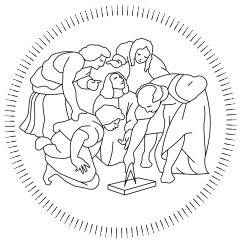

POLITECNICO
MILANO 1863

Data Intelligence Applications

Project report

Advertising and Pricing

Authors:
Eknid Mucollari
Stefano Minotti

Professor:
Nicola Gatti

July 15, 2021

Contents

1 Introduction

In this document we will describe our results for the advertising and pricing project. We begin introducing the scenario end the environment we modeled, than we'll move to the theoretical formulation of the problem describing the objective function and all the involved variables (steps 1 and 2) and finally we'll present our solutions for the practical steps (3 to 7) of the project.

2 Scenario and environment

In our scenario an e-commerce website specialized in fitness products has to face the problem of selling its new sport drink. The problem needs to be tackled under two aspects:

- Advertising: advertisement is used to attract users on the website, we suppose to have an unlimited budget for our campaign but each new click involves a cost based on the advertising bid.
- Pricing: each new user will decide whether to buy or not the product at a certain price, once a user makes a purchase then it may come back and buy again the product at the same price during the following 30 days.

The product can be sold in a price range of [0.5, 5] euros while the bid for the advertising campaign is in a range of [0.8, 8] euros.

2.1 Customer classes

We supposed that the advertising campaign is carried on through Facebook. With Facebook's advertising we can discriminate the users by estimating two binary features:

- Under 30: people under 30 years old.
- Sporty: people that have shown interest in fitness or sport in general.

Between the four possible combinations of these two features the advertising campaign will target only three classes of users.

Under 30 \ **Sporty**	False	True
False		*C2*
True	*C1*	*C3*

Each class of users is characterize by:

- A stochastic number of daily clicks of new users as a function depending on the bid.
- A stochastic cost per click as a function of the bid.
- A conversion rate function providing the probability that a user will buy the product given a price.
- A distribution probability over the number of times the user will come back to the website to buy the product during the next 30 days after the first purchase.

2.2 Class functions

2.2.1 Daily clicks

The daily clicks function is the following:

$$daily\_clicks(bid) = max\_clicks \cdot \left(1 - e^{-slope \cdot bid}\right) + \epsilon$$

where ϵ is a Gaussian noise and the slope is calculated as:

$$slope = \frac{-\log(0.01)}{saturating\_bid}.$$

In practice this parameter is calculated in a way that the daily clicks function reaches the 99% of its max value when the bid is *saturating_bid*

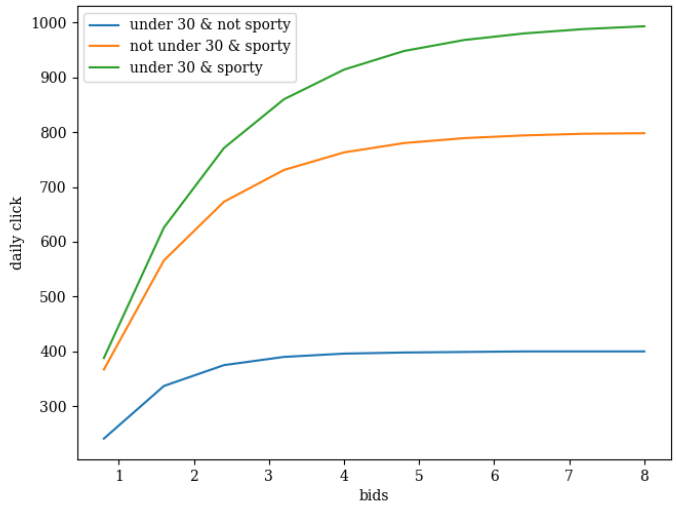

Figure 1: Daily click functions

2.2.2 Cost per click

The cost per click function is linear and is modeled as a coefficient depending on the customer class multiplied by the bid with the addition of a Gaussian noise:

$$cpc(bid) = \mu \cdot bid + \epsilon$$

The more a class is subject to be targeted by advertising campaigns the more the coefficient μ will be higher.

