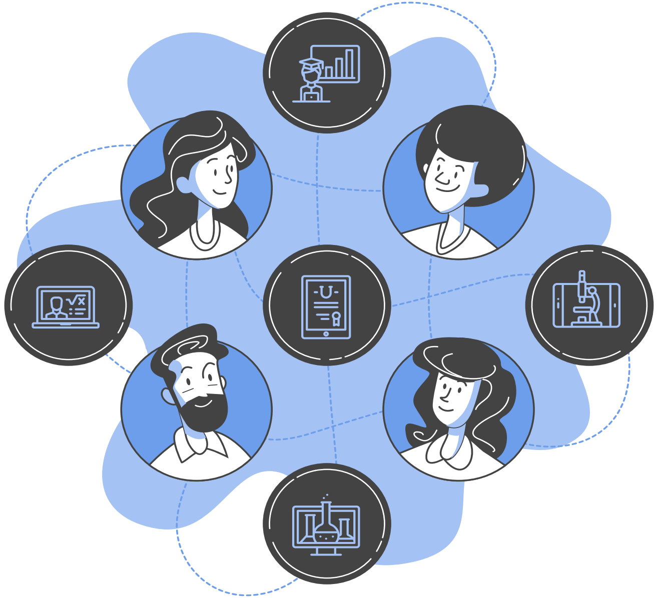
Data Intelligence Application Pricing + Advertising project

Authors

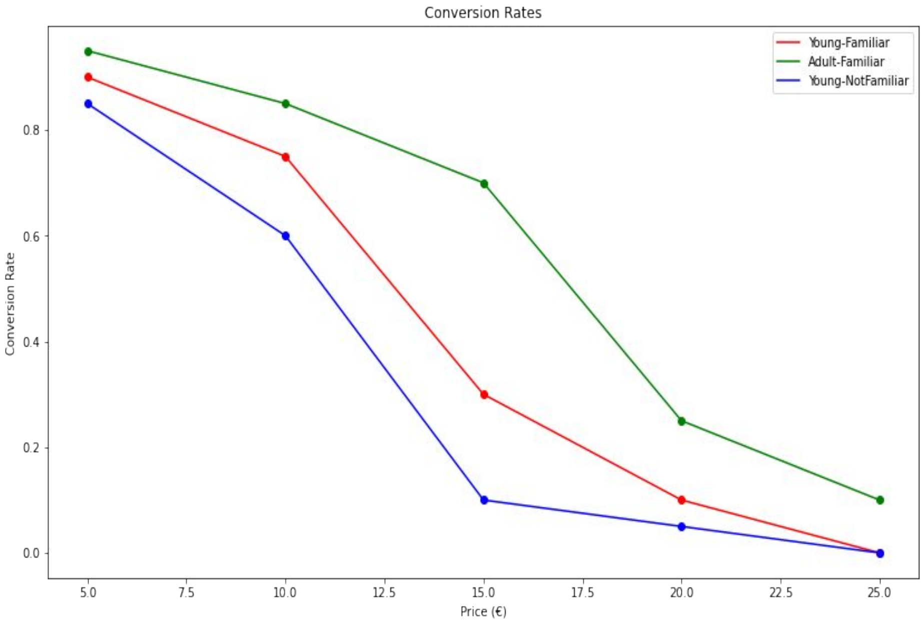
David Baroncini, Stefano De Cillis, Gionatan Del Giudice, Massimo Gennaro, Edoardo Lamonaca

Part 1
Introduction

Introduction

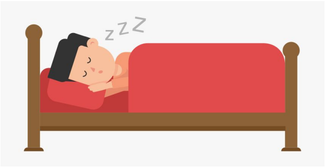


Model and conversion rates



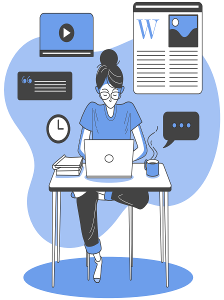
	Familiar	Not Familiar
Young	x	x
Adult	x	

Abrupt phases



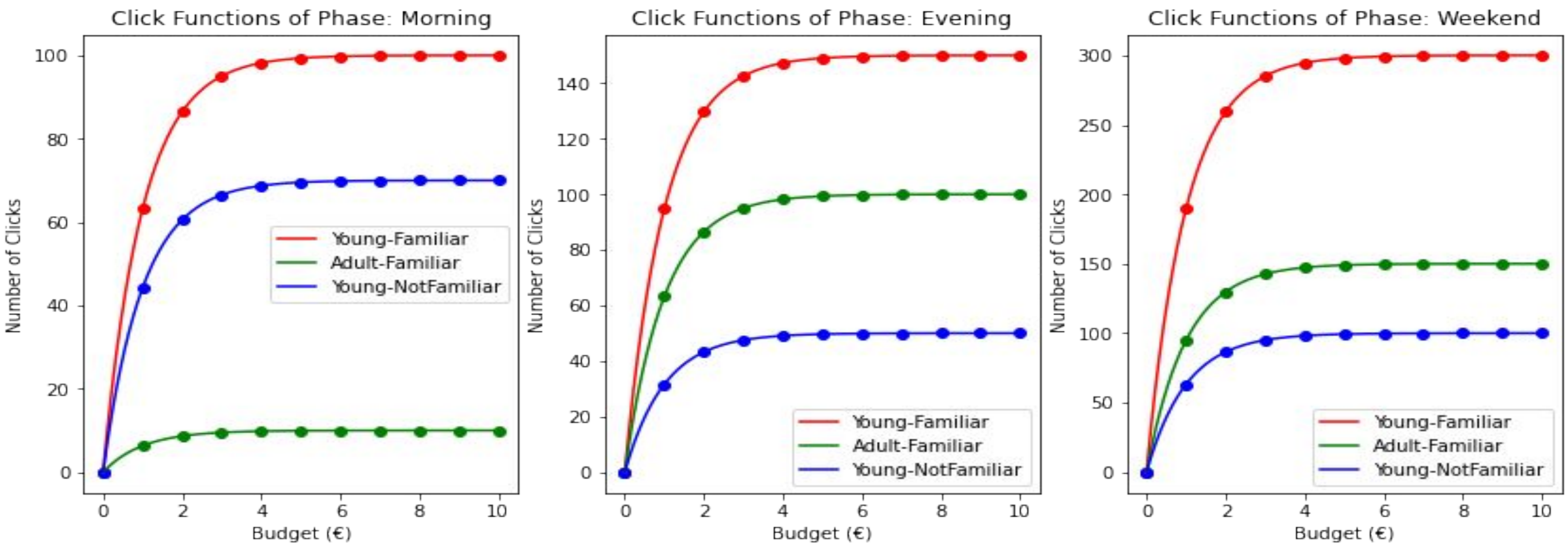
We decided to consider the whole week and to split it into three phases as follows:

