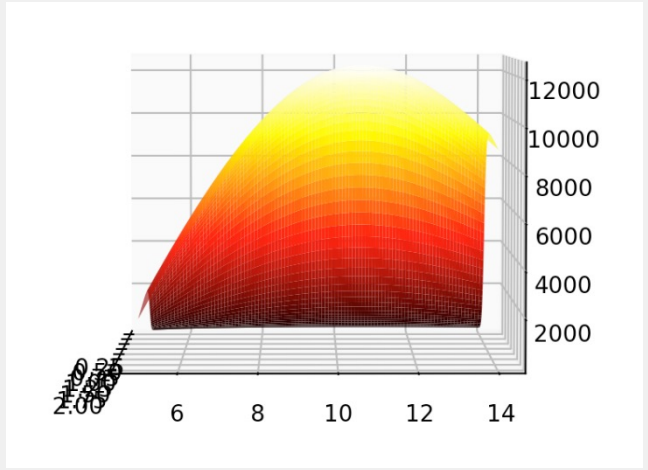
STEP 1

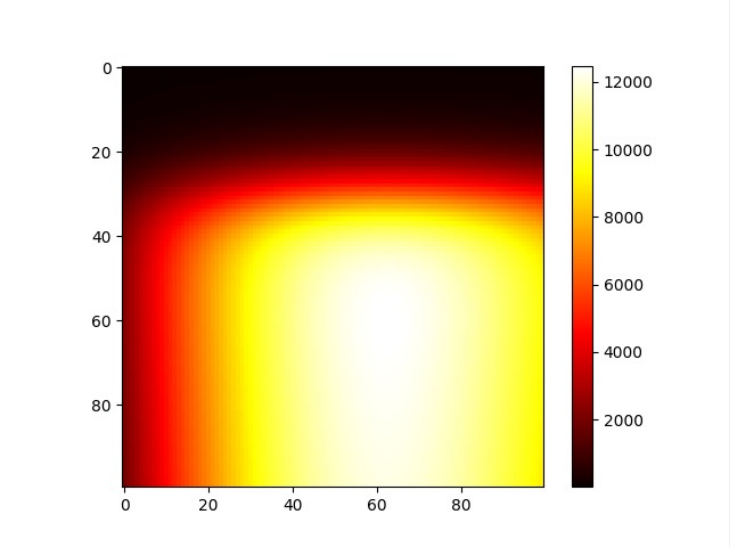
$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

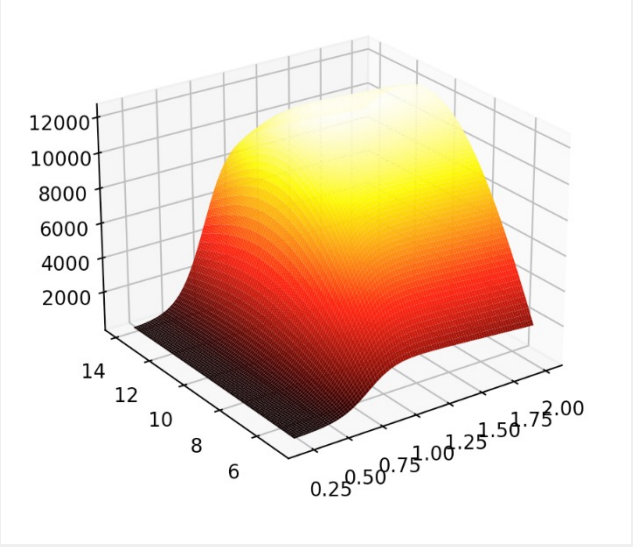


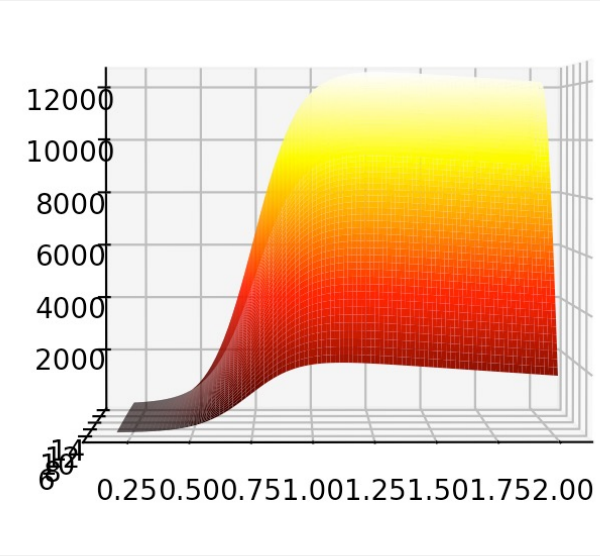
= **OPTIMAL PRICE-BID** **(11, 1.3)**

OPTIMAL REWARD **12439.62**









STEP 1

$$\sum_{i=1}^{3} \max_{p_i, b_i} \{n_i(b_i)[\, q_i(p_i) r(p_i)(N_i + 1) - c(b_i)]\}$$

DISAGGREGATED

=

OPTIMAL PRICE-BID	OPTIMAL REWARD
8, 1.2	
7, 1	19716.79
11, 1.3	

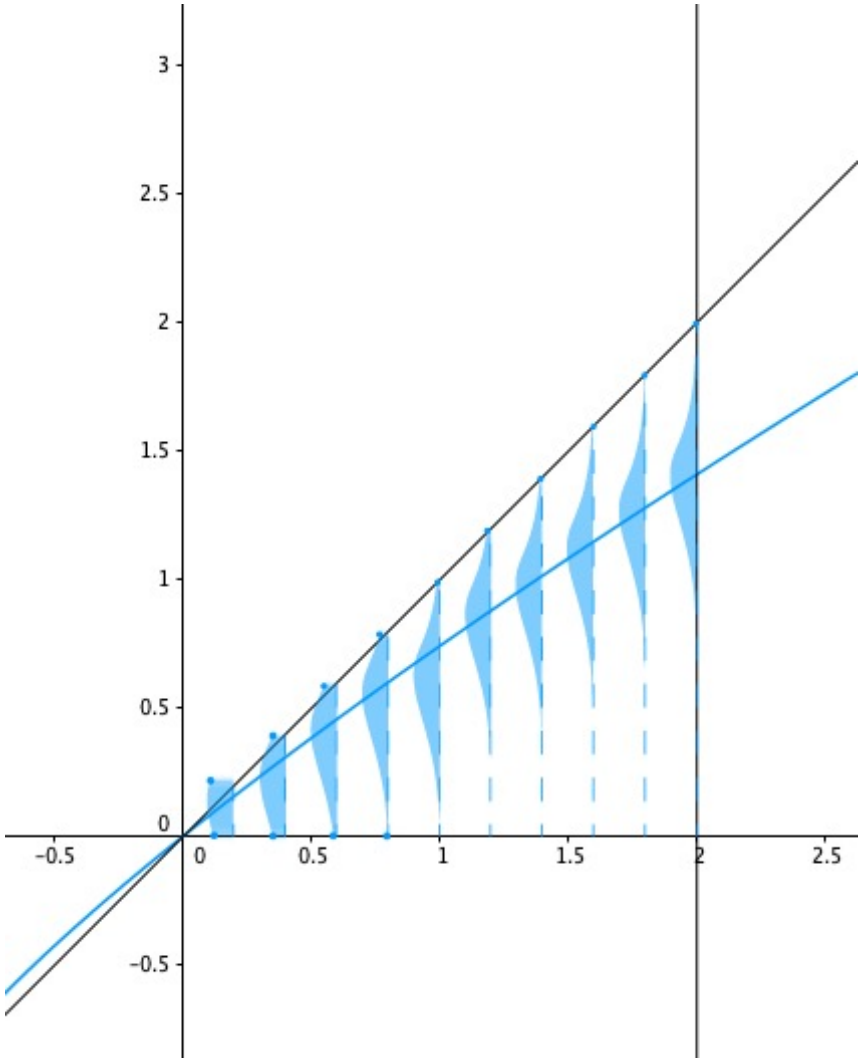
Online learning version of our optimization problem

STEP 2 – COST PER CLICK

Regarding the **cost per click** we added a **stochastic term** distributed as a **N(0, 0.01).**

However we know that the cost per click can neither be greater than the bid nor negative. As a consequence we limited our distribution by providing a **maximum** and a **minimum**. Note that the impact of this approach is significant only for small values of the bid:

	Bid = 0.4	Bid = 1.2 (optimal)	Bid = 1.6
Probability of negative cost per click	0.10%	0%	0%
Probability of Cost per click > bid	18.1%	0.06%	0%



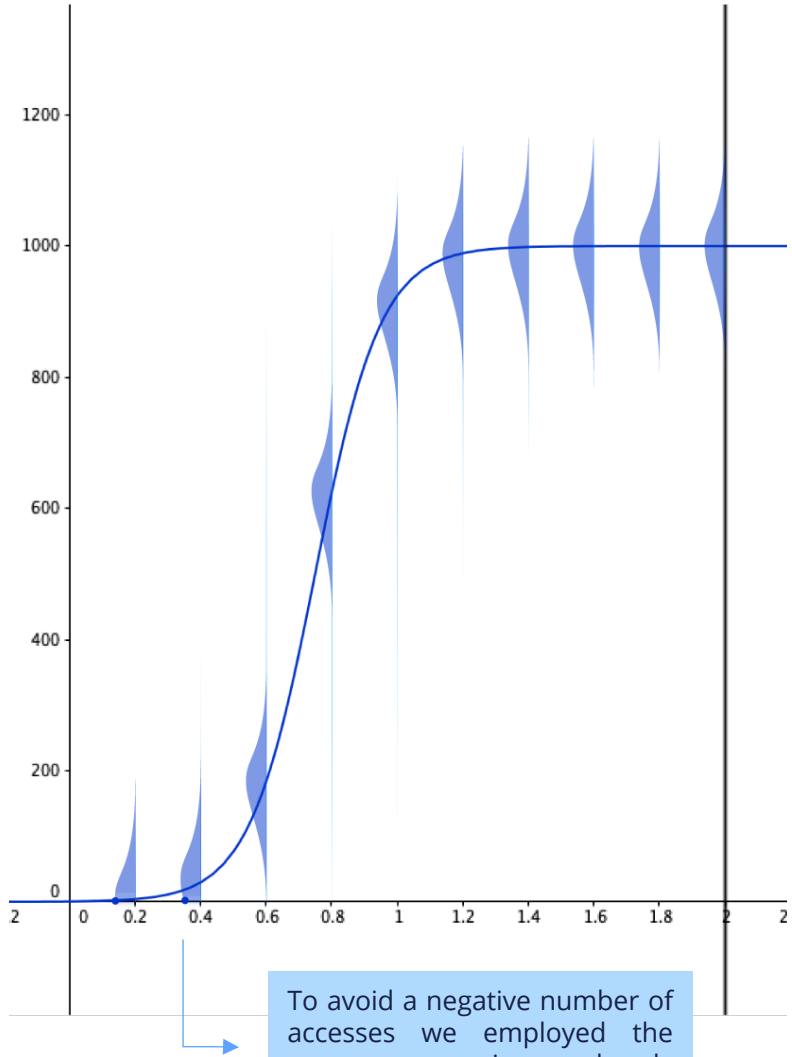
STEP 2 – NUMBER OF ACCESSES

For each class, we added to the deterministic **number of accesses** proposed before a **stochastic part**. In practice, given a bid ***b***, we assumed the daily number of accesses to be distributed as a $\mathbf{n_i(b) + N(0, 50^2)}$

For the class **sporty**, in the optimal case the daily number of accesses will be in **[775, 975]** with probability **0.95**

For the class **young**, in the optimal case the daily number of accesses will be in **[900, 1100]** with probability **0.95**

For the class **old**, in the optimal case the daily number of accesses will be in **[500, 700]** with probability **0.95**



To avoid a negative number of accesses we employed the same reasoning already presented for the cost per click

STEP 2 – FEEDBACK

We imagine a scenario in which the **feedback** over the **number of clicks** and of **purchases** is delayed by **one day**. For what concerns the number of returns, on the other hand, the information is provided at the **end of the 30 days** (namely, if 100 users buy our product at day 94, we will receive the information about their returns only at day 124).