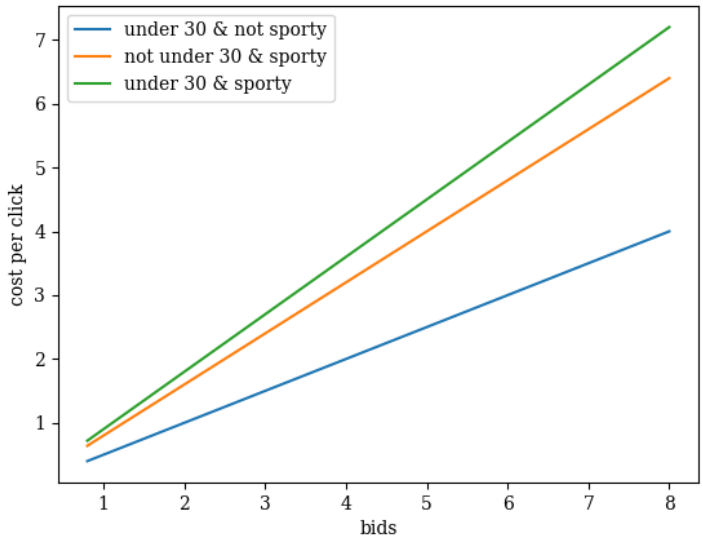

Figure 2: Cost per click functions

2.2.3 Conversion rate

The conversion rate function is generated as the linear interpolation of a set of points. The function is monotonically decreasing for *Under 30 - Sporty* and *Under 30 - Not Sporty* while for the class *Over 30 - Sporty* the function is initially increasing because the users of this class may be diffident at buying a product with a price too low.

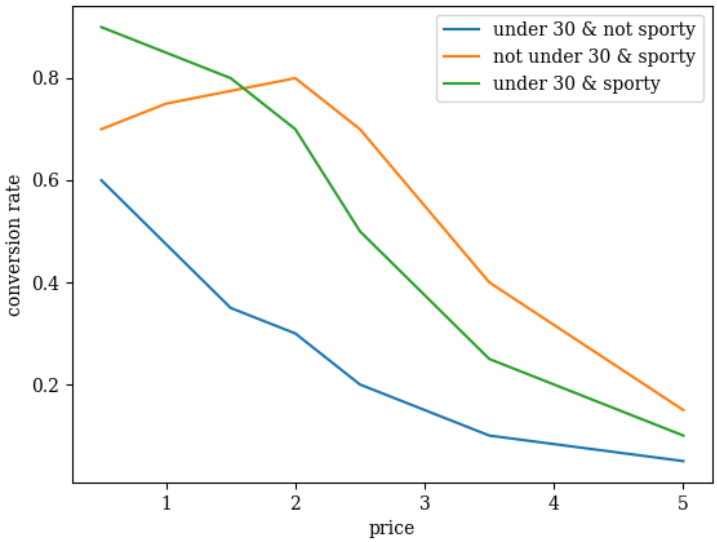

Figure 3: Conversion rate functions

2.2.4 Returns probability distribution

The distribution probability function over the number of times a user will return to buy the product is a discrete Gaussian distribution truncated between 0 and 30. The mean is higher as more as a class will use the product.

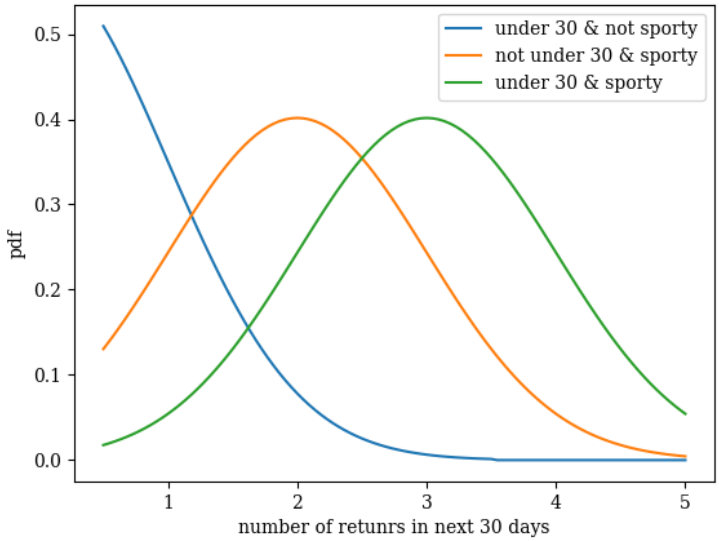

Figure 4: Returns probability distributions

3 Optimization problem

3.1 Objective function

The problem of finding the best pricing and bidding strategy can be solved by finding the pair of price and bid for each customer class that maximizes the following objective function:

$$\sum_{i \in classes} daily\_clicks_i(b_i) \cdot \{conv\_rate_i(p_i) \cdot p_i \cdot (1 + \mathbf{E}[returns_i]) - cpc_i(b_i)\}$$

where p_i and b_i are the bid and the price for customer class i and $\mathbf{E}[returns_i]$ is the expected number of time a customer of class i will return to buy the product in the following 30 days. We will consider a time horizon of 365 days.