- Morning: 00-12 AM during the week (Mon-Fri)
- Evening: 00-12 PM during the week (Mon-Fri)
- Weekend: all day of Saturday and Sunday



	Mon AM	Mon PM	Tue AM	Tue PM	Wed AM	Wed PM	Thu AM	Thu PM	Fri AM	Fri PM	Sat AM	Sat PM	Sun AM	Sun PM
Morning	x		x		x		x		x					
Evening		x		x		x		x		x				
Weekend											x	x	x	x

Click functions



$$click\_function(x|m, s) = m(1 - e^{sx})$$

Timeline

ADVERTISING

stationary experiment

non-stationary experiment

PRICING

w/o context generation

w/ context generation

ADVERTISING + PRICING

different price for every context

unique price

Part 2 and 3
Advertising Campaign

Assumptions

- Pay-per-click advertising
- Finite values of daily budgets
- The bid value is automatically computed by the advertising platform
- Independent sub campaigns' performance
- Unique value-per-click (i.e. $v_j = 1$)
- Goal: optimal cumulative daily budget partition over the sub campaigns

Part 2 - Stationary Experiment

- One cumulative phase
- A time horizon which subdivide a week into 112 time-instants (8 for each half-day)
- A cumulative budget of 10M €
- 3 independent sub campaigns

Knapsack algorithm

- 11 evenly spaced values of budget $y_{j,t} \in [0,10]$
- 3 sub campaigns
- Maximization of the cumulated number of clicks $n_j(x_{j,t})$

n_j	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	Budget
C0	0.00	110.62	151.32	166.29	171.80	173.82	174.57	174.84	174.94	174.98	174.99	4
C1	0.00	51.92	71.03	78.05	80.64	81.59	81.94	82.07	82.12	82.13	82.14	3
C2	0.00	45.15	61.76	67.87	70.12	70.95	71.25	71.36	71.41	71.42	71.43	3
TOT	171.80 + 78.05 + 67.87 = 317.12k clicks											**10M €**

Knapsack execution over the environment's real values (clairvoyant solution)

Combinatorial GP bandits

- We have correlation between among the points belonging to each click-function curves
- A Gaussian Process (GP) - Thompson Sampling (TS) - learner is used for each sub campaign
  - Gaussian Kernel with 0 mean and variance of 10
  - Hyperparameters learnt before the actual experiment
- We can pull at most an arm for each GP
  - Such that the knapsack constraint is satisfied

Combinatorial GP-TS algorithm

At every time $t \in T$

1. For every subcampaign $j \in N$, for every arm $a \in A$:

$$\tilde{n}_{a,j} \leftarrow Sample(\mathbb{P}(\mu_{a,j} = n_{a,j}))$$

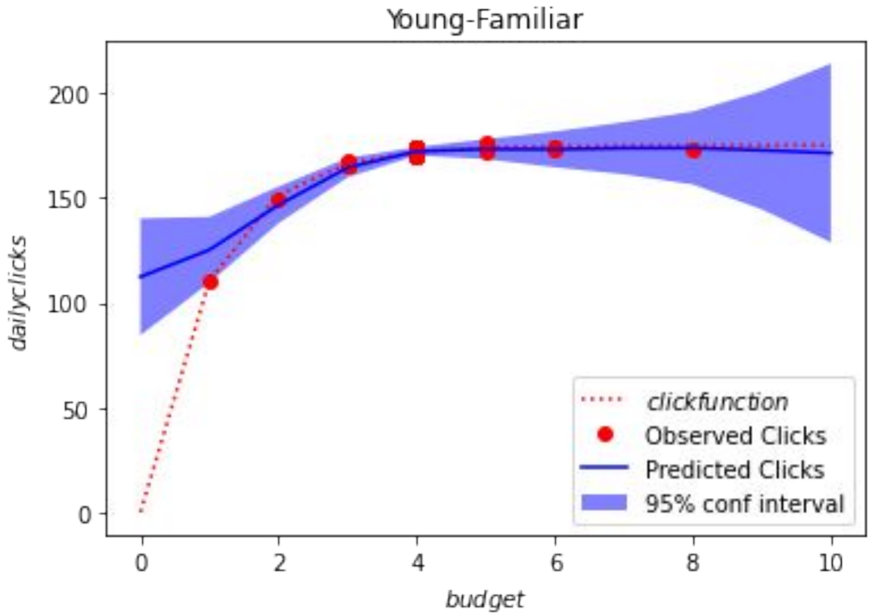
2. Execute the Knapsack algorithm

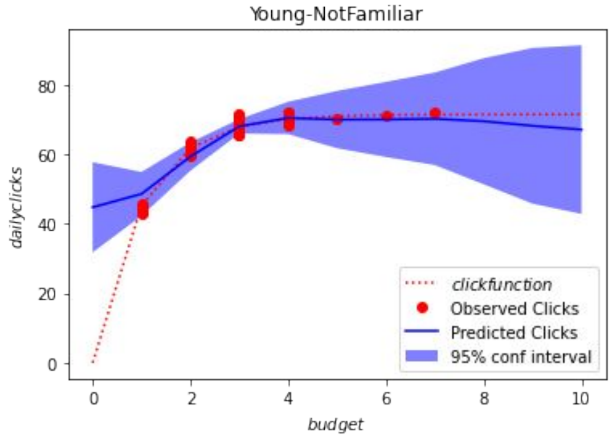
$$\{\hat{y}_j\}_{j \in N} \leftarrow Knapsack(\{(v_j \tilde{n}_{a,j}(y), \bar{y}_j\}_{j \in N})$$

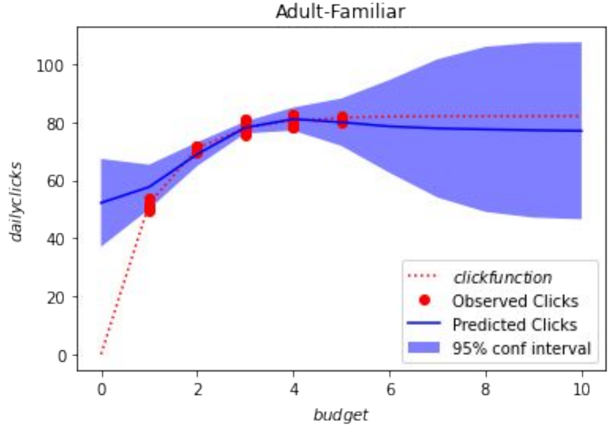
3. For every subcampaign $j \in N$, play arm $\hat{y}_j$