PSEUDOCODE

```
class Environment:

    def __init__(self, n_arms, probabilities, margins, poissons, return_time=None, prices=None, bid=None):
    #initialize class


    def round(self, pulled_arm, user_c, n_trials=1):
        successes = np.random.binomial(n_trials, self.probabilities[user_c.index][pulled_arm])
        number_returns = np.random.poisson(successes * self.poissons[user_c.index])
        gain = self.margins[pulled_arm] * (number_returns + successes)
        reward = compute_reward(b=self.bid, gain=gain, n_trials=n_trials)
        return successes, reward, number_returns
```

PSEUDOCODE

```
class Environment:

    def __init__(self, n_arms, probabilities, margins, poissons, return_time=None,
prices=None, bid=None):
    #initialize class


    def round(self, pulled_arm, user_c, n_trials=1):
        successes = np.random.binomial(n_trials, self.probabilities[user_c.index][pulled_arm])
        number_returns = np.random.poisson(successes * self.poissons[user_c.index])
        gain = self.margins[pulled_arm] * (number_returns + successes)
        reward = compute_reward(b=self.bid, gain=gain, n_trials=n_trials)
        return successes, reward, number_returns
```

Draw the number of purchases given the number of accesses and the conversion rate

STEP 2

PSEUDOCODE

```
class Environment:

    def __init__(self, n_arms, probabilities, margins, poissons, return_time=None,
prices=None, bid=None):
    #initialize class


    def round(self, pulled_arm, user_c, n_trials=1):
        successes = np.random.binomial(n_trials, self.probabilities[user_c.index][pulled_arm])
        number_returns = np.random.poisson(successes * self.poissons[user_c.index])
        gain = self.margins[pulled_arm] * (number_returns + successes)
        reward = compute_reward(b=self.bid, gain=gain, n_trials=n_trials)
        return successes, reward, number_returns
```

Compute the total number of returns given the number of purchases

PSEUDOCODE

```
class Environment:

    def __init__(self, n_arms, probabilities, margins, poissons, return_time=None,
prices=None, bid=None):
    #initialize class


    def round(self, pulled_arm, user_c, n_trials=1):
        successes = np.random.binomial(n_trials, self.probabilities[user_c.index][pulled_arm])
        number_returns = np.random.poisson(successes * self.poissons[user_c.index])
        gain = self.margins[pulled_arm] * (number_returns + successes)
        reward = compute_reward(b=self.bid, gain=gain, n_trials=n_trials)
        return successes, reward, number_returns
```

Compute the total gain

PSEUDOCODE

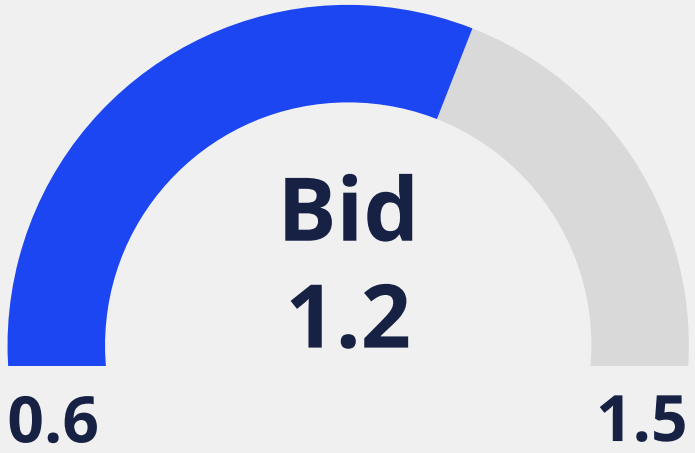
```
class Environment:

    def __init__(self, n_arms, probabilities, margins, poissons, return_time=None,
prices=None, bid=None):
    #initialize class


    def round(self, pulled_arm, user_c, n_trials=1):
        successes = np.random.binomial(n_trials, self.probabilities[user_c.index][pulled_arm])
        number_returns = np.random.poisson(successes * self.poissons[user_c.index])
        gain = self.margins[pulled_arm] * (number_returns + successes)
        reward = compute_reward(b=self.bid, gain=gain, n_trials=n_trials)
        return successes, reward, number_returns
```

Subtract negative term due to cost per click

Pricing Problem with fixed bid and NO context generation

UCB1 and TS approach

STEP 3 – AGGREGATED PRICING PROBLEM



AGGREGATED

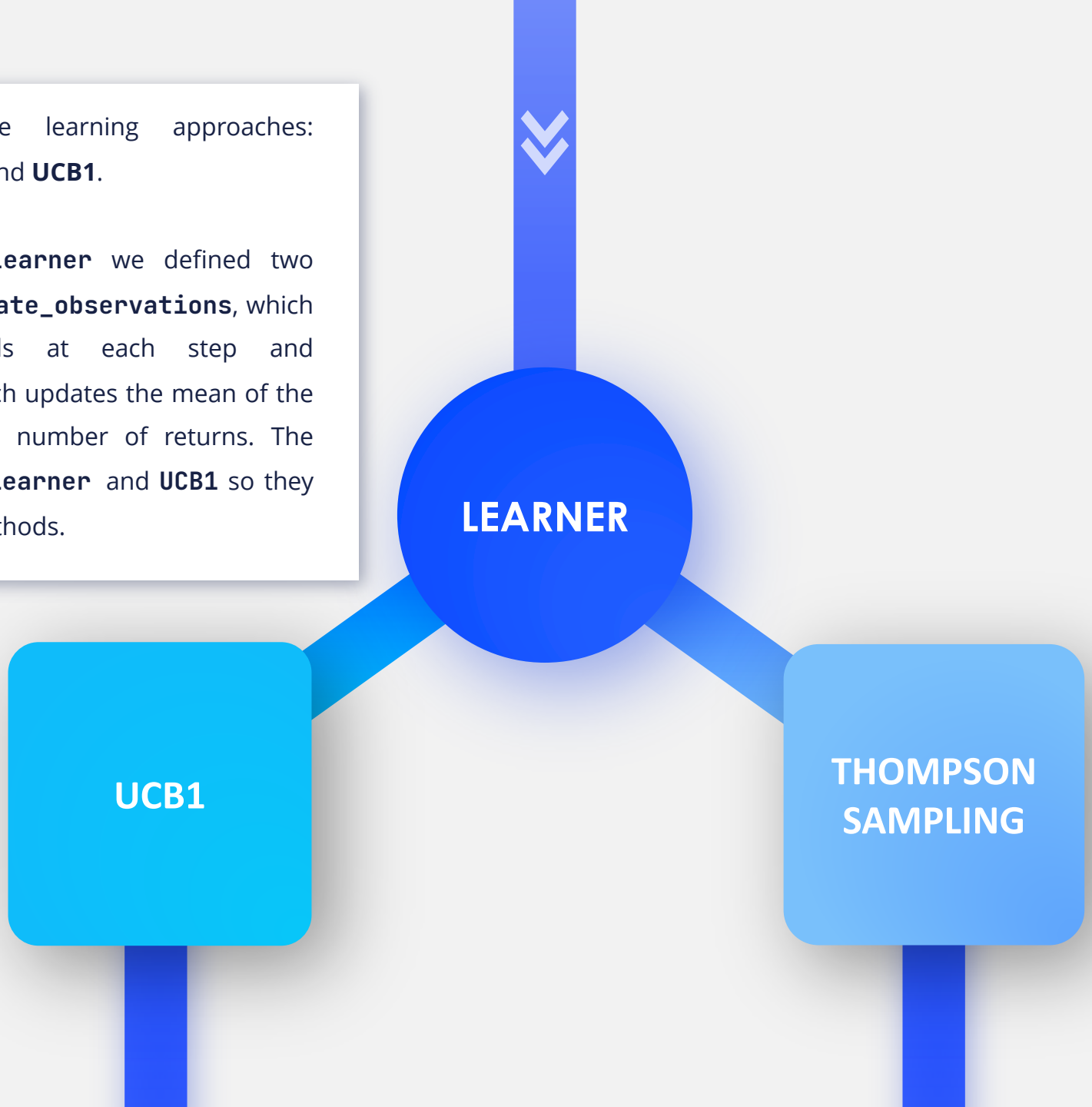
In this step we consider the **bid fixed (1.2)** and we optimize only the **pricing problem**. We adopt the following assumptions:

- **Aggregate data**: we do not discriminate between the classes at this stage
- The number of daily clicks n and the daily cost per click c are **known**

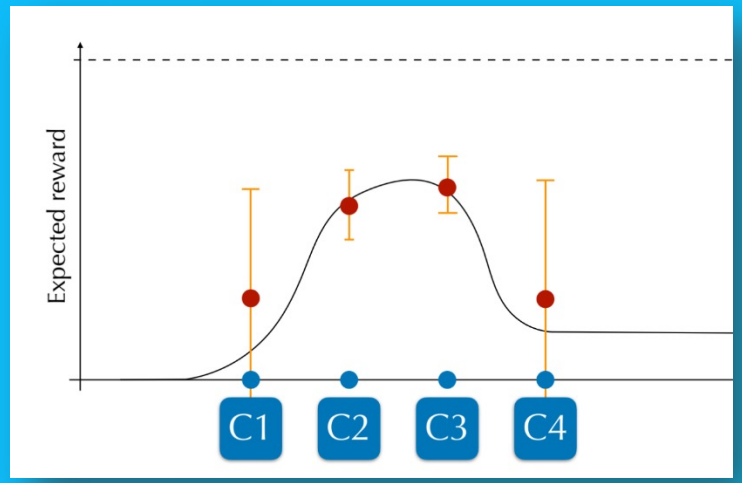
In this case the **optimal price** is 9.

Two different online learning approaches: **Thompson Sampling** and **UCB1**.

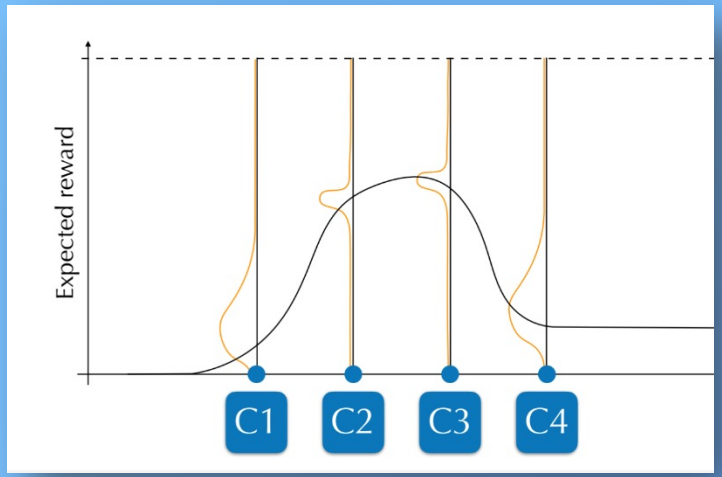
In the father class **`Learner`** we defined two different methods: **`update_observations`**, which updates the rewards at each step and **`update_poisson`** which updates the mean of the poisson modelling the number of returns. The "child" classes are **`TS_learner`** and **`UCB1`** so they inherit the previous methods.



UCB1



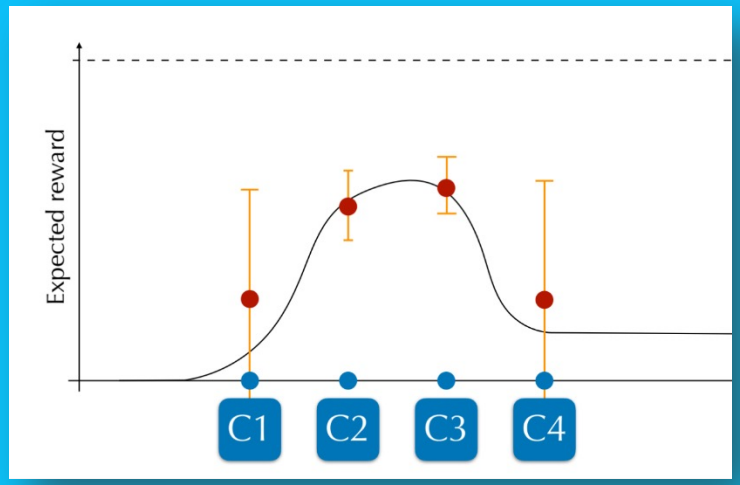
THOMPSON SAMPLING



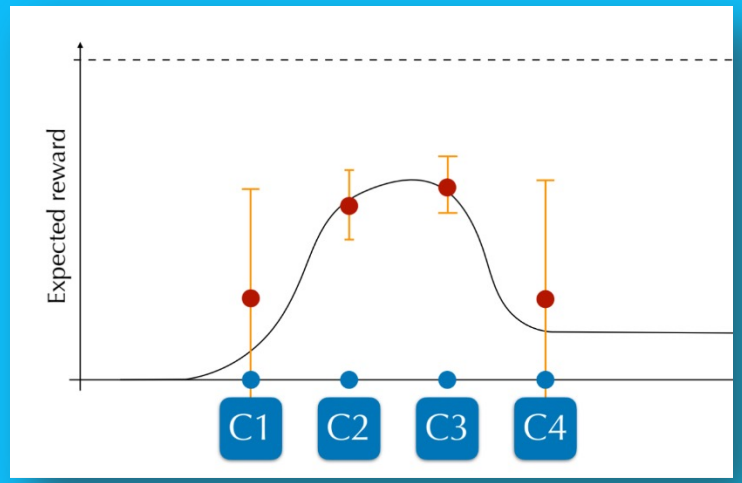
UCB1

UPPER CONFIDENCE BOUND ALGORITHM

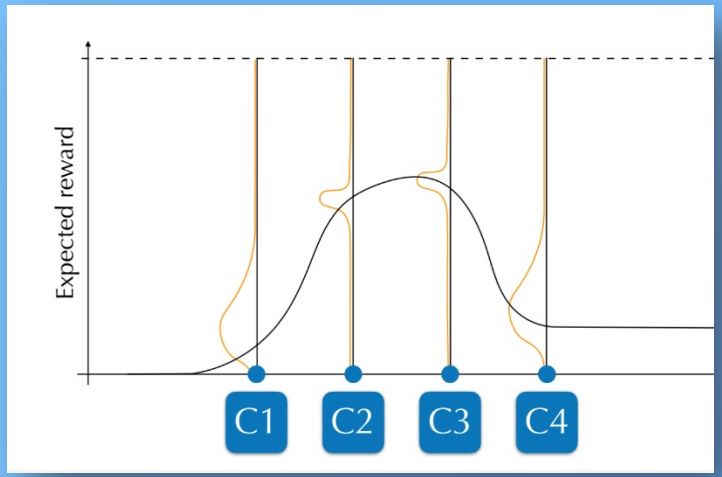
- Associate to each price a **confidence interval** for the **conversion rate**
- Choose the arm providing the **best upper bound**
- Each arm is chosen **at least once** at the beginning of the experiment
- After each round, **update** the parameters