3.2 Optimal solution

Given a set of candidates for the prices and a set of candidates for the bids a possible algorithm to obtain the optimal pair of price and bid for each customer class is the following:

best_price_bid_pairs $\leftarrow$ [];
for $i \in$ *classes* **do**
  best_p $\leftarrow 0$;
  best_p_reward $\leftarrow 0$;
  for $p \in$ *prices* **do**
    p_reward $\leftarrow$ *conv_rate*$_i(p) \cdot p$;
    if *p_reward* $\geq$ *best_p_reward* **then**
      best_p $\leftarrow p$;
      best_p_reward $\leftarrow$ *p_reward*;
    end
  end
  best_b $\leftarrow 0$;
  best_b_reward $\leftarrow -\infty$;
  for $b \in$ *bids* **do**
    b_reward $\leftarrow$ *daily_clicks*$_i(b) \cdot \{$*best_p_reward* $\cdot (1 + \mathbf{E}[returns_i]) -$ *cpc*$_i(b)\}$;
    if *b_reward* $\geq$ *best_b_reward* **then**
      best_b $\leftarrow b$;
      best_b_reward $\leftarrow$ *b_reward*;
    end
  end
  append $(best\_p, best\_b)$ to *best_price_bid_pairs*;
end
return *best_price_bid_pairs*.

The algorithm first iterates over all the prices to maximize the reward term that depends only on the price and then, fixed the best price, iterates over all the bids to maximize the full objective function. This is done for each customer class. The complexity of the algorithm is in the order of $O(\#classes \cdot (\#prices + \#bids))$

3.3 Online learning version

In a real scenario the optimization problem described must be solved in an online fashion. All the parameters of the objective function are unknown, except for the sets of candidate prices and bids. At each day we have to select a pair of price and bid for each customer class, observe the realization of the unknown parameters and update our estimation on them. In our problem we have to keep in mind that we will have a delay in the feedback. The reward for a day will be available only after 30 days because it have to keep into consideration the number of times a user will return to buy the product in the 30 days after his first purchase. So every online learning algorithm will require 30 days of "bootstrap" before collecting the first reward and being able to start optimizing.

4 Pricing under fixed bid

4.1 Problem description

The goal is finding the price the maximizes the reward keeping the bid fixed. In this case we don't discriminate over the customer classes, so we have to work with aggregate data and find a single price for all the three classes. The problem reduces to the maximization of:

$$conv\_rate(p, b) \cdot p \cdot (1 + \mathbf{E}[returns]_{p,b})$$

where conversion rate and expected returns are the aggregated values for the three classes. In this problem we have to consider also the expected returns in the reward function because even if it does not depend on the price when considering the customer classes individually, it will depend on the conversion rates of the single classes and so on the price when considering aggregate data. In fact the aggregated expected returns is:

$$\mathbf{E}[returns]_{p,b} = \frac{\sum_{i \in classes} daily\_clicks_i(b) \cdot conv\_rate_i(p) \cdot \mathbf{E}[returns_i]}{\sum_{i \in classes} daily\_clicks_i(b) \cdot conv\_rate_i(p)}$$

To be more clear consider the following example. We have two possible prices ($p_1 = 1$, $p_2 = 2$) and two classes:

- For the first class we have $conv\_rate_1(p_1) = conv\_rate_1(p_2) = 0.1$ and $\mathbf{E}[returns_1] = 2$
- For the second class we have $conv\_rate_2(p_1) = 0.4$ and $conv\_rate_1(p_2) = 0.1$ and $\mathbf{E}[returns_1] = 0$

Assuming the same daily clicks for the two classes we can calculate:

- The aggregate conversion rates, $conv\_rate(p_1) = 0.25$ and $conv\_rate(p_2) = 0.1$.
- The aggregate expected returns, $\mathbf{E}[returns]_{p_1} = 0.4$ and $\mathbf{E}[returns]_{p_2} = 1$.

Consider:

$$v_1 = conv\_rate(p_1) \cdot p_1 = 0.25$$
$$v_2 = conv\_rate(p_2) \cdot p_2 = 0.2.$$

We have:

$$v_1 > v_2$$

but:

$$0.35 = v_1 \cdot (1 + \mathbf{E}[returns]_{p_1}) < v_2 \cdot (1 + \mathbf{E}[returns]_{p_2}) = 0.4.$$

So the best price will be wrong without considering the aggregated expected returns.