4. Update the GP according to the observed rewards so far

GP-TS learners

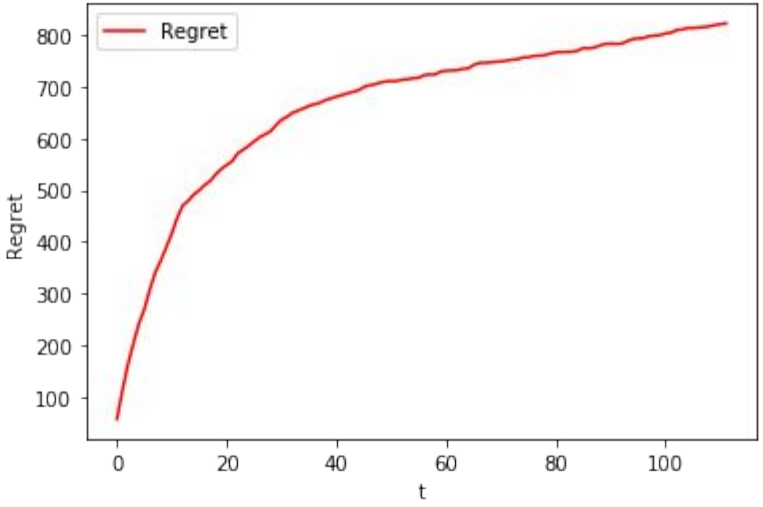


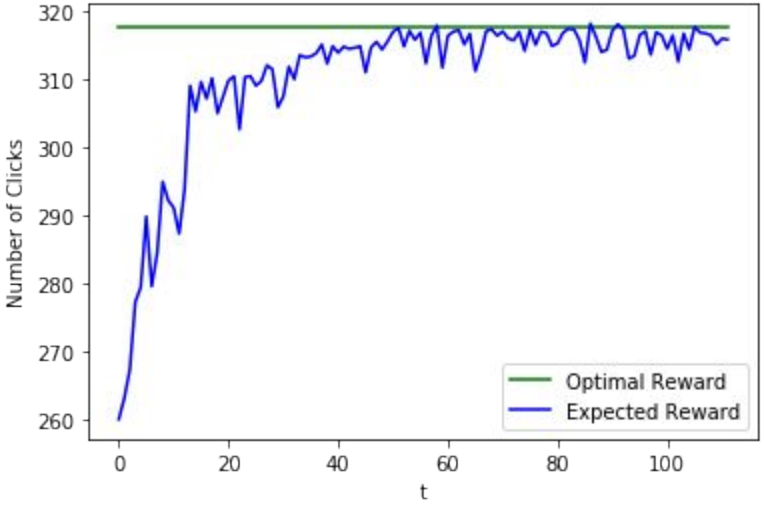




Results

Performing 10 experiments and averaging





Part 3 - Non Stationary Experiment

- 3 different phases (morning, evening, weekend) composing 11 abrupt changes
- A fixed value of sliding window
- A time horizon which subdivide a week into 112 time-instants (8 for each half-phase)
- A cumulative budget of 10M €
- 3 independent sub campaigns

Combinatorial GP-SW-TS algorithm

At every time $t \in T$

1. For every subcampaign $j \in N$, for every arm $a \in A$:

$$\tilde{n}_{a,j} \leftarrow Sample(\mathbb{P}(\mu_{a,j} = n_{a,j}))$$

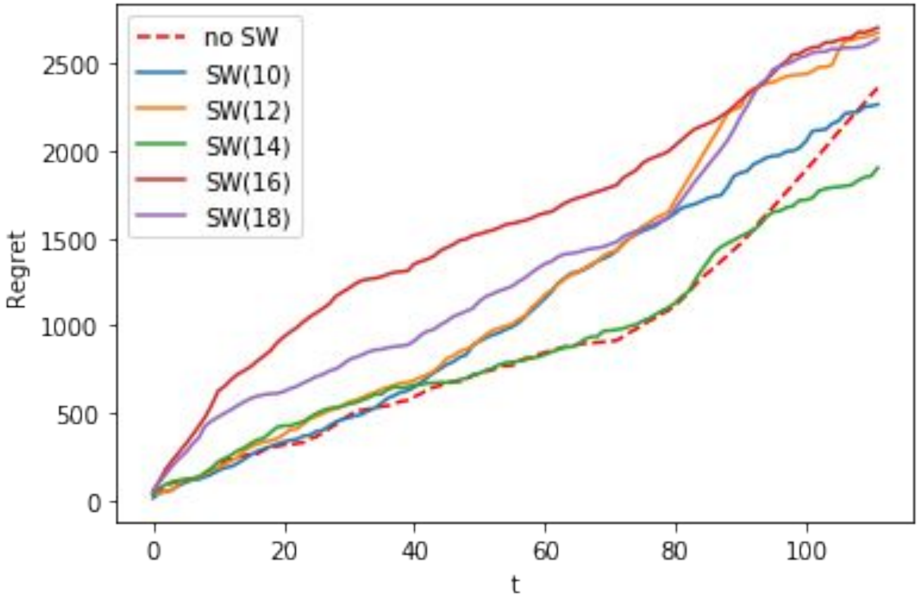
2. Execute the Knapsack algorithm

$$\{\hat{y}_j\}_{j \in N} \leftarrow Knapsack(\{(v_j \tilde{n}_{a,j}(y), \bar{y}_j\}_{j \in N})$$