UCB1



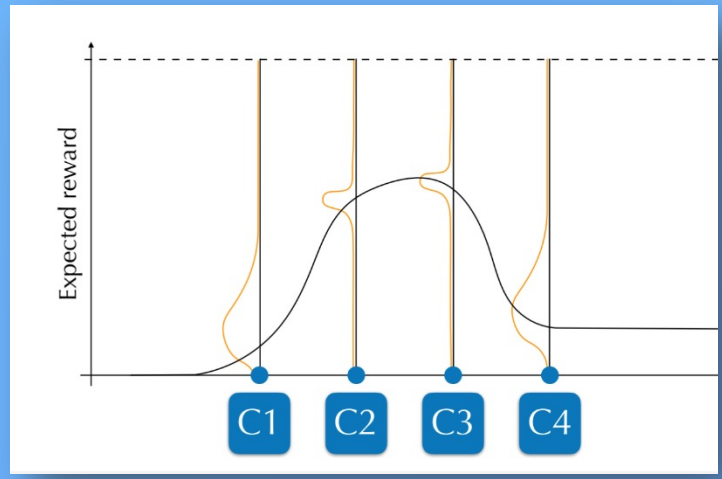
THOMPSON SAMPLING



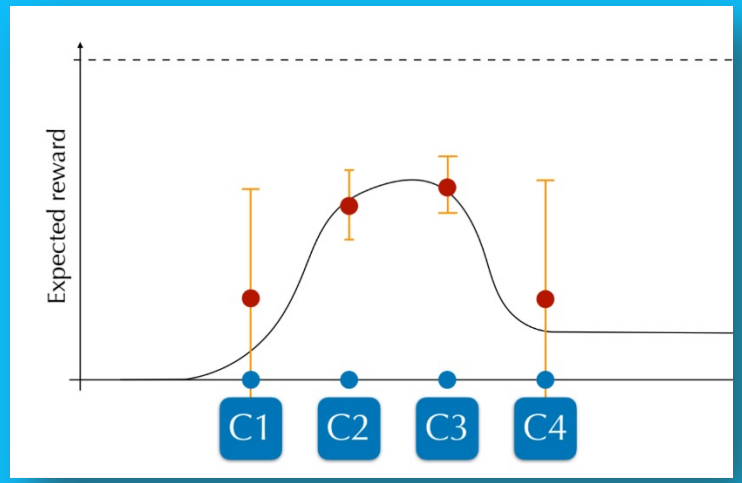
THOMPSON SAMPLING

THOMPSON SAMPLING ALGORITHM

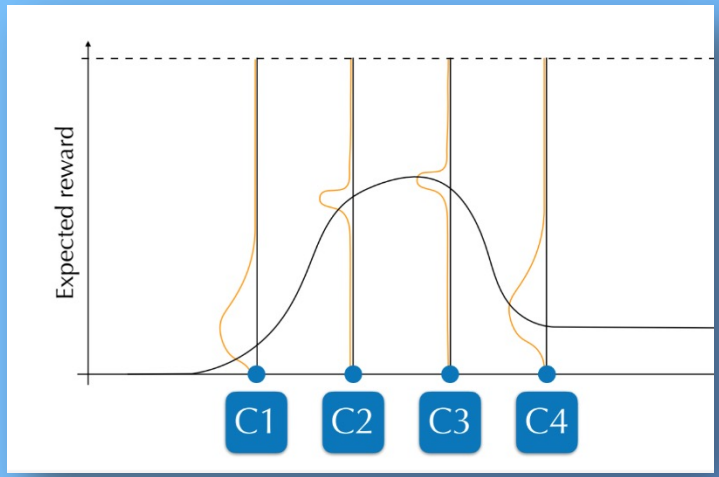
- Associate a **beta distribution** to each price
- At each step **sample** the beta distribution to choose the best arm
- After each round, **update** the parameters



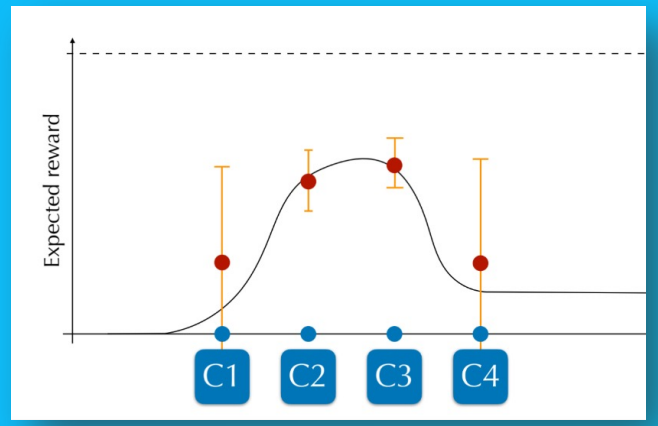
UCB1



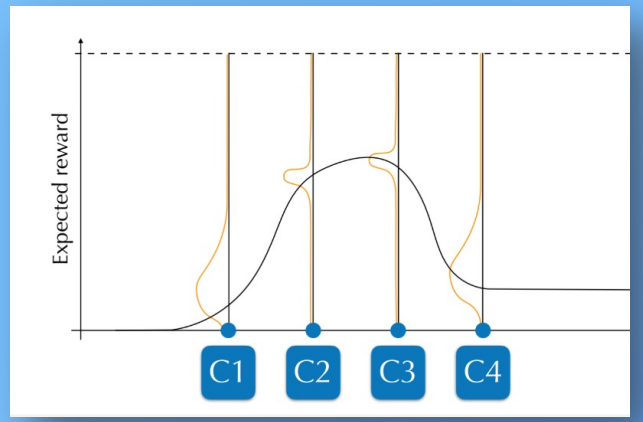
THOMPSON SAMPLING



UCB1



THOMPSON SAMPLING



PSEUDOCODE

UCB1

For the first **10 rounds** we select each arm one by one and then we compute the **upper bound** relative to each arm in this way:

(empirical mean + confidence) * (number of returns +1) * margin.

After that we select the arm corresponding to the **maximum upper bound.**

THOMPSON SAMPLING

We extract the samples from the **beta distributions** of each arm and then we compute the rewards multiplying the values obtained to the number of returns +1 and to the margins of the prices.

We select the arm corresponding to the **maximum reward.**

PSEUDOCODE

```
For each day:
    Pull an arm
```

We create **two dictionaries ts_daily** and **ucb_daily**, which **collect the information** for the two learners about the rewards, the successes, the returns and the accesses of each day

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
```

We create **two dictionaries ts_daily** and **ucb_daily**, which **collect the information** for the two learners about the rewards, the successes, the returns and the accesses of each day

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
```

For every learner, given the number of **daily accesses**, we compute the following quantities:

- For the successes we draw a sample from a **binomial**
- For the returns we draw from a **poisson distribution**
- For the rewards we multiply the **marginal profit** relative to the arm chosen to the **sum of the successes** and the **returns**.

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
```

For every learner, given the number of **daily accesses**, we compute the following quantities:

- For the successes we draw a sample from a **binomial**
- For the returns we draw from a **poisson distribution**
- For the rewards we multiply the **marginal profit** relative to the arm chosen to the **sum of the successes** and the **returns**.

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
```

UCB1

We modify the **empirical means** of the pulled arm with the information on the daily accesses and successes.

Then we update the confidence of the interval using the formula $\sqrt{\frac{2\log(t)}{n_a(t-1)}}$ where $n_a(t-1)$ is the number of times we choose the arm **a** from **T = 1 to T = t-1**

THOMPSON SAMPLING

We update the **beta parameters** with the information collected on the accesses and the successes
(α = α + successes,
β = β + accesses – successes)

We then update the aggregate daily reward, successes and returns

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
```

UCB1

We modify the **empirical means** of the pulled arm with the information on the daily accesses and successes.

Then we update the confidence of the interval using the formula $\sqrt{\frac{2\log(t)}{n_a(t-1)}}$ where $n_a(t-1)$ is the number of times we choose the arm **a** from **T = 1 to T = t-1**

THOMPSON SAMPLING

We update the **beta parameters** with the information collected on the accesses and the successes
(α = α + successes,
β = β + accesses – successes)

We then update the aggregate daily reward, successes and returns

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
```

For each learner we create a **new dictionary** containing the pulled arm, the average number of daily returns (returns/successes) and the number of daily successes.

Then we append them into **two deques** having **30 elements**, each of them initialized at zero. We pop the first dictionary of the deque from the left and we pass it to the function that update the parameter of the poisson distribution that models the returns.

In this way we have that **only at day 30** we use the information collected at **day 1**.

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
    Manage the delay of the
    returns
```

For each learner we create a **new dictionary** containing the pulled arm, the average number of daily returns (returns/successes) and the number of daily successes.

Then we append them into **two deques** having **30 elements**, each of them initialized at zero. We pop the first dictionary of the deque from the left and we pass it to the function that update the parameter of the poisson distribution that models the returns.

In this way we have that **only at day 30** we use the information collected at **day 1**.

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
    Manage the delay of the
    returns
```

For every learner we append in a list the value of the daily reward. This will help us to compute the final regret of the two learners to compare them.

Then we have run the simulation. Every day from each of the two learners we have collected the **arms (prices) pulled and the rewards**. Then we have run **50 simulations** and we have plot the average of the regret obtained in each simulation.

PSEUDOCODE

```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
    Manage the delay of the
    returns
    Save daily reward
```

For every learner we append in a list the value of the daily reward. This will help us to compute the final regret of the two learners to compare them.

Then we have run the simulation. Every day from each of the two learners we have collected the **arms (prices) pulled and the rewards**. Then we have run **50 simulations** and we have plot the average of the regret obtained in each simulation.

PSEUDOCODE

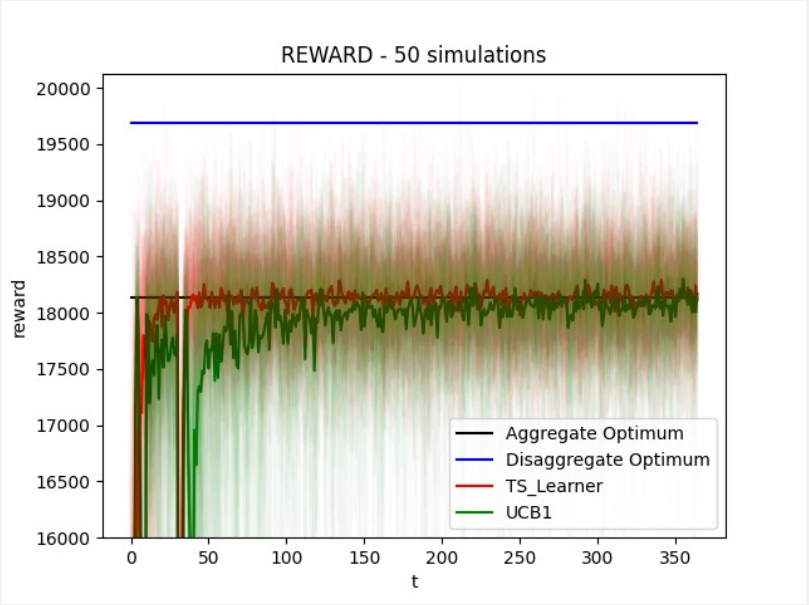
```
For each day:
    Pull an arm
    Initialize rewards,
    successes, returns and
    accesses
    For each class:
        Compute daily
        reward, successes
        and returns
    Update
    Manage the delay of the
    returns
    Save daily reward
```

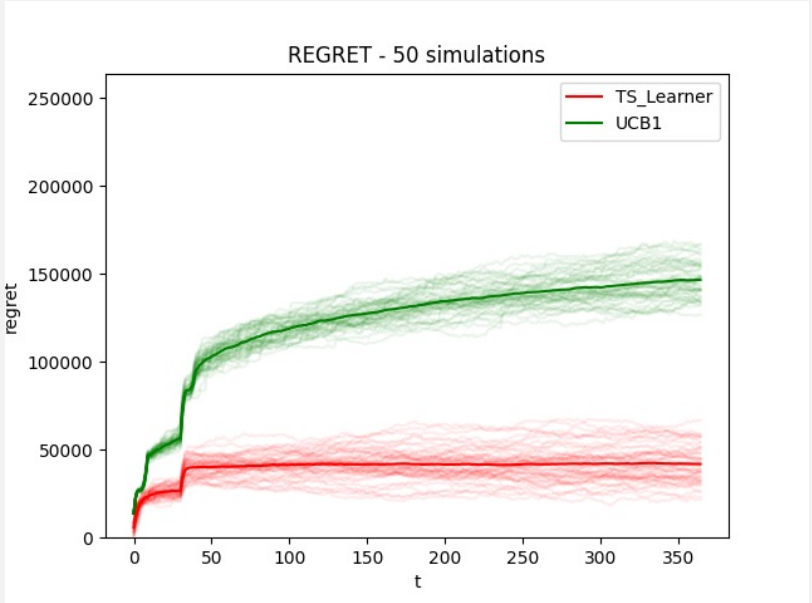
Both the approaches reached the **optimal price of 9€** with a **reward of 18140,28€,** much lower than the disaggregated optimum.