4.2 Algorithm

We designed two different bandit algorithms for this problem. The first algorithm is based on an upper confidence bound approach while the second is based on Thompson sampling.

4.2.1 Upper confidence bound

The algorithm estimates the expected conversion rate and the expected number of returns for each price p (arm) as the mean of all the collected samples. At each round the upper bounds for the two unknown variables are calculated for each arm. For the conversion rate the upper bound is:

$$conv\_rate\_ucb_p = \overline{conv\_rate_p} + \sqrt{\frac{\log(\#customers)}{\#customers_p}},$$

where $\#customers$ is the total number of observed customers and $\#customers_p$ is the number of customers observed for price p. For the expected returns the upper bound is:

$$returns\_ucb_p = \overline{returns_p} + 30 \cdot \sqrt{\frac{\log(\#converted\_customers)}{\#converted\_customers_p}}.$$

At each round t the selected price is:

$$p_t \leftarrow \arg\max_{p \in prices} \{conv\_rate\_ucb_p \cdot p \cdot (1 + returns\_ucb_p)\}$$

4.2.2 Thompson sampling

In the Thompson sampling version the algorithm estimates, for each price, the parameters of a Beta distribution using the observations on user conversions while using the observations on the number of returns for each converted customer it estimates the mean and standard deviation of a Gaussian distribution. At each round t a sample is drawn from the two distribution for each price. If we call $conv\_rate\_s_{p,t}$ and $returns\_s_{p,t}$ the samples, the played arm is:

$$p_t \leftarrow \underset{p \in prices}{\arg\max} \{conv\_rate\_s_{p,t} \cdot p \cdot (1 + returns\_s_{p,t})\}$$

4.3 Results

We have run experiments for both the UCB and TS algorithms and averaged the daily results of each experiment. The settings were the following:

- Number of experiments: 10
- Rounds horizon: 365 days
- Returns horizon: 30 days
- Fixed bid: 1.6€
- Price arms: [0.5€, 1€, 1.5€, 2€, 2.5€, 3€, 3.5€, 4€, 4.5€, 5€]

We have a "bootstrap" of 30 days before being able to collect the first reward because the number of times a customer returned to buy our product is available after 30 days from his first purchase. For these days, in this and all the next problems, the arms have to be pulled at random. The two algorithms performed very similar, both converged to the optimal price of 2€ but TS provided a slightly lower cumulative regret.

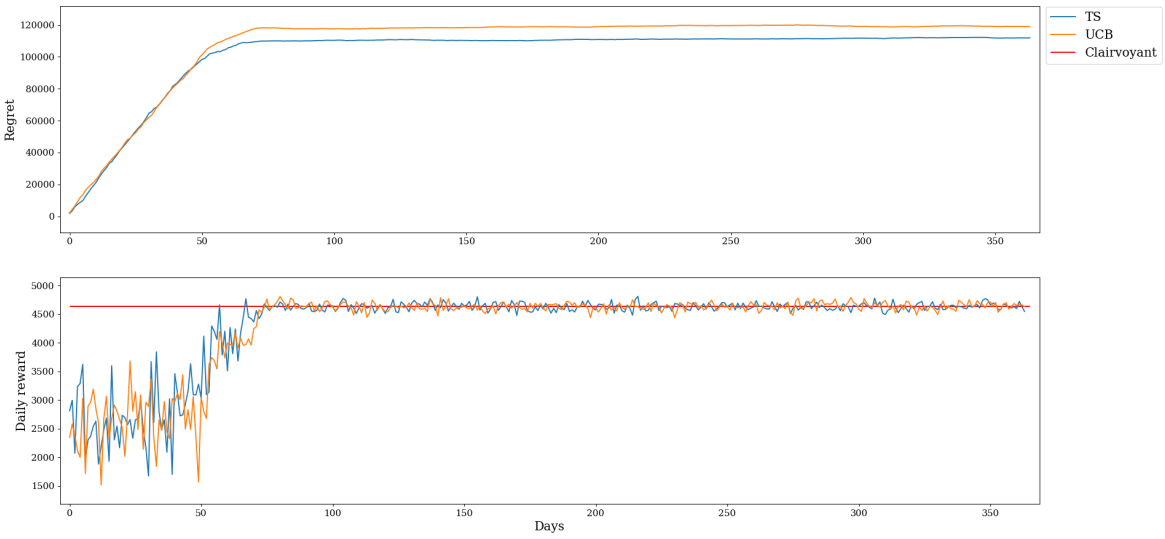

Figure 5: Pricing under fixed bid results

5 Pricing under fixed bid with discrimination

5.1 Problem description

The problem is very similar to the one before. The difference is that in this case we have to adopt a context generation approach in order to discriminate the user classes and select a potentially different price for each class.