3. For every subcampaign $j \in N$, play arm $\hat{y}_j$

4. Update the GP according to the **last observed rewards** (according to the sliding window size)

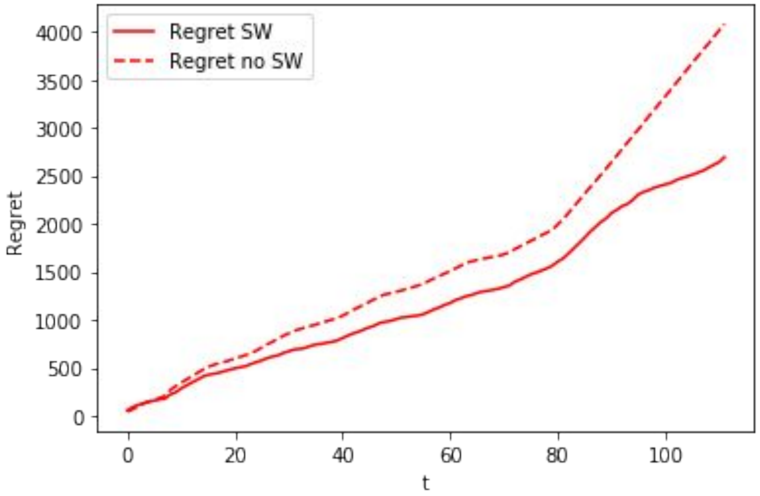
Sliding window size setting

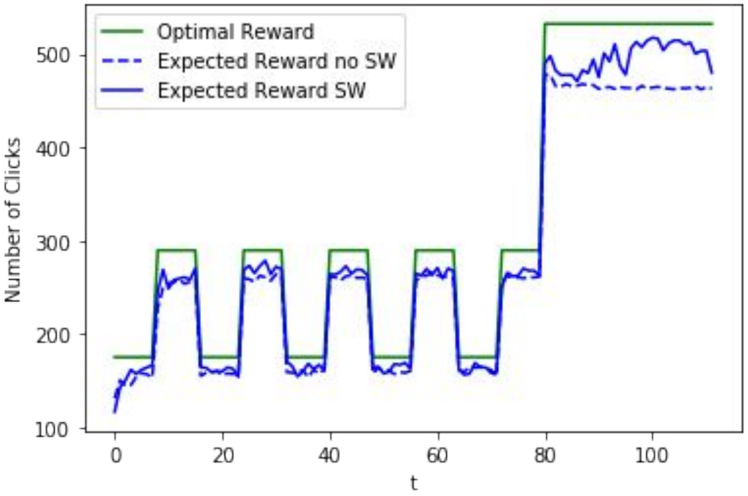


- 3 experiments
- 1 GP-TS learner
- 5 GP-TS-SW different learners

Results

Performing 10 experiments with a sliding window of 14 and averaging





Part 4 and 5
Pricing Campaign

Part 4 - Pricing without Context Generation

For the pricing scenario, we have chosen 5 candidates of prices, accordingly to the assumption with the budget of the different classes of users, and the companies in the competition: 5, 10 ,15, 20, 25 euros.

In the First installment of the pricing task, we used a Thompson Sampling MAB to learn the Expected Value Distribution for the different candidates.

Learning the Aggregated Distribution

Thompson Sampling

1. At every time t, for every arm a

$$\tilde{\theta}_a \leftarrow Sample(\mathbb{P}(\mu_a = \theta_a)) \text{ (Beta Distribution)}$$

2. At every t play a such that:

$$a_t \leftarrow \arg\max_{a \in A} \{\tilde{\theta}_a price_a\}$$

3. Update the Beta distribution of arm as:

$$(\alpha_{a_t}, \beta_{a_t}) \leftarrow (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t})$$

For each incoming user, the MAB chooses the arm to propose to it, based on the estimated probability of the arm multiplied by the value of the candidate.

In this way, at each instant of time, Thompson Sampling chooses the candidate with the best estimated expected value.

Then it updates its Beta Distribution given the result.

Aggregated Regret

To evaluate the performance of the MAB, we compared it to the performance of a clairvoyant algorithm, that at each instant of time chooses the best candidate given the aggregated distribution of the expected values

The expected values of each candidate for the aggregated case is determined with a ponderate mean of the values of the candidate for each class of users, multiplied by the class probability.

$$R_{T_{aggregated}}(U) := \sum_{t=1}^{T} (\mu_{a^*} - \mu_{a_t}) = T\mu_{a^*} - \sum_{t=1}^{T} \mu_{a_t}$$

Disaggregated Regret

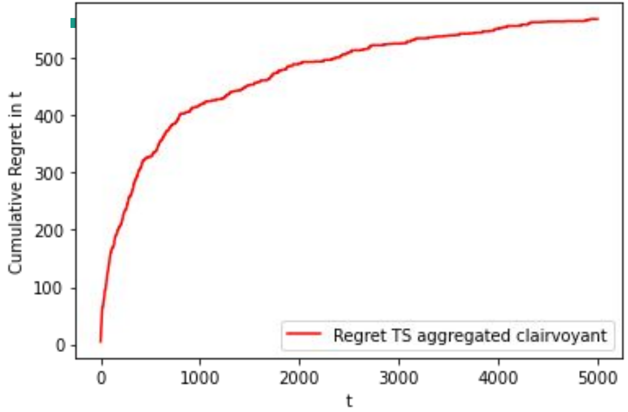
To evaluate the performance of the MAB in the disaggregated case,, we compared it to a clairvoyant algorithm that chooses the candidate with the best expected value given the class of the incoming user.

c_t: the class of the user at time t $\mu_{a^*_{c_t}}$: the best candidate for class c_t

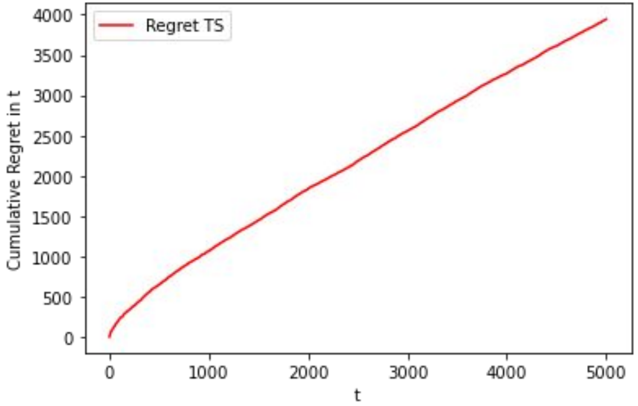
$\mu_{a_{t,c_t}}$: the expected value of the chosen candidate for class c_t

$$R_{T_{disaggregated}}(U) := \sum_{t=1}^{T} (\mu_{a^*_{c_t}} - \mu_{a_{t,c_t}})$$

Performance of the Aggregated MAB



We can clearly see that the Aggregated MAB evaluated with the Aggregated Regret converge to an optimal value with a logarithmic growth.



Instead, the Aggregated MAB doesn't converge to an optimal value using the Aggregated Regret, having a linear growth.

This happens because the optimal candidate for each class is not necessarily the optimal candidate for the aggregated case.

Part 5 - Pricing with Context Generation

In this part we explore how and if we can improve the performance of the pricing part introducing a context generation algorithm.

We runned our pricing algorithm and after a threshold (simulation of a week) we perform our context generation and then we will use this context for the next week.

We performed a greedy approach that select from the possible split the best possible feature for the split and then make it.

Split Condition

Lower bound on the probability that context c_1 occurs

Lower bound on the probability that context c_2 occurs

$$p_{c_1}\mu_{a^*_{c1},c1} + p_{c_2}\mu_{a^*_{c2},c2} \geq \mu_{a^*_{c0},c0}$$

Lower bound on the best expected reward for context c_1

Lower bound on the best expected reward for context c_2

Lower bound on the best expected reward for the context c_0

Specific case: limitation of the model

In our specific case the feature space is very small, so we can compute no more than 3 split. We can see than that the exploring phase in which is chosen the best feature in order to do the split is quite useless in the long term.

Using different conversion rate for the three model of user the common result at the end of the experiment is to do 1 or 2 (all) the possible split and the order in which the split are performed did not affect so much the final result in this specific case with a small number of features.