Another thing that we can notice in both methods is that at **day 30** there is a **drop** in the reward due to the new **information about returns** that we start receiving only after this period of time.

We can see that the **TS Learner** is much faster in reaching the optimum and, moreover, it is affected by less noise with respect to **UCB1**.

Looking at the **regret plot** we can also see that, while **UCB1** continues growing logarithmically, the **TS learner** seems to stabilizer after a very short period of time.





Pricing Problem with fixed bid and context generation

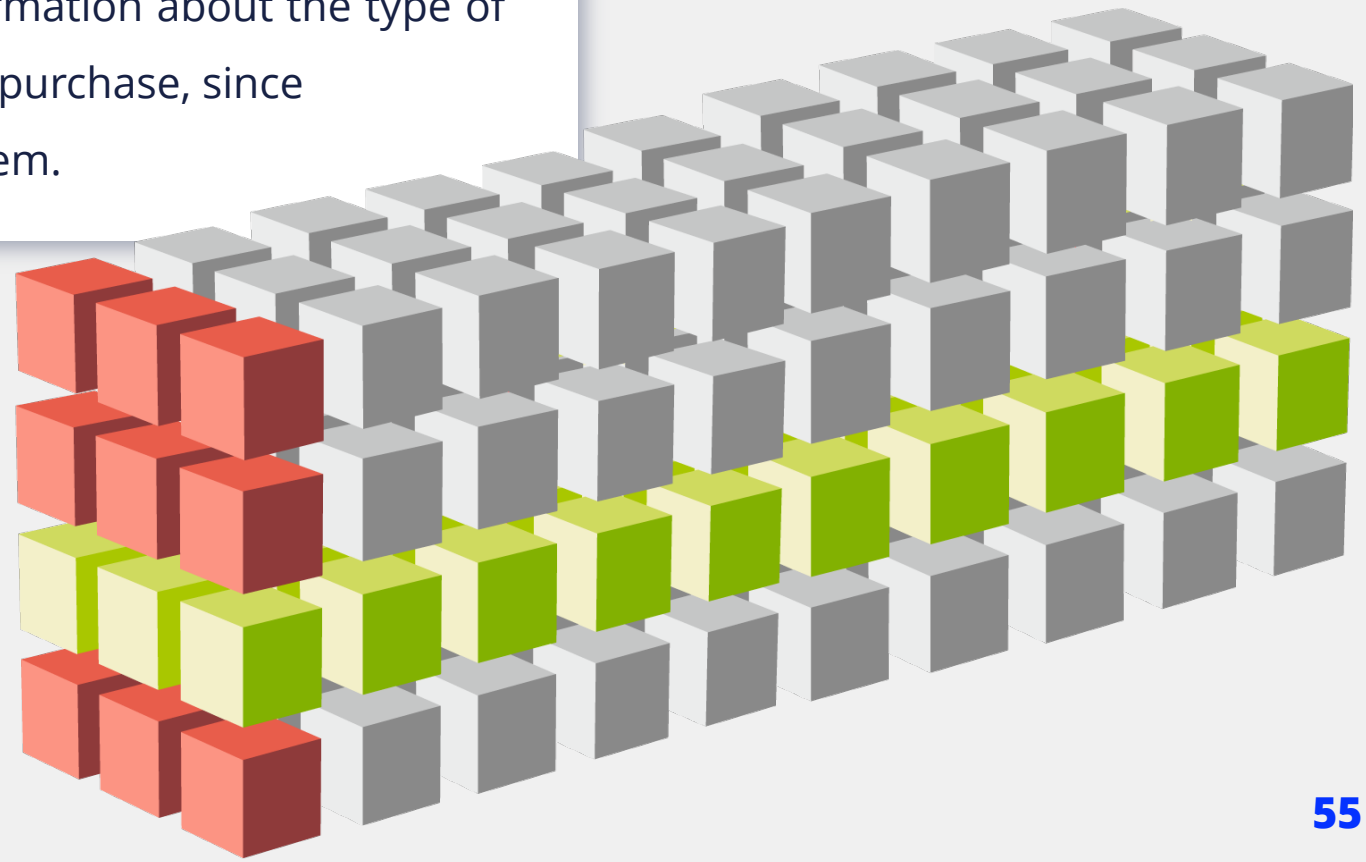
Only TS approach

STEP 4 – CONTEXT GENERATION PROBLEM

We decided to proceed only with the **Thompson Sampling approach** since it provided better results both in terms of final regret and speed of convergence.

Up to this point we never considered the information about the type of users accessing the website or completing the purchase, since they were not relevant for the aggregate problem.

We developed a new data structure **("SplitTable")** to collect the information we needed for the disaggregated approach. In particular, for **each arm** and **each class** we stored the **number of clicks made, the number of purchases and the number of returns after 30 days**. This table is then **updated every day** with the new incoming data.



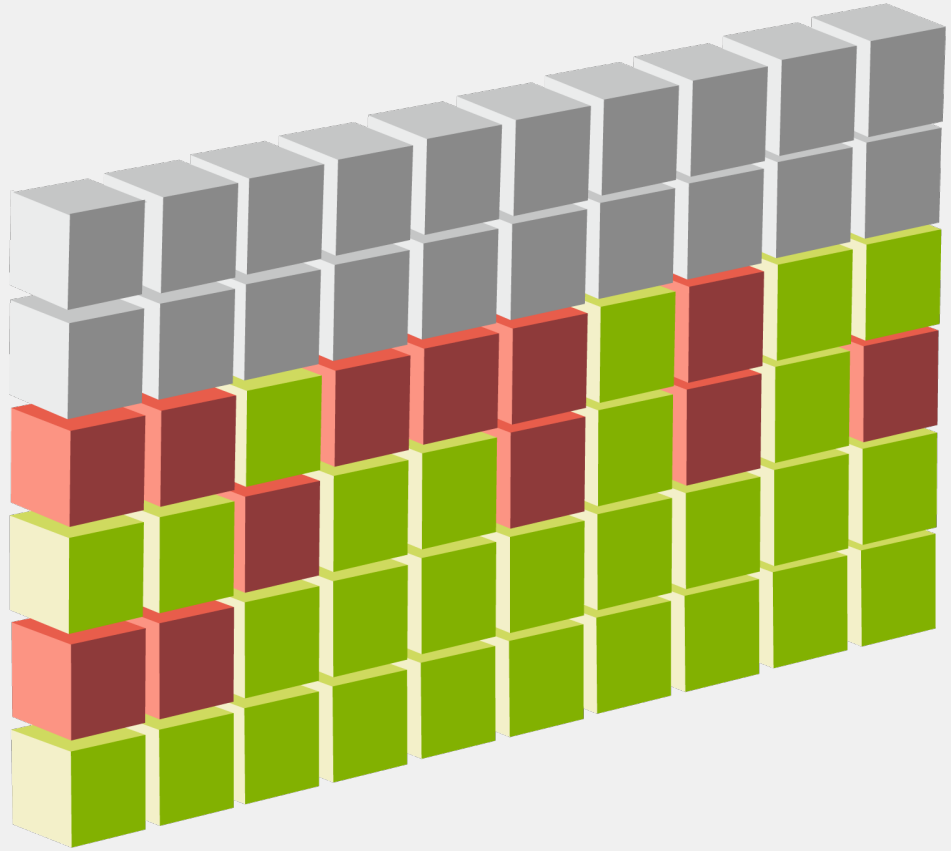
STEP 4 – CONTEXT GENERATION PROBLEM

The **algorithm** is **analogous to the one of point 3**, until we reach convergence.

Convergence criterion: the same arm has to be chosen at least **70%** of the times **over a time-span of the last 40 days** for a year-long experiment.

Convergence is checked **every day:**
if the condition is not satisfied we just complete the day, otherwise the **learner evaluates all available splits**

For the first split the comparison is made between three options: **aggregate**, split based on **age**, split based on **sportive attitude**.



STEP 4 – CONTEXT GENERATION PROBLEM

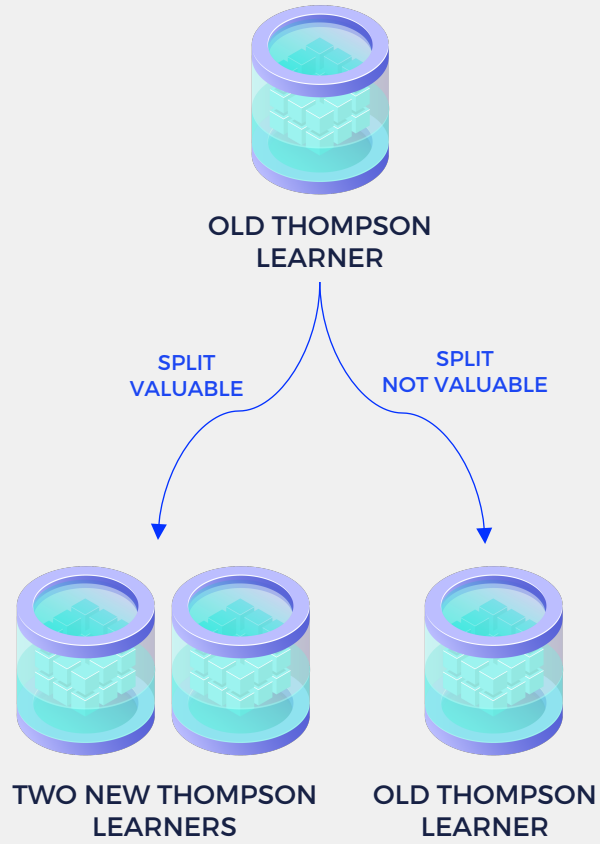
For every split we decide if it is **valuable** or not by comparing it with the aggregate case, according to a **lower bound** on the expected rewards. We used a **confidence term inspired by Hoeffding bound**, multiplied by a coefficient to adapt it to our specific problem.

$$\bar{x} - 50 * \sqrt{-\frac{\log(\delta)}{2|Z|}}$$

If a split is **deemed valuable** we proceed with the creation of two new Thompson Learner, each one initialized with the information of the corresponding group.