5.2 Algorithm

The solution of the problem can be reached by using two algorithms cooperating one with the other. The first one is one of the two optimizations algorithms presented in the previous section that is used to identify the optimal price for each context. The second algorithm is a context generation algorithm that uses the rewards collected during time to periodically generate the contexts. We implemented two versions of the context generation algorithm, a greedy one and a brute force.

5.2.1 Greedy context generation

The goal of the algorithm is building a binary tree in which each node represents the feature that has been split. Then each path from the root to a leaf represents a context. In order to generate the tree we need to perform the following steps for each node:

1. For every feature i that has not already been split, evaluate the value v_i of the split of context c_0 into the two contexts c_1, c_2 obtained by splitting the customer classes of c_0 based on feature i.

   $$v_i = \underline{p}_{c_1} \cdot \underline{\mu}_{p^*_{c_1},c_1} + \underline{p}_{c_2} \cdot \underline{\mu}_{p^*_{c_2},c_2}$$

   where $\underline{p}_{c_i}$ is the lower bound of the probability of context c_i and $\underline{\mu}_{p^*_{c_i},c_i}$ is the lower bound of the expected reward for the optimal price of context c_i ($p^*_{c_i}$).

2. Among all the features for which holds:

   $$v_i \geq \underline{\mu}_{p^*_{c_0},c_0}$$

   split the one with the largest v_i

The best arm for each context is identified by training a bandit over all the previously collected samples belonging to that context. The lower bound on the probability of a context is calculated using the Hoeffding formula:

$$\underline{p}_{c_i} = \frac{\#customers_{c_i}}{\#customers} - \sqrt{-\frac{log(\delta)}{2 \cdot \#customers}}$$

where δ is the confidence, $\#customers_{c_i}$ is the number of observed customers belonging to c_i and $\#customers$ is the total number of observed customers. The expected reward $\mu_{p^*_{c_i},c_i}$ is the average of the rewards of all the observed customers of the context. For a customer j the reward is calculated as:

$$r_j = \begin{cases} 0 \; if \; not \; converted \\ p_j \cdot (1 + returns_j) \; if \; converted \end{cases}$$

The lower bound on the reward of a context is the following:

$$\underline{\mu}_{p^*_{c_i},c_i} = \mu_{p^*_{c_i},c_i} - z^* \cdot \frac{\sigma_{p^*_{c_i},c_i}}{\sqrt{\#customers_{c_i}}}$$

where z^* is the z-value of the confidence (e.g. at 95% confidence $z^* = 1.96$) and $\sigma_{p^*_{c_i},c_i}$ is the standard deviation of the reward.

5.2.2 Brute-force context generation

Since we have only three customer classes, the number of possible contexts is restricted. We decided to implement also a brute-force version of the context generation algorithm. The algorithm enumerates all the possible context structures and for each of the candidates calculates its value as:

$$v_i = \sum_{c \in C_i} \underline{p}_c \cdot \underline{\mu}_{p_c^*, c}$$

where C_i is the set of contexts in the context structure i and $\underline{p}_c$ and $\underline{\mu}_{p_c^*, c}$ are the lower bounds of context probability and expected reward calculated as shown for the greedy algorithm. Among all the possible context structures the algorithm selects the one with the largest v_i.

5.2.3 Variants

For each of the two context generation algorithms we tried two alternatives:

- The first one is generating each time the contexts from scratch ignoring the context structure active at the moment.
- The second one is generating the new context structure starting from the context structure active at the moment. In this way the generation algorithm is applied to all the contexts in the active structure and the contexts are never re-aggregated.

5.3 Results

We have run experiments by using all the four combinations of price learners and context generators in two cases. One case is when the context generation is incremental and so new contexts are generated starting from the old context structure, the contexts are never re-aggregated. The other case is when each time the context structure is generated from scratch. The settings were the following:

- Number of experiments: 5
- Rounds horizon: 365 days
- Returns horizon: 30 days
- Context generation rate: 40 days
- Fixed bid: 1.6€
- Price arms: [0.5€, 1€, 1.5€, 2€, 2.5€, 3€, 3.5€, 4€, 4.5€, 5€]
- Hoeffding confidence: 95%

We have decide to reduce the number of experiments to 5 since the time required for each experiment becomes larger due to the increased complexity of the problem.