Log, fetch of the log

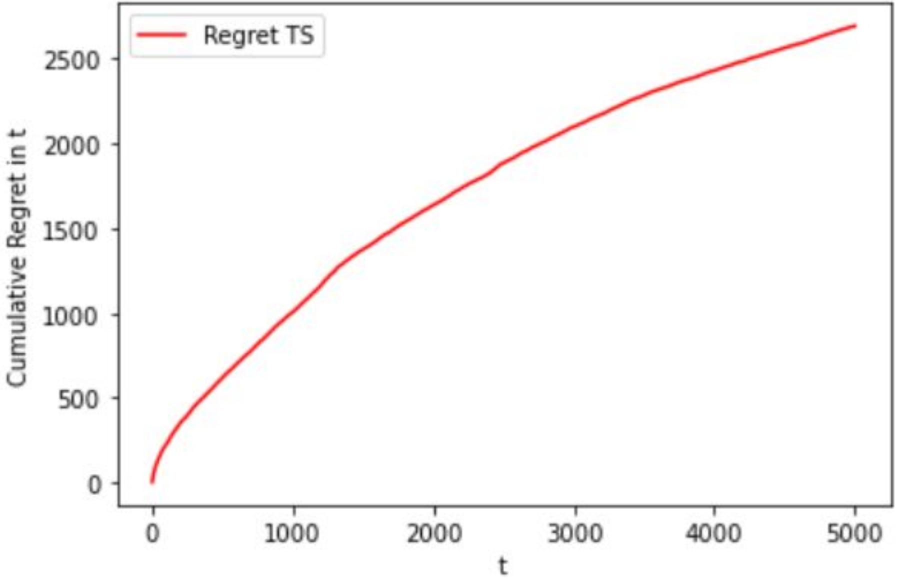
In order to evaluate the split condition we use a log that has all the information of the users. And then we use a fetch function

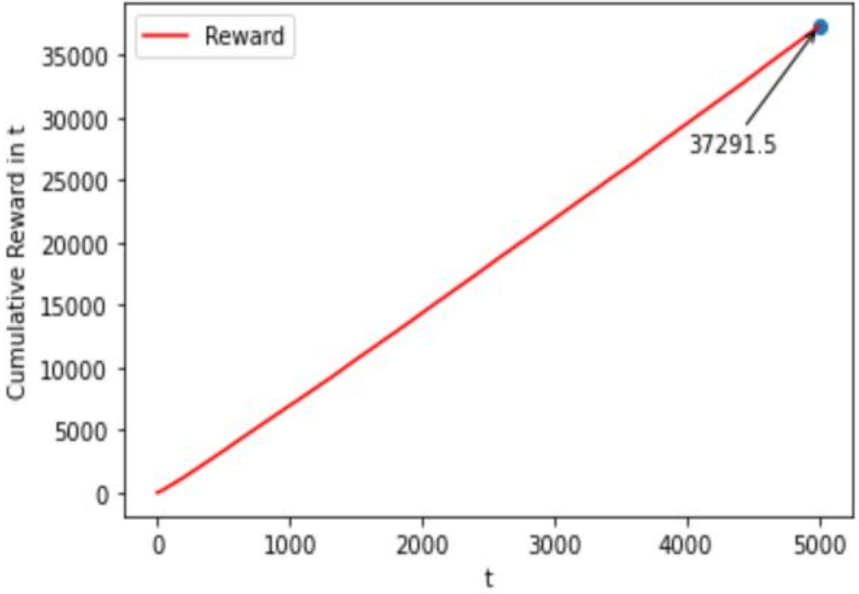
```
def fetch_log(self, feature):
```

to evaluate the split condition and also, if we make the split to build new learner for the new two subspace of features with the past information regarding their subspace of features.

Results

Performing 10 experiment with 5000 people each and a week of 1200 people





Lower bound and not perfect splitting

After running different experiments with the same setting as above, we saw that not all the time the algorithm perform 2 split. This is due to the lower bound that is quite conservative and make the split possible only if there are **strong** evidence of a worth split.

This is the Hoeffding bound used:

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

Lower bound and not perfect splitting

		Familiarity	
		f	u
Age	y	0	1
	a	0	1

splitting at 1199

		Familiarity	
		f	u
Age	y	0	1
	a	0	1

splitting at 2399

		Familiarity	
		f	u
Age	y	0	1
	a	0	1

splitting at 3599

		Familiarity	
		f	u
Age	y	0	1
	a	0	1

splitting at 4799

```
Performing experiment: 2
```

		Familiarity	
		f	u
Age	y	0	0
	a	1	1

splitting at 1199

		Familiarity	
		f	u
Age	y	0	0
	a	1	1

splitting at 2399

		Familiarity	
		f	u
Age	y	0	2
	a	1	1

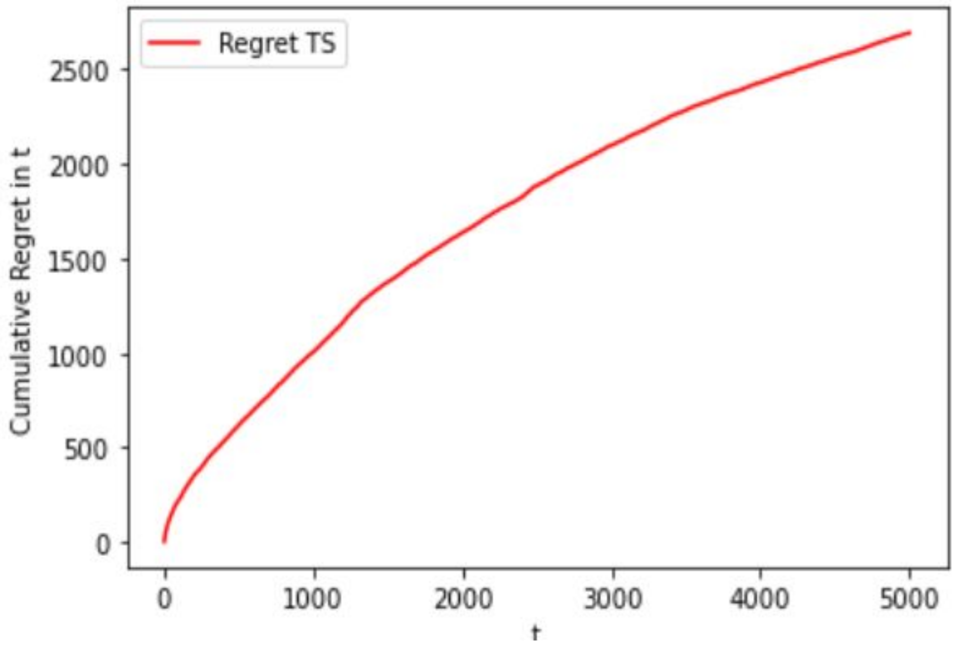
splitting at 3599

		Familiarity	
		f	u
Age	y	0	2
	a	1	1

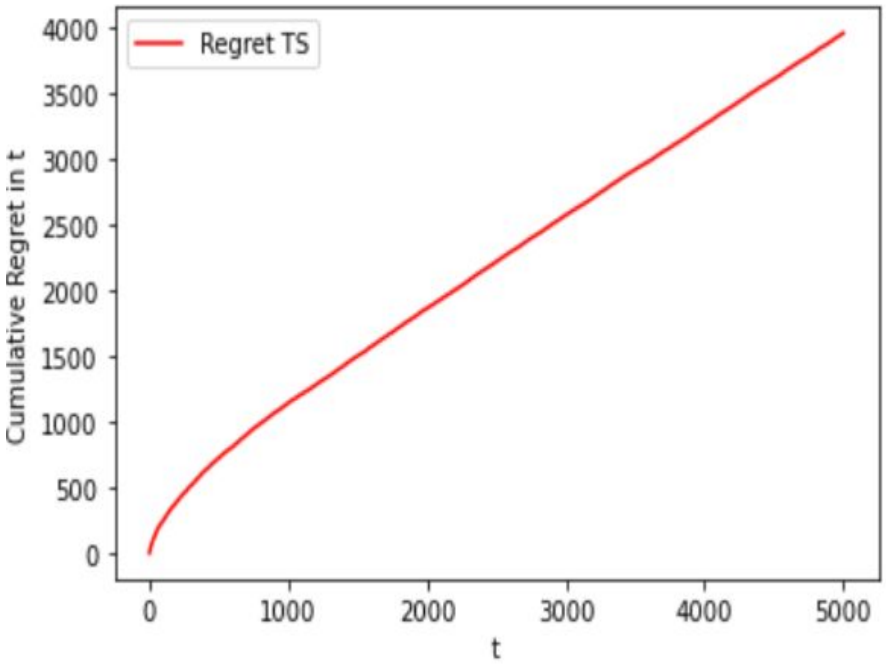
splitting at 4799

```
Performing experiment: 3
```

Comparison with the non-context generation



Context generation



Non context generation

Part 6 and 7
Combining the two algorithms

Part 6 - Different prices for different contexts

In this setting we combine the algorithms of **Advertising** and **Pricing**, trying to decide the allocation of budget in the different sub campaigns when we are learning both conversion rates and budget allocation.

No abrupt phases.

Seller knows a priori that each sub campaign is associated to a different class of users and charging different price to every context.

With this assumption the two problems can be decomposed.

How?

We decide to compute the best number of clicks for each sub campaign using the Knapsack optimization algorithm but this time we use as **value-per-click** for each sub campaign the highest product of number of clicks and the expected rewards.

The expected rewards are obtained at each round by the Pricing algorithm to be used in the next one.

The Pricing algorithm runs with the number of users that are actually observed from the environment with the currently budget allocation.

How?

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
C0	0.000	110.621	151.316	166.287	171.795	173.821	174.566	174.840	174.941	174.978	174.992
C1	0.000	51.924	71.026	78.053	80.638	81.589	81.939	82.068	82.115	82.133	82.139
C2	0.000	45.151	61.762	67.872	70.120	70.947	71.252	71.363	71.405	71.420	71.425

✖

P_{best} * Conversion_Rate_$C_0 P_{best}$
P_{best} * Conversion_Rate_$C_1 P_{best}$
P_{best} * Conversion_Rate_$C_2 P_{best}$

Pseudocode

At every time $t \in T$

1. For every subcampaign $j \in N$, for every arm $a \in A$:

   $\tilde{n}_{a,j} \leftarrow Sample(\mathbb{P}(\mu_{a,j} = n_{a,j}))$

2. For every expected value $v_{p,j}$ for every price p, for every class of users j

   $v_j \leftarrow \max_p(v_{p,j})$

3. Execute the Knapsack algorithm:

   $\{\hat{y}_j\}_{j \in N} \leftarrow Knapsack(\{(v_j * \tilde{n}_{a,j}(y), \bar{y}_j\}_{j \in N})$

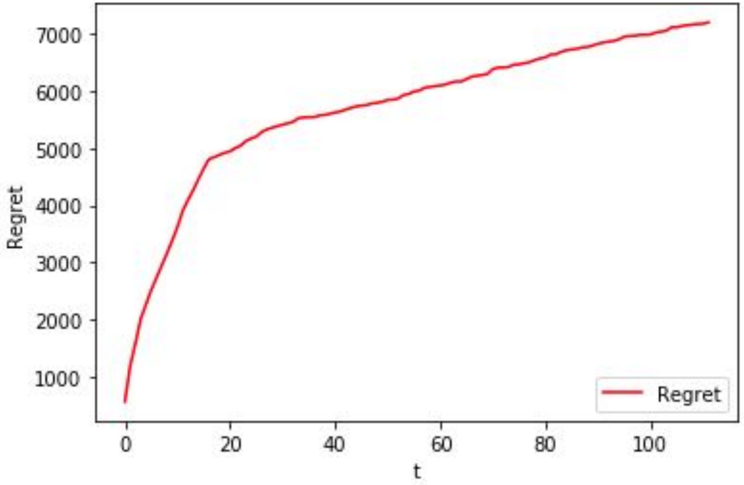
4. For every subcampaign $j \in N$, play arm $\hat{y}_j$ and save the rewards of the current t $\{y_{j,t}\}_{j \in N}$

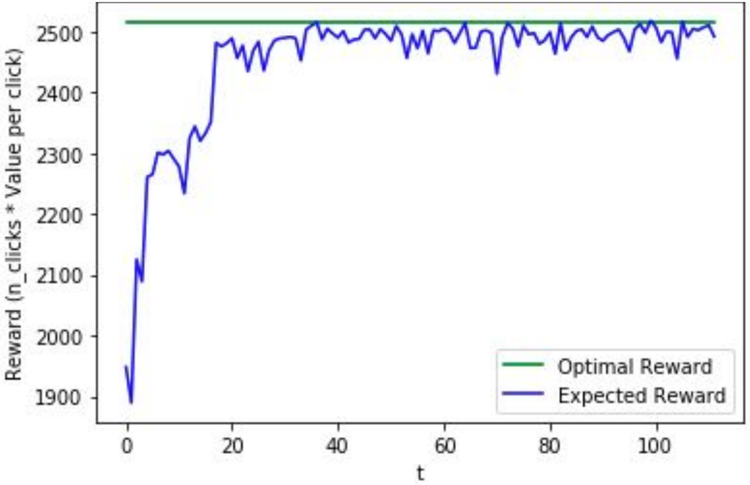
5. Update the GP according to the observed rewards so far

6. Execute the Pricing algorithm with $\{y_{j,t}\}_{j \in N}$ numbers of users for each class $j \in N$ and update the expected values $v_{p,j}$ to be used in the next round

Results

Performing 10 experiments and averaging





Part 7 - Same price to all classes

In this setting we assign the same price to every class of users.