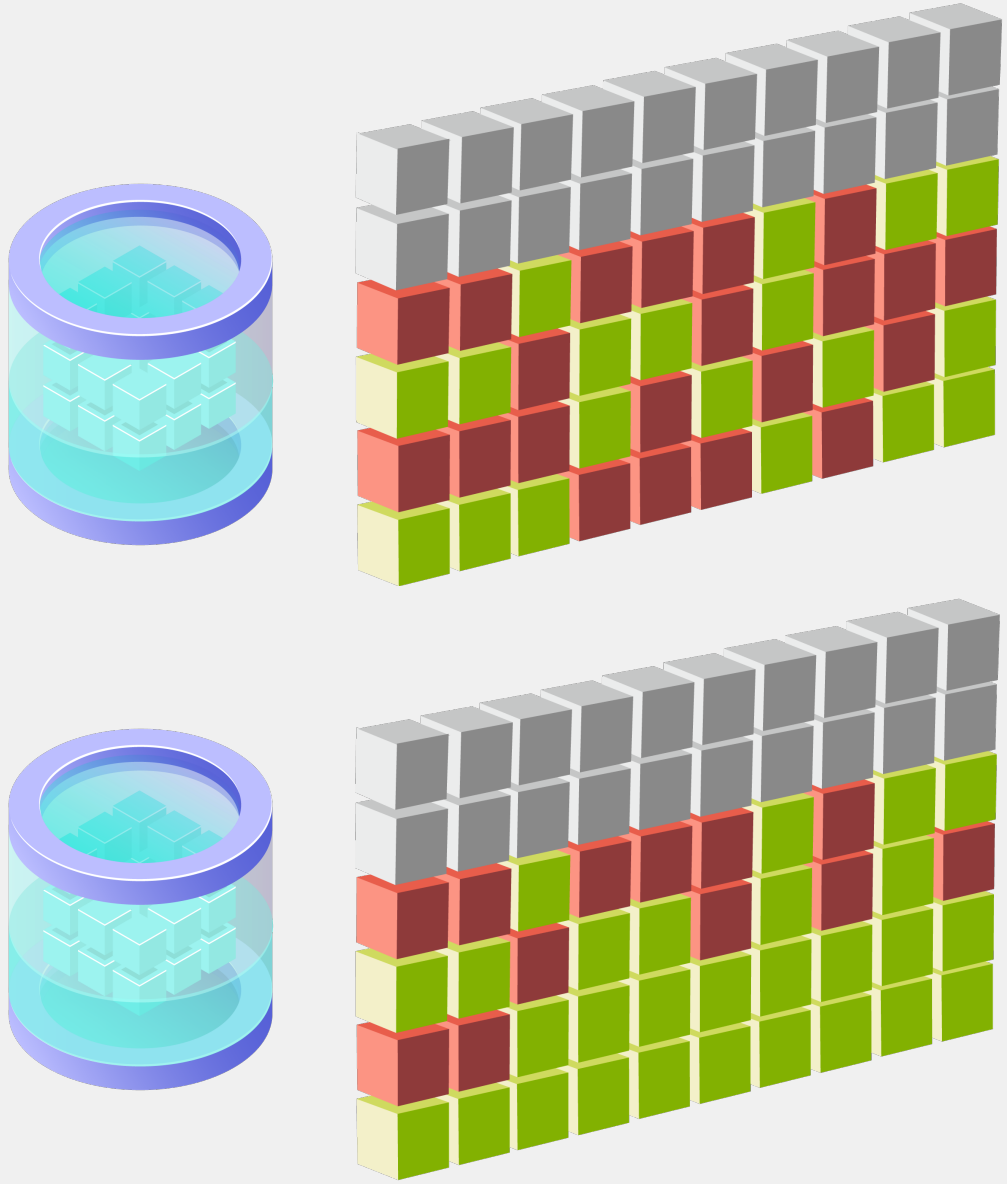
Otherwise, if a split is **not deemed valuable** we simply return our original Thompson Learner and we proceed with another day.

When a splitting is performed we continue our experiment using only the two new Thompson Learner, that are then updated every day with the new incoming data.



STEP 4 – CONTEXT GENERATION PROBLEM

When one of the two Learner reaches a new **point of convergence** we start again **evaluating** every day the **possible splits** on that specific Learner.



If another split is deemed valuable we **repeat the process as before.**

EVALUATE SPLIT

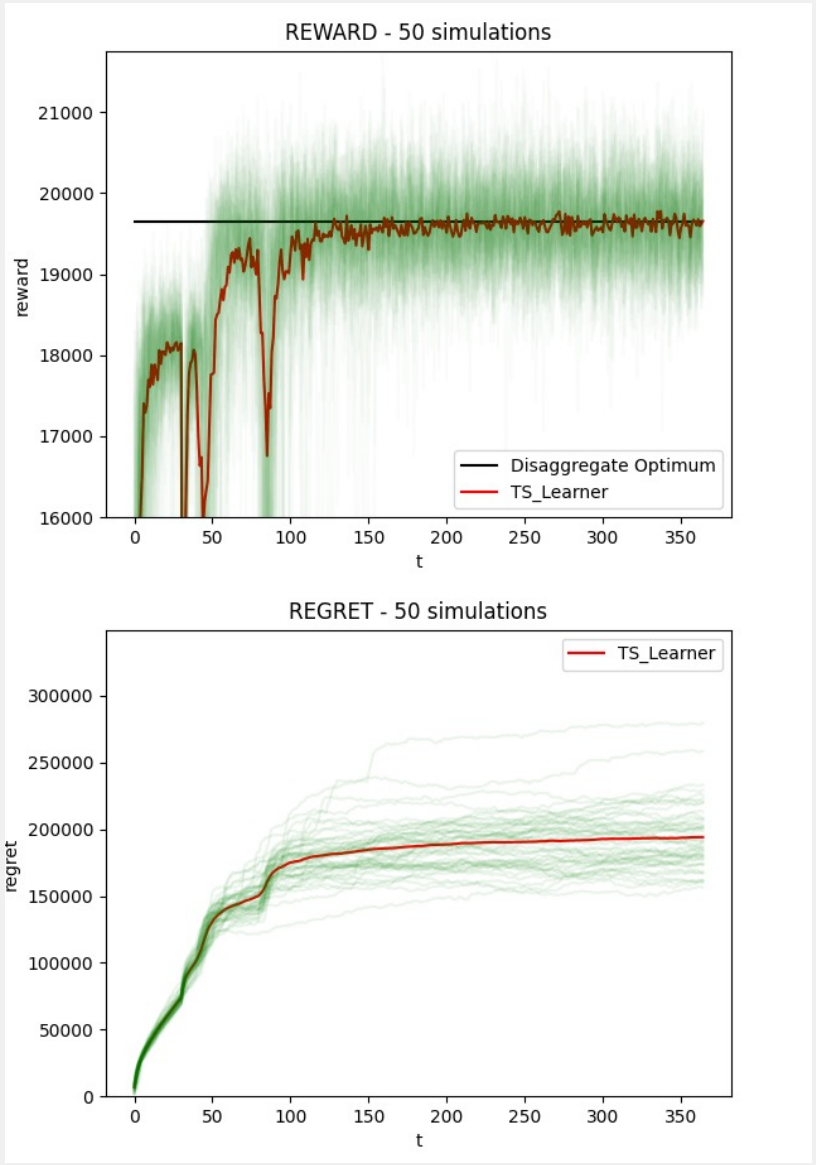
STEP 4 – CONTEXT GENERATION PROBLEM

RESULTS

We can see from our **aggregate reward plot** that the **first split** happens almost immediately after the convergence criterion starts to be checked (around 40/50 days into the experiment).

Then we can see a **small drop** in our reward because the **two new Thompson Learner are initialized** with the information of the aggregated Thompson Learner and therefore, they need some days to recalibrate and to find a new optimal arm for the underlying group.

As we can see from the graph **the increment** in our reward after the first split is quite important, as it passes **from approximately 18000 to over 19000**. This great leap is one of the main results that justify the implementation of context generation in our algorithm.



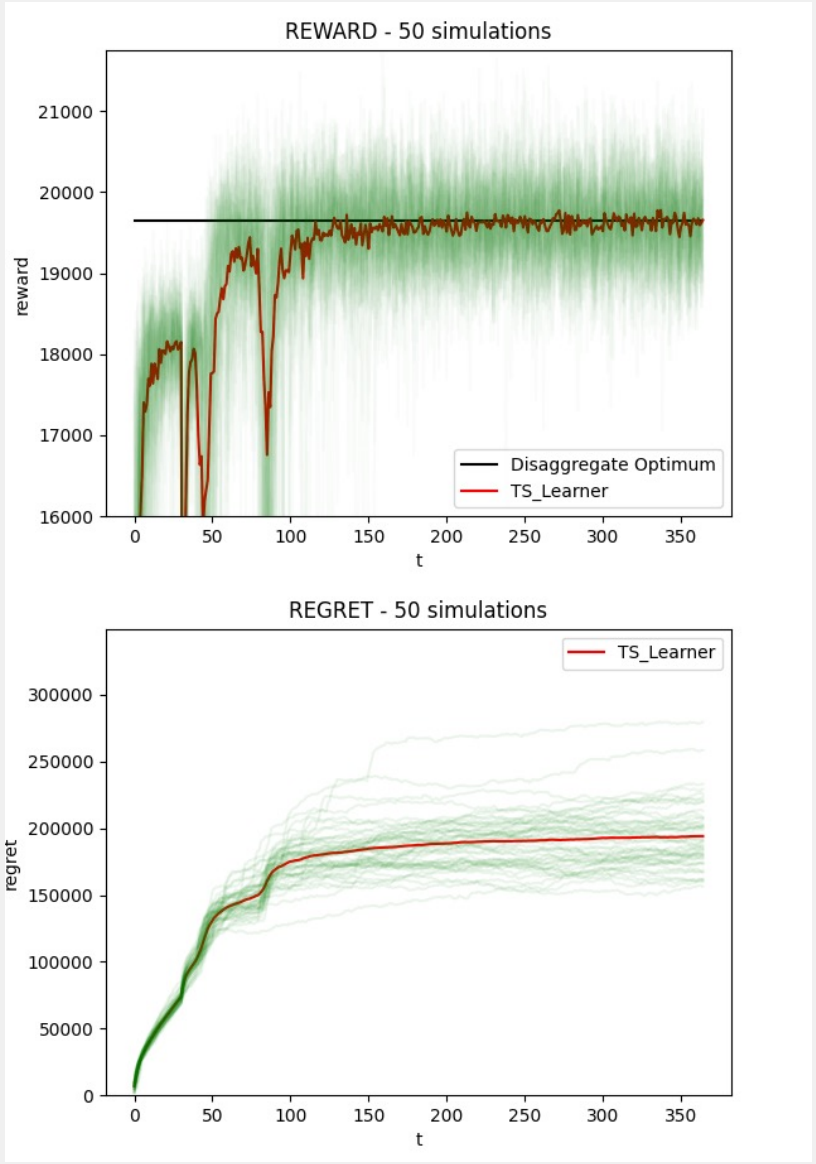
STEP 4 – CONTEXT GENERATION PROBLEM

The **second split** is roughly 40 days after the first one and, even if it brings a **smaller increment** with respect to the first one, it is still a notable improvement for our reward.

As we can see from the plot there is also a **vague green drop around day 150**: this is due to the possibility that **in some cases a third split happens**, that is a split within the category of Sporty People. This is not necessary, since there is no real distinction between young and old people belonging to this class, but it is most probably due to **stochastic effects**.

What is important to notice is that the algorithm reaches a **very good performance** in a rather small amount time and most of the time it would need also less than 40 days, but it's slowed by our convergence criterion.

In the **regret plot** we can highlight a **steep increase** in correspondence of the days where a split is performed or when the information about the returns is made available.



Bidding Problem with fixed price and NO context generation

Only GPTS approach

STEP 5 – AGGREGATED BIDDING PROBLEM

AGGREGATED

In this step we consider the **price fixed (9)** and we optimize only the **bidding problem**. We adopt the following assumptions:

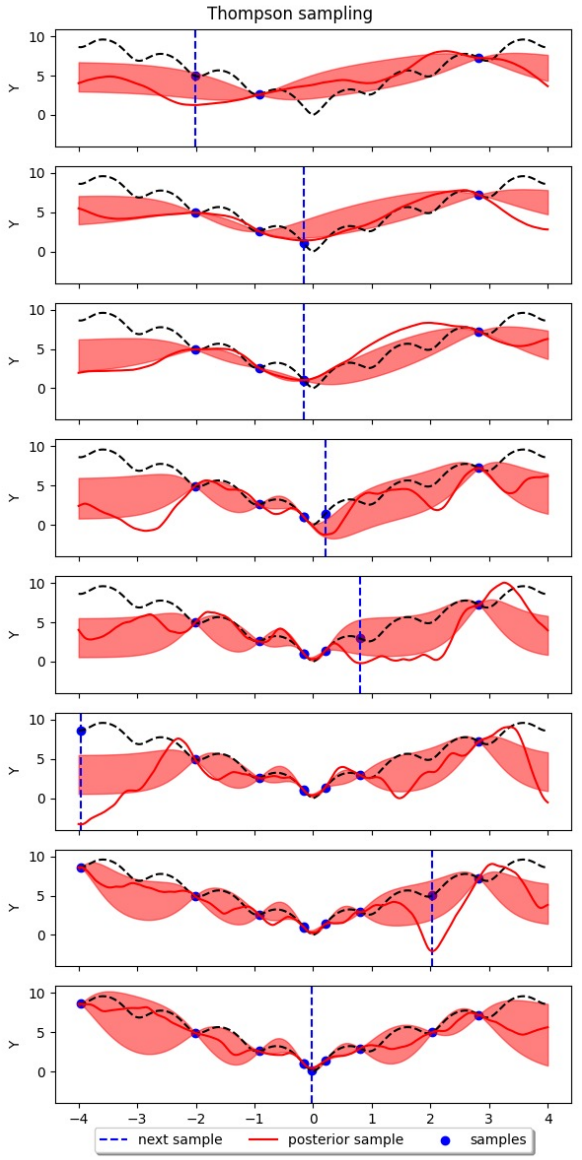
- **Aggregate data**: we do not discriminate between the classes at this stage
- The **conversion rate** is known
- We consider the **number of clicks** and the **cost per click** as **stochastic variables**

In this case the **optimal bid** is **1.2.**

STEP 5 – AGGREGATED BIDDING PROBLEM

In place of the classical Thompson Sampling seen before where each arm is modeled through a Beta, this time we used a **Gaussian Process** approach. In this way we can also take into account the **correlation** between the different arms by introducing a **kernel function** which provides the **covariance** between two inputs.