5.3.1 Incremental generation

Here we can see, for the three classes, the results of the four algorithms using incremental context generation. For all the classes the combination of TS learner and brute force context generation provided a much lower regret with respect to the other three combinations. For the third class (*Under 30-Sporty*) all the algorithms reached the optimal price of 2€ while for the first class (*Under 30-Not sporty*) one combination, UCB-BF, had difficulties converging to the optimal price that is always of 2€. The second class (*Over 30-Sporty*) has an optimal price of 2.5€ but the reward provided by this price is very close to the one provided by a price of 2€ (2264.9€ against 2008.66€), so for this class

we had experiments in which the optimal price converged to 2€ instead of 2.5€. The main reason may be that the majority of samples collected are for a price of 2€ since it is also the optimal price when the classes are aggregated. So with few samples collected for the price 2.5€ the algorithms may not detect that isolating the second class as a single context should be the best choice.

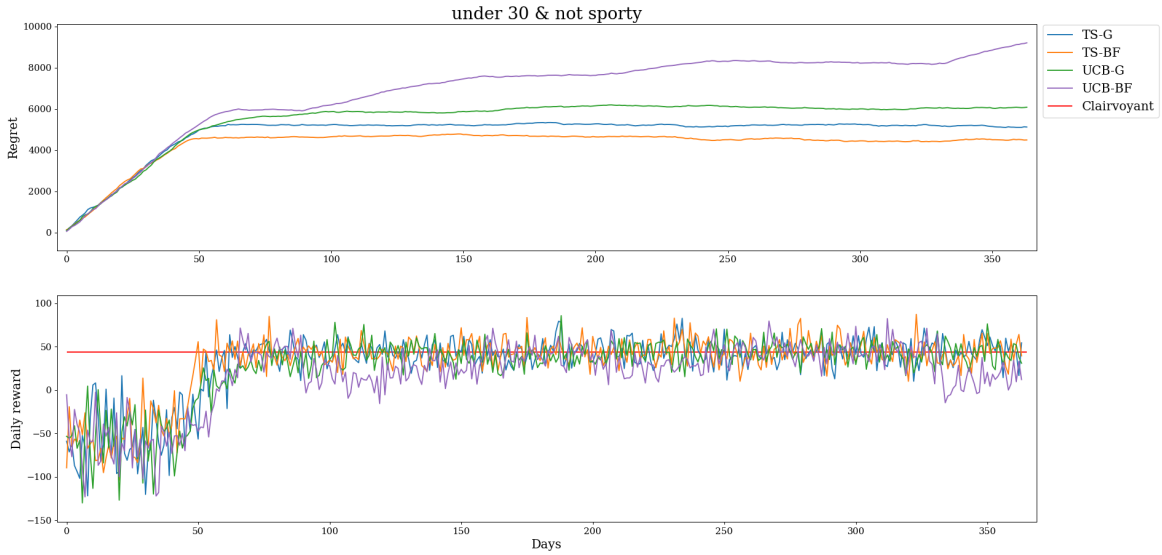

Figure 6: Pricing under fixed bid with incremental context generation results for class *Under 30-Not sporty*