To select which price we run the previous algorithm as many time as the number of price values.

Goal: Know whether the pricing algorithm is useful combined with the budget optimization one.

How?

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
C0	0.000	110.621	151.316	166.287	171.795	173.821	174.566	174.840	174.941	174.978	174.992
C1	0.000	51.924	71.026	78.053	80.638	81.589	81.939	82.068	82.115	82.133	82.139
C2	0.000	45.151	61.762	67.872	70.120	70.947	71.252	71.363	71.405	71.420	71.425

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
C0	0.000	110.621	151.316	166.287	171.795	173.821	174.566	174.840	174.941	174.978	174.992
C1	0.000	51.924	71.026	78.053	80.638	81.589	81.939	82.068	82.115	82.133	82.139
C2	0.000	45.151	61.762	67.872	70.120	70.947	71.252	71.363	71.405	71.420	71.425

✖

P_1 * Conversion_Rate_C_0P_1
P_1 * Conversion_Rate_C_1P_1
P_1 * Conversion_Rate_C_2P_1

• • •

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
C0	0.000	110.621	151.316	166.287	171.795	173.821	174.566	174.840	174.941	174.978	174.992
C1	0.000	51.924	71.026	78.053	80.638	81.589	81.939	82.068	82.115	82.133	82.139
C2	0.000	45.151	61.762	67.872	70.120	70.947	71.252	71.363	71.405	71.420	71.425

✖

P_5 * Conversion_Rate_C_0P_5
P_5 * Conversion_Rate_C_1P_5
P_5 * Conversion_Rate_C_2P_5

Pseudocode

At every time $t \in T$

1. For every subcampaign $j \in N$, for every arm $a \in A$:

   $\tilde{n}_{a,j} \leftarrow Sample(\mathbb{P}(\mu_{a,j} = n_{a,j}))$

2. For every value of price p execute the Knapsack algorithm with a diffrent table, whose element are $v_{p,j} * \tilde{n}_{a,j}$

   $\{\hat{y}_{p,j}\}_{p \in P, j \in N} \leftarrow Knapsack(\{(v_{p,j} * \tilde{n}_{a,j}(y), \bar{y}_j\}_{p \in P, j \in N})$

3. Select the knapsack result corresponding to a value of price $p*$ such that:

   $\{\hat{y}_{p^*,j}\}_{j \in N} \leftarrow \arg\max_p \sum_a \{\hat{y}_{p,j}\}_{p \in P, j \in N}$

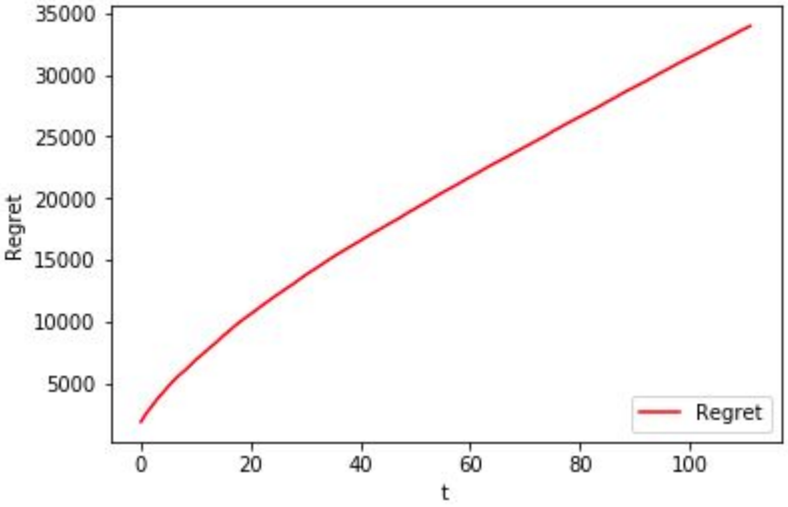
4. For every subcampaign $j \in N$, play arm $\hat{y}_{p*,j}$ and save the rewards of the current t $\{y_{j,t}\}_{j \in N}$

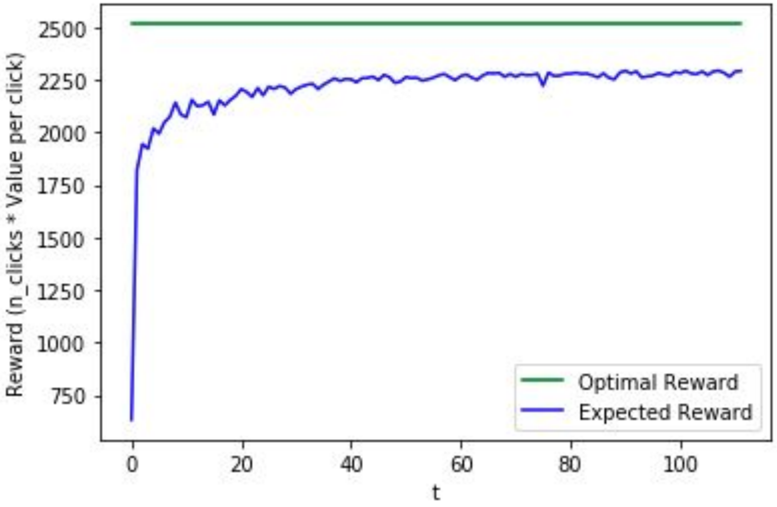
5. Update the GP according to the observed rewards so far

6. Execute the Pricing algorithm with $\{y_{j,t}\}_{j \in N}$ numbers of users for each class $j \in N$ and update the expected values $v_{p,j}$ to be used in the next round