This gives us a measure of the uncertainty of the estimation and allow us to speed up our convergence by using the neighbour arms information.



LEARNER

Our bidding learner tries to find the bid providing the best expected reward, by **estimating** both **the number of accesses and the cost per click** related to the bids chosen in previous days.

In particular, during the **first 10 days each arm is pulled once**, to avoid situations in which an arm is never played for the whole experiment.

After the day has been simulated, we pass the information about the true accesses and costs to the **learner**, which then **updates his belief using the two gaussian processes**. Theoretically, the amount of uncertainty on the best arms will be reduced over time, as they get chosen more and more.

GPTS for ACCESSES

GPTS for COST PER CLICK

LEARNER

We introduce a **safety check**, in order to avoid playing arms with significant probabilities of obtain a negative result.
We assumed **a normal distribution over the expected revenue**, with standard deviation of 2000. These parameters have been finetuned after performing different attempts.
We chose to discard each arm with an expected revenue smaller than 2000, thus keeping, in the worst case, only arms providing negative reward with a probability smaller than 15.86%.
The safety check works only after **20 days** from the start of the experiment, ensuring that every arm is played at least once.

The **pseudocode** for this approach is **very similar** to the one already presented for the classical Thompson Sampling:

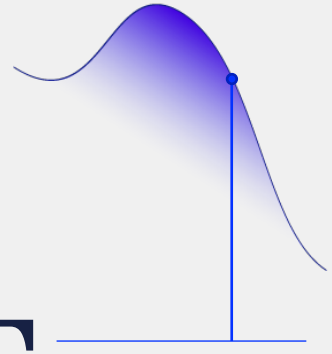
PSEUDOCODE

```
Pull arm
    pulled_arm = gpts_learner.pull_arm()
Compute reward (using the function round)
    reward = 0
            for i, c in enumerate(classes):
                reward += env.round(pulled_arm, i)
Update learner
    gpts_learner.update(pulled_arm, reward)
```

BIDDING ENVIRONMENT:

In every round we **sample** both the number of accesses and the cost per click from a **gaussian distribution**, respectively with parameter $\mathbf{n_i(b) + N(0, 50^2)}$ and $\mathbf{c_i(b) + N(0, 0.01^2)}$ where $\mathbf{n_i}$ and $\mathbf{c_i}$ are the mean for every class that has been presented in the introduction.

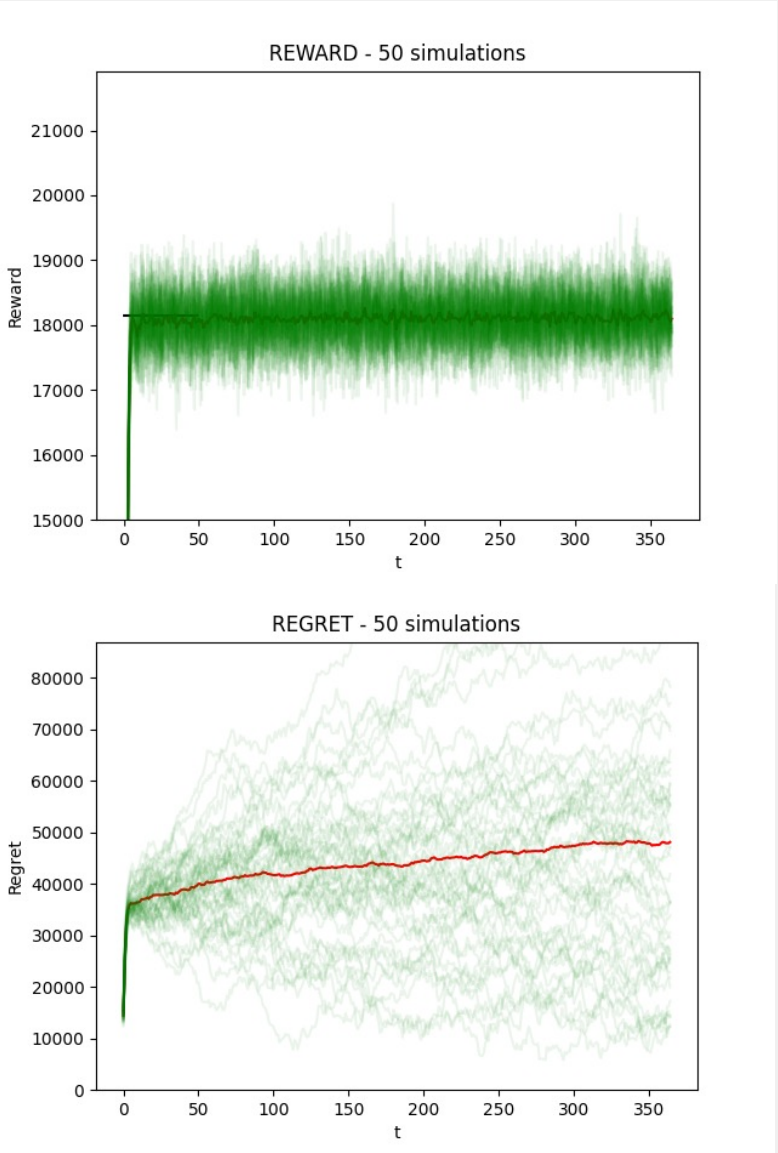
Then we computed the rewards for every class and we summed them.



$$\max_{p,b} \sum_{i=1}^{3} n_i(b)[\, q_i(p) r(p)(N_i + 1) - c(b)]$$

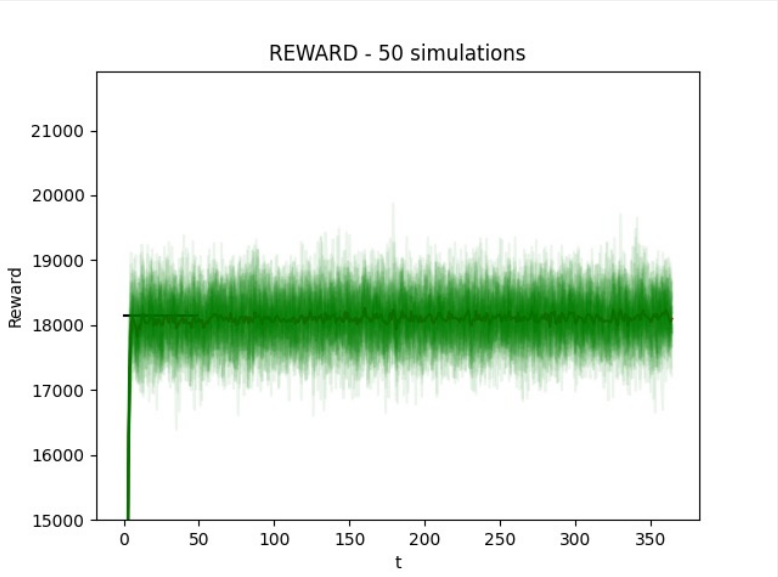
RESULTS

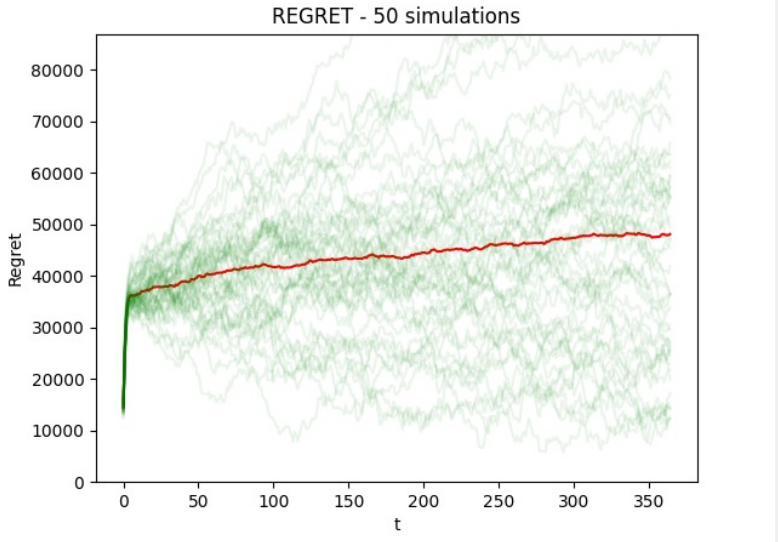
The interplay between the two variances generate a **quite significant cumulative variance** and this is the reason behind the **noise** that is evident in our reward plot. Nonetheless, we can see that the **optimum is reached almost immediately**, and this is a sign of the **good performance** obtained with the Gaussian Process approach.



RESULTS

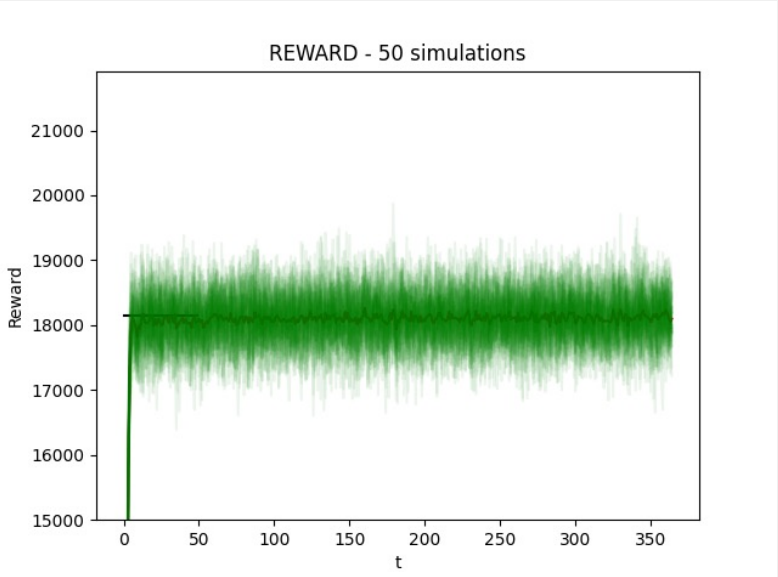
We have also introduced a **threshold** to prevent the choice of arm with negative revenues with a given probability, but this has **not improved the performance** of our algorithm since among the most promising arms, none of them actually provided negative revenues with significant probability.

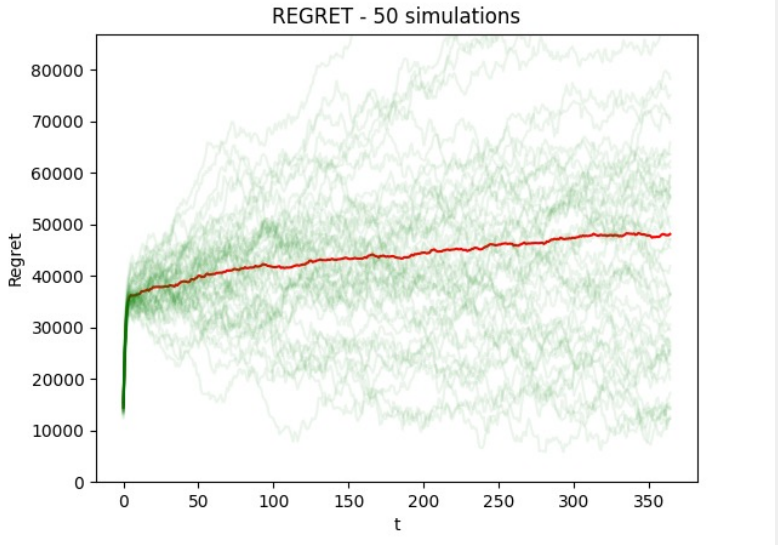




RESULTS

Since the reward function is **not very steep with respect to the bid**, our Gaussian Process Regressor sometimes **fails to identify the right arm** and stabilizes on a suboptimal one. This causes a **little increase in our regret** plot but, generally, these are only isolated cases that do not affect the final result.