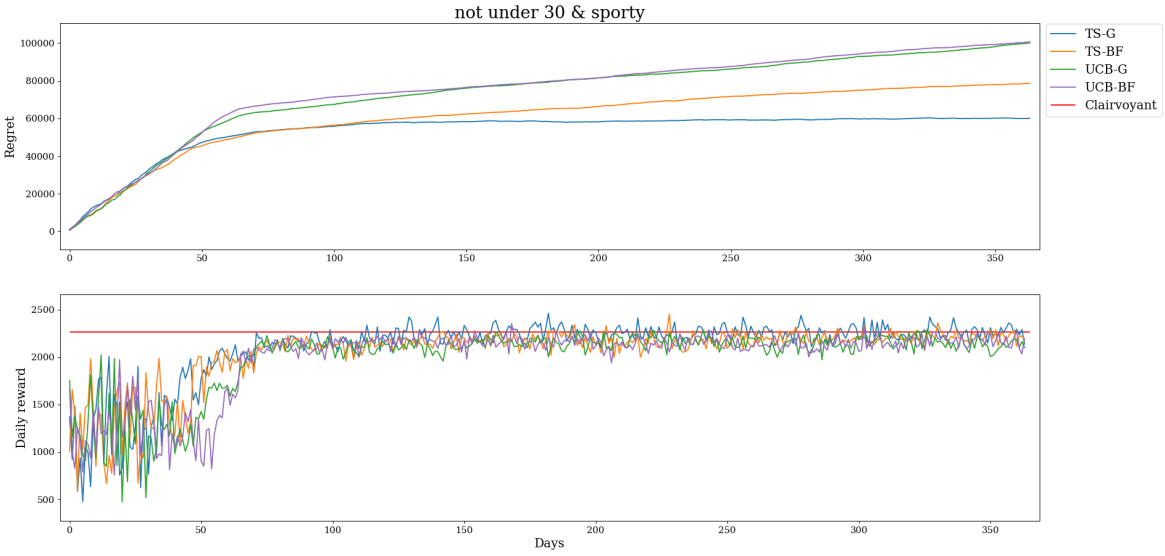

Figure 7: Pricing under fixed bid with incremental context generation results for class *Over 30-Sporty*

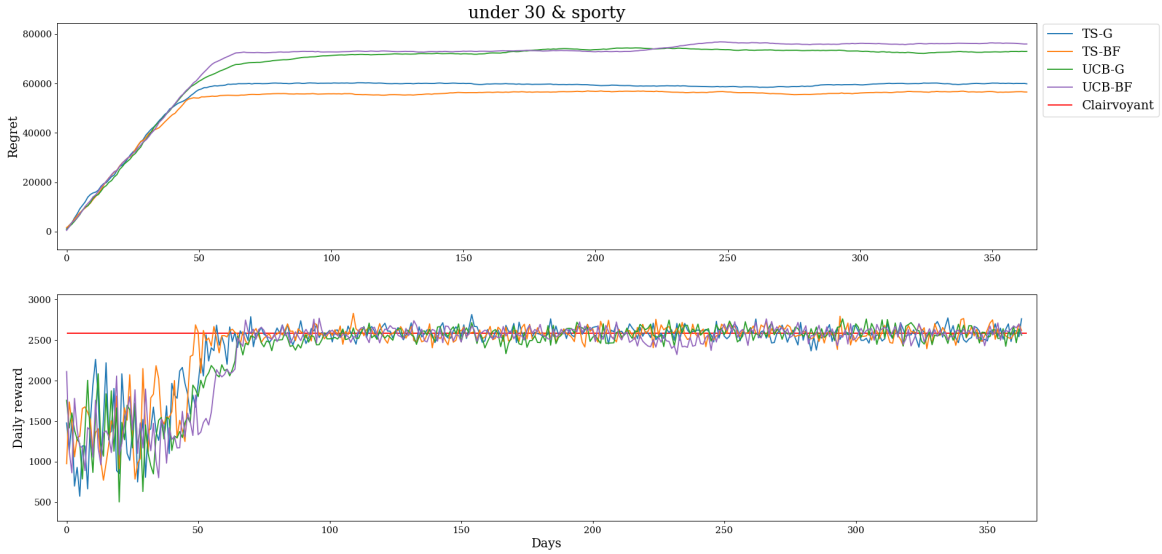

Figure 8: Pricing under fixed bid with incremental context generation results for class *Under 30-Sporty*

5.3.2 Generation from scratch

When generating each time from scratch the performances are worse with respect to the incremental generation. As we can see from the graphs, re-generating the context structure from scratch after we have reached an optimal price for a class may bring to a new structure in which it belongs to a context with an optimal price that is not the best one for the class. So we regress to a worst situation and the regret increases.

Figure 9: Pricing under fixed bid with context generation from scratch results for class *Under 30-Not sporty*