Results

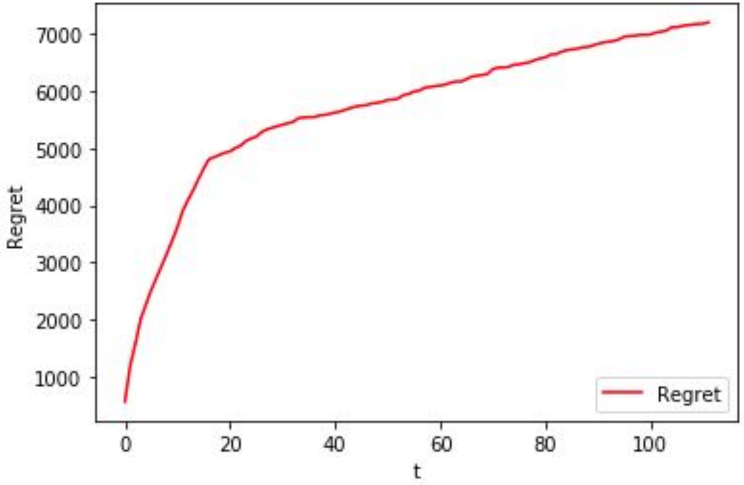
Performing 10 experiments and averaging



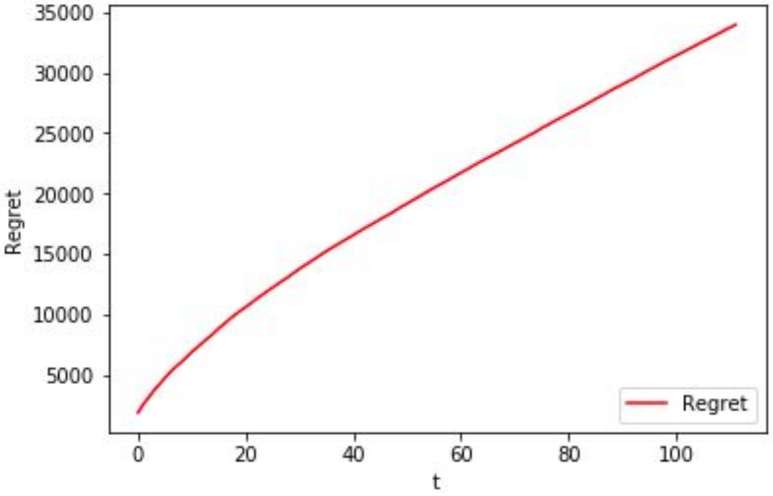


Comparing the two approaches - Regret

Different prices for different classes

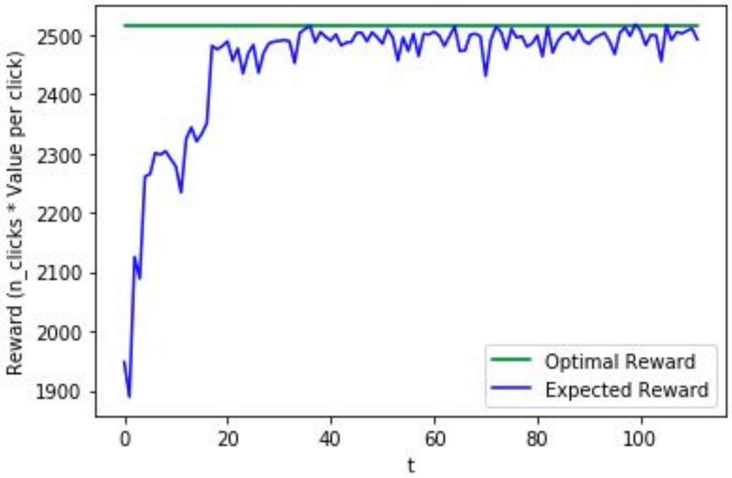


Same price for every class

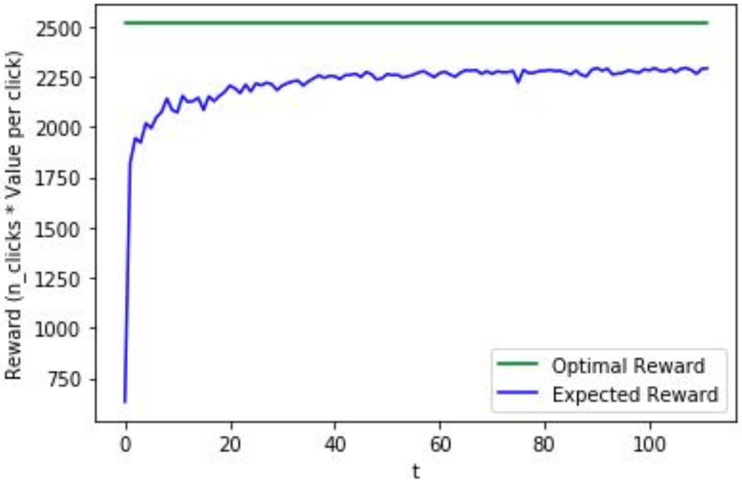


Comparing the two approaches

Different prices for different classes

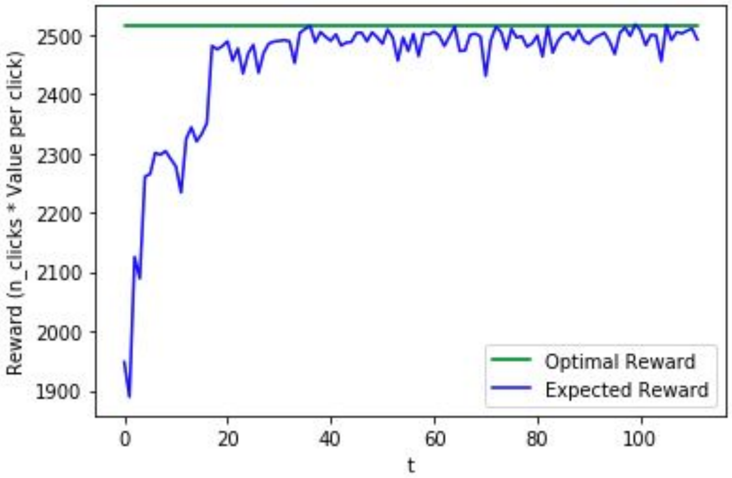


Same price for every class

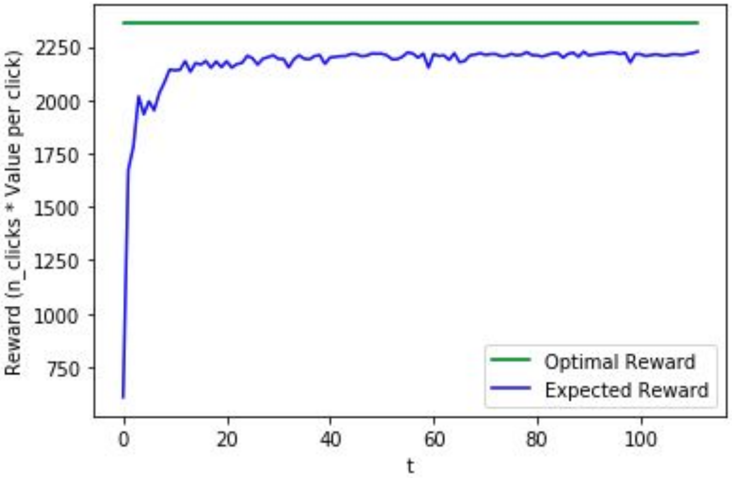


Comparing the two approaches - alt Clairvoyant

Different prices for different classes



Same price for every class



Thanks for the attention!