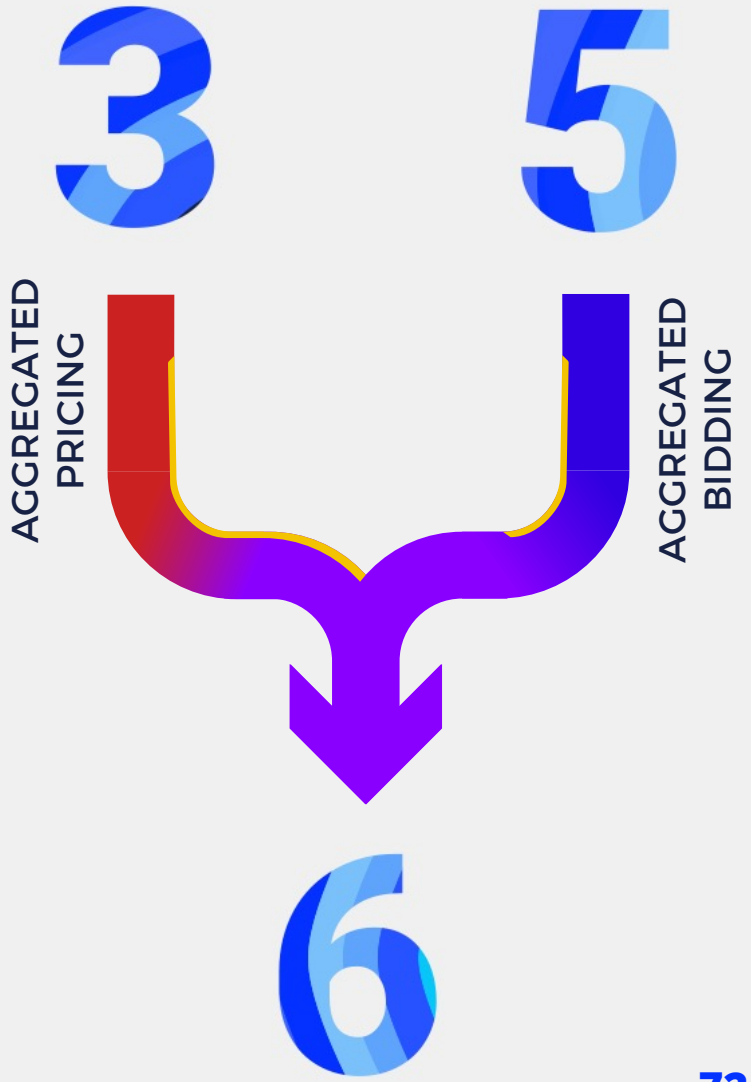


Joint problem with **NO** context generation

STEP 6 – AGGREGATED JOINT PROBLEM

In this step we have to put together **points 3 and 5** and learn the **joint pricing and bidding strategy**.

We assume to be in an aggregated scenario and we consider the number of accesses, successes, returns and cost per click as stochastic variables.



STEP 6 – AGGREGATED JOINT PROBLEM

We create **2 different environments** for bidding and pricing:

BIDDING ENVIRONMENT:

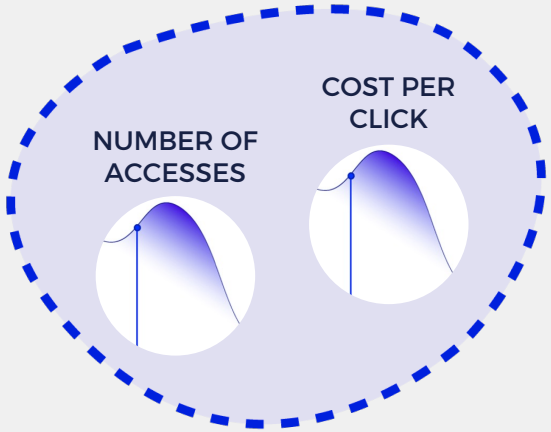
The **number of accesses** and **cost per click** are sampled from a **gaussian distribution** with mean equal to the deterministic values corresponding to the pulled arm and standard deviation **50** for the number of accesses, **0.01** for the cost per click

PRICING ENVIRONMENT:

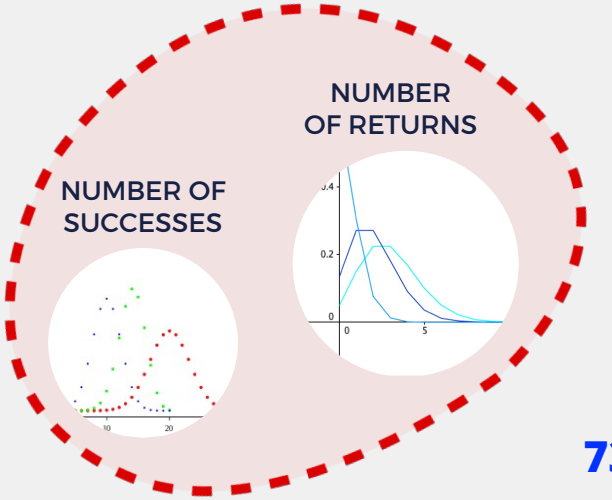
The **number of successes** is sampled from a **binomial distribution** with number of trials equal to the number of accesses and as probabilities the values of the conversion rate.

The **number of returns** is sampled form a **poisson distribution** with parameter the number of successes * the lambda associated to the classes

BIDDING ENVIRONMENT:



PRICING ENVIRONMENT:



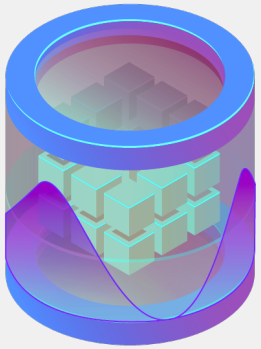
STEP 6 – AGGREGATED JOINT PROBLEM

LEARNER:

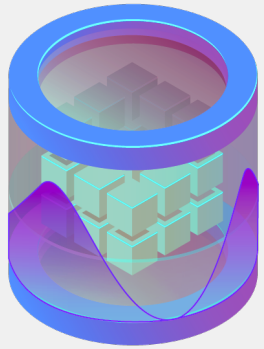
We create a new Learner for the bidding part and as in step 5 we adopt a **Gaussian Process approach**; in particular we use **two different gaussian processes**, one for the **accesses** and one for the **cost per click**. In both cases the parameters (kernel and alpha) have been fine tuned.

This learner includes the same **safety constraint** used in step 5 to avoid playing arms with expected negative revenue.

For the **pricing** part we use the **Thompson Learner already implemented** in the previous steps.



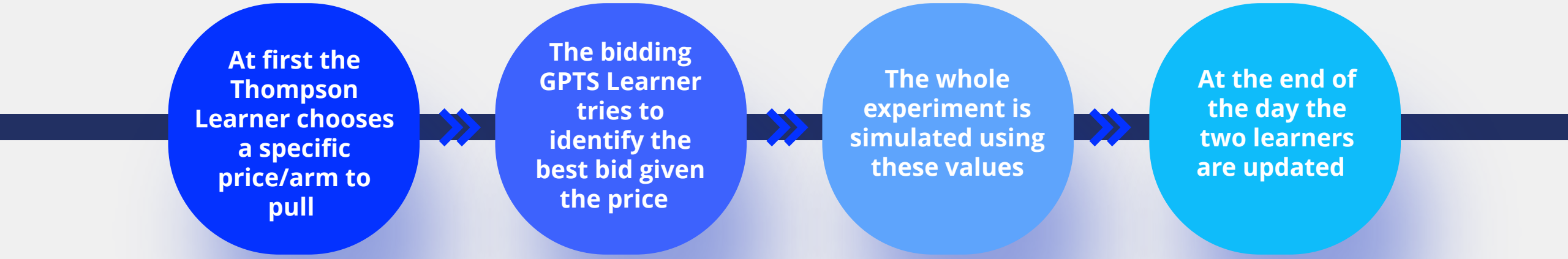
GPTS for ACCESSES



GPTS for COST PER CLICK

PRICING THOMPSON LEARNER

The program works iteratively in this way:

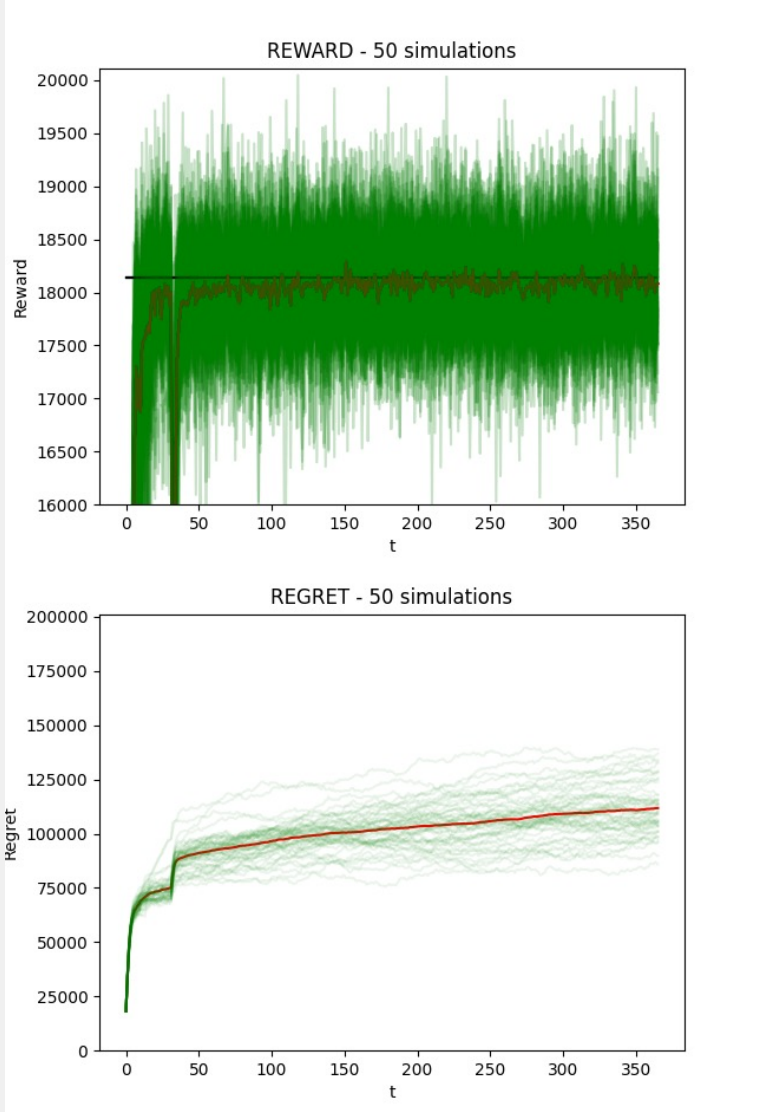


RESULTS

From the reward plot we can still see that there is the **drop at day 30** due to the information about returns that we already justified in point 3.

Other than that, we can observe from both the plots that **convergence is reached in roughly 50 days** in most of the experiments.

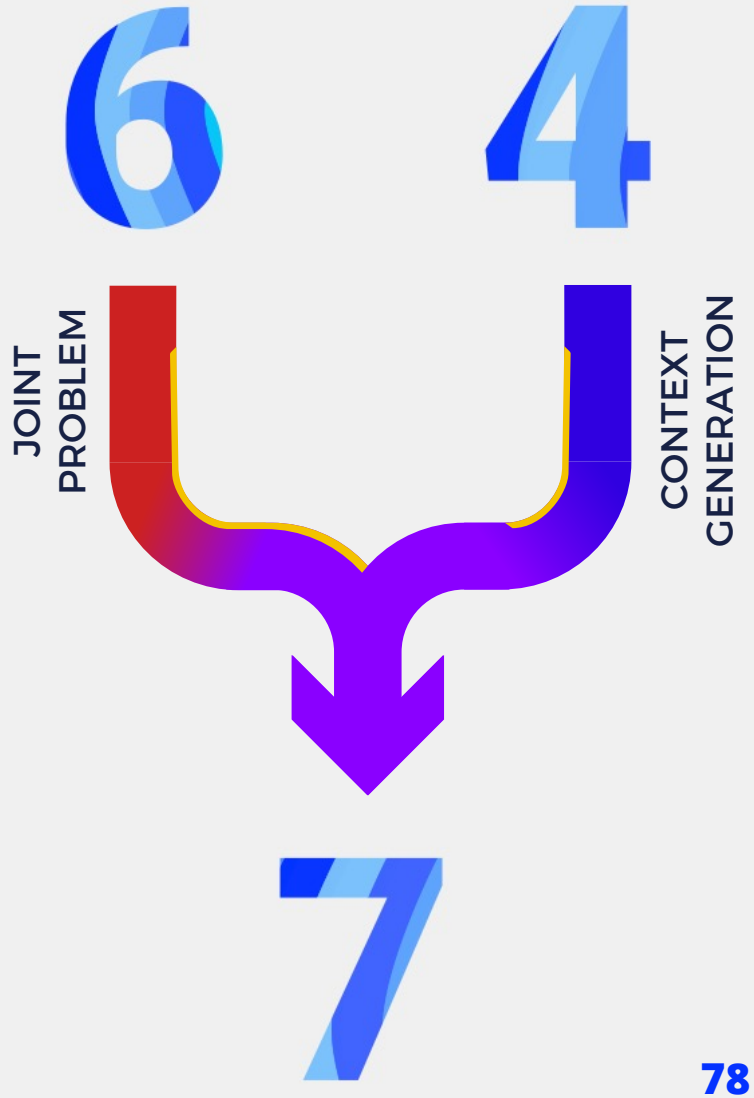
Again we can see that our learner is quite affected by the noise.



Joint problem with context generation

STEP 7 – JOINT PROBLEM WITH CONTEXT GENERATION

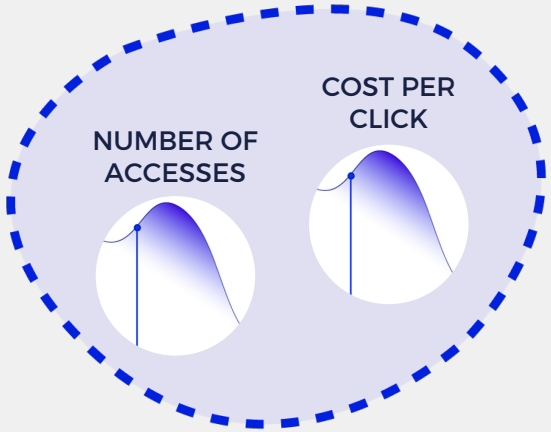
Starting from the problem faced in **point 6** we also integrate the **context structure** we had developed in **step 4**, thus creating the **most realistic version** of our online learning problem.



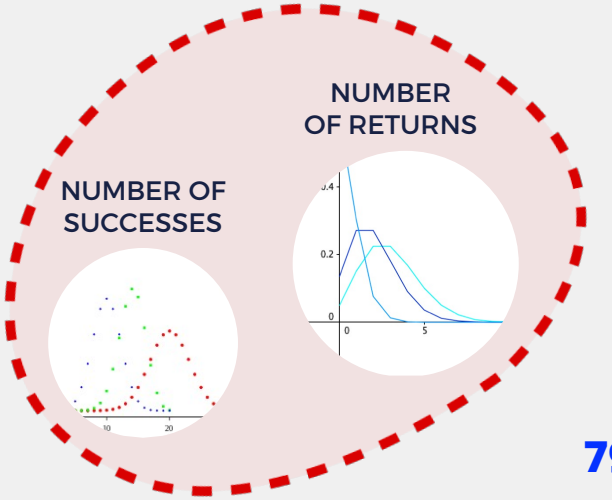
STEP 7 – JOINT PROBLEM WITH CONTEXT GENERATION

The ENVIRONMENT structure is **analogous** to the one of problem 6, with one Environment (the **Bidding** one) dedicated to the **simulation of the number of accesses and cost per click** and the other one (**Pricing**) utilized to **simulate the actual purchases and returns of the users**.

BIDDING ENVIRONMENT:



PRICING ENVIRONMENT:

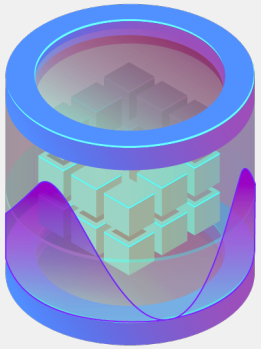


STEP 7 – JOINT PROBLEM WITH CONTEXT GENERATION

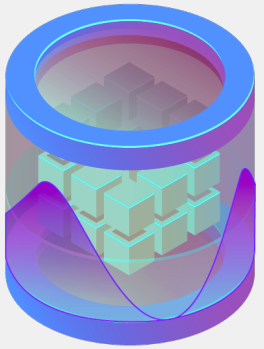
LEARNER:

We use **two** different **type of learners** working at the same time, one for the **pricing** problem (estimate the conversion rate associated to each price using Thompson Sampling techniques) and one for the **bidding** (estimate the number of accesses and cost per click related to each bid using Gaussian Processes).

PRICING THOMPSON LEARNER



GPTS for ACCESSES



GPTS for COST PER CLICK

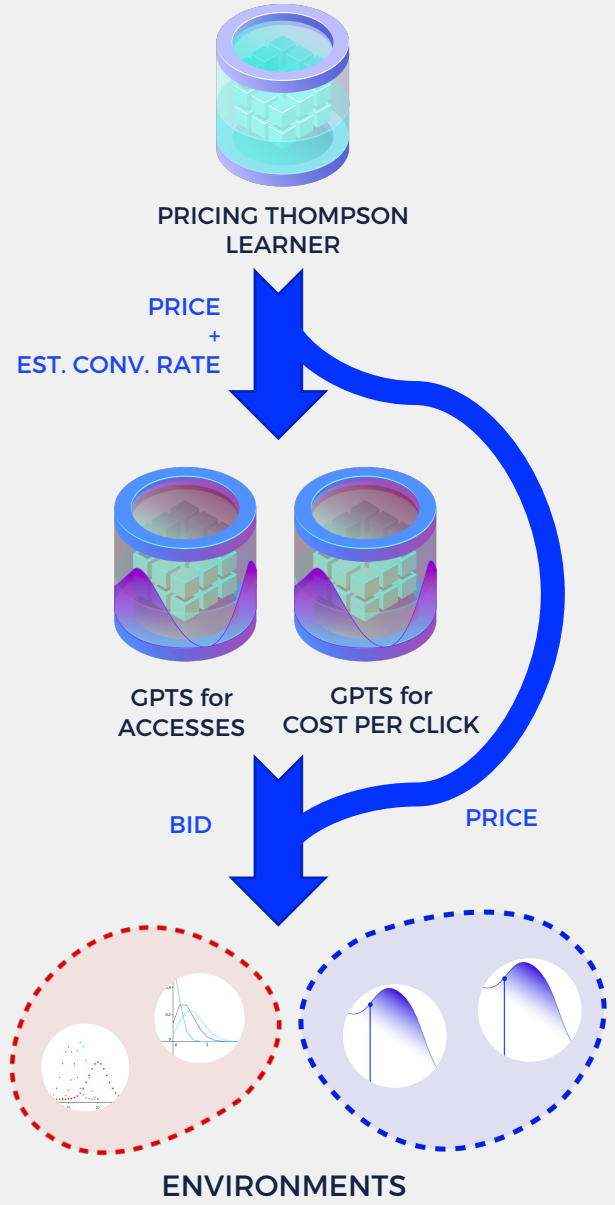
STEP 7 – JOINT PROBLEM WITH CONTEXT GENERATION

We start with the aggregate population, without any prior info about the division in classes. At the beginning of **each day the pricing learner chooses an arm**, following a Thompson Sampling approach.

The **bidding learner** then **receives the information** about the chosen price and the estimate of the associated conversion rate from the pricing learner.

Then, **for every possible bid**, we have an estimate of the **number of accesses** and the **cost per click**, given by the two gaussian processes: thus we can estimate the value of the reward function for every feasible bid. The **GPTS Learner chooses the bid** providing the best expected reward.

The **experiment** is then **simulated using the price and bid proposed**, and we iterate in this way.

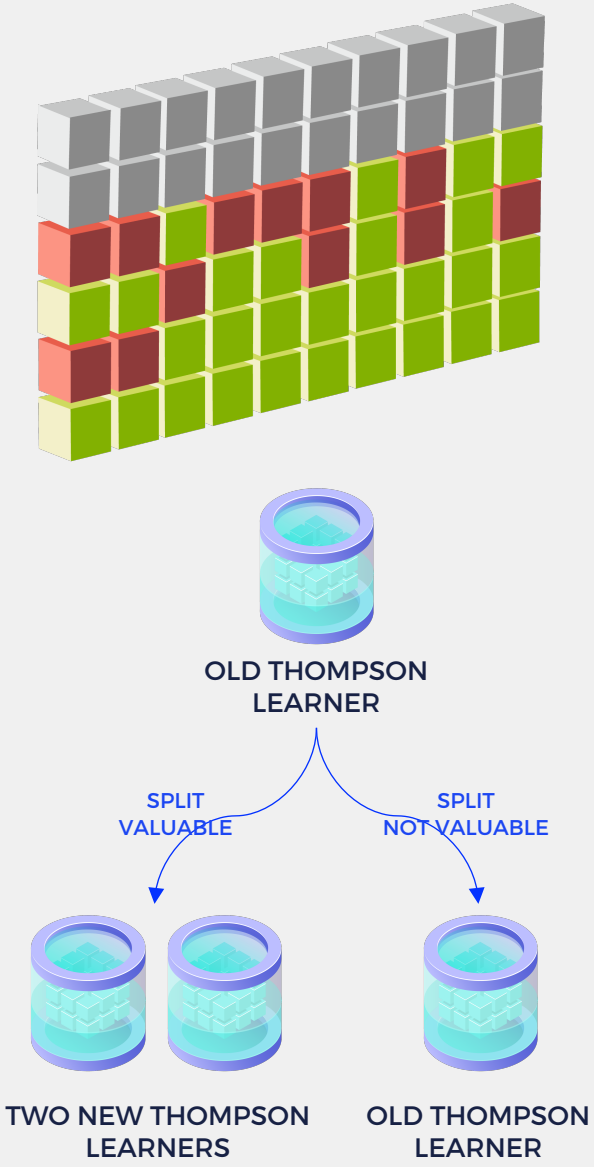


STEP 7 – JOINT PROBLEM WITH CONTEXT GENERATION

Since we are using the **context structure of problem 4** (we are not discriminating over the bid), we just need to check the convergence over the pulled prices in order to be able to evaluate a possible split. We choose to use the same **convergence criteria of previous points** (analyse a 40-day window and one arm repeated at least 70% of the times). **The possible split is then evaluated** with the same algorithm we used in step 4.

If a **split** is deemed **valuable** we **create two new Bidding Environments**, one for each of the new scenarios, and **two new pricing learners**, which inherit information about returns and conversion rates from the original one.

The algorithm proceeds then in the **same way for the new two scenarios**, while the previous one is stopped. The simulations run in parallel, until one split is again deemed valuable: then the procedure is repeated.

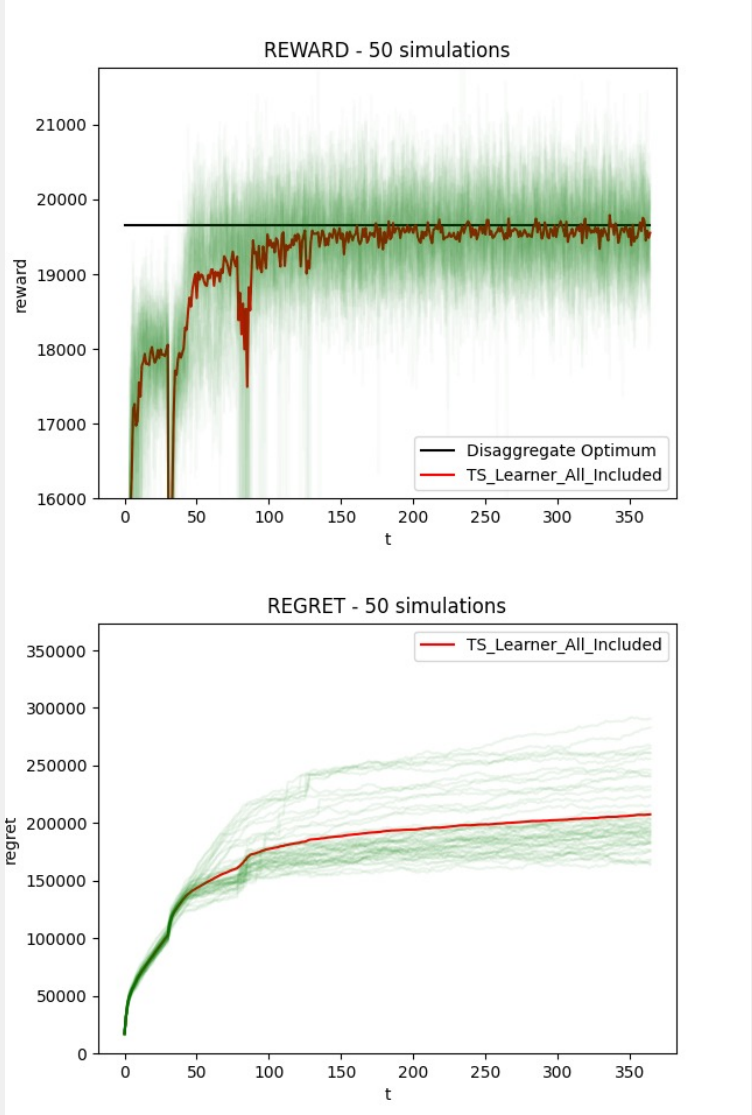


RESULTS

If the algorithm works in the optimal way, it should split the ensemble of users in **three groups**: sporty people (without age differences), non sporty young people and non sporty old people.

The **performance of the algorithm is generally good**: the three splits are achieved almost every time (around 81% of the experiments).

Sometimes the algorithm performs 4 splits, dividing the sporty class in 2: this is superfluous and it is probably due to a slightly excessive sensibility to noise of our Learners.



BLUECOW

L'ENERGY DRINK.

THANKS FOR THE ATTENTION!

Bressan Manuel – Carrara Davide – Tombari Filippo – Zanotti Daniela

Data Intelligence Applications – A.Y. 20/21

POLITECNICO
MILANO 1863