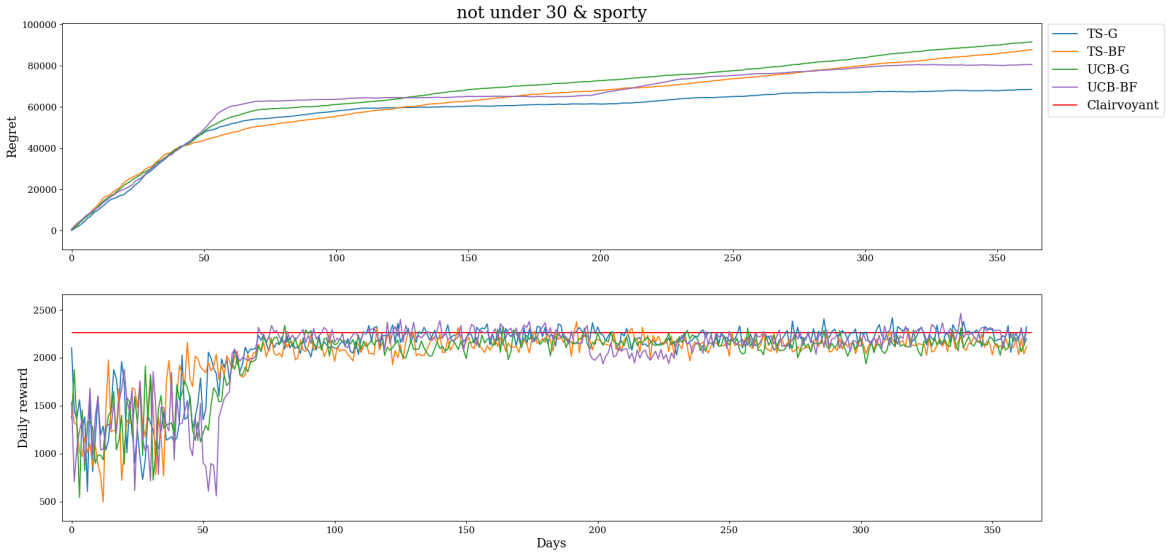

Figure 10: Pricing under fixed bid with context generation from scratch results for class *Over 30-Sporty*

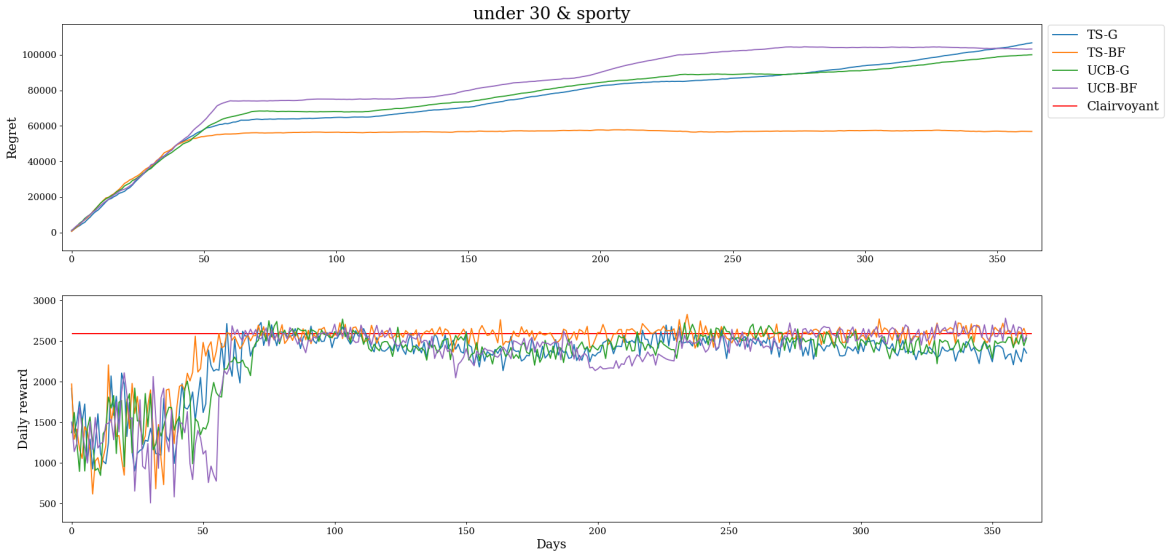

Figure 11: Pricing under fixed bid with context generation from scratch results for class *Under 30-Sporty*

6 Advertising under fixed price

6.1 Problem description

The goal is finding the bid the maximizes the reward keeping the price fixed. In this case we don't discriminate over the customer classes neither for pricing nor for advertising, so we have to work with aggregate data considering a single price and a single bid for all the three classes. The problem reduces to the maximization of:

$$daily\_clicks(b) \cdot \{conv\_rate(p,b) \cdot p \cdot (1 + \mathbf{E}[returns]_{p,b}) - cpc(b)\}$$

where all the variables are the aggregated values for the three classes. Since we are working with aggregated data we have a situation similar to the one described in section 4.1. The aggregated conversion rates and expected returns do not depends only on the price but they will also depend on the proportion with which the number of daily clicks are distributed between the three classes and so indirectly on the bid. In fact the aggregated conversion rate is:

$$conv\_rate(p,b) = \frac{\sum_{i \in classes} daily\_clicks_i(b) \cdot conv\_rate_i(p)}{\sum_{i \in classes} daily\_clicks_i(b)}$$

while the formula of the aggregated expected number of returns is the one shown in section 4.1.

6.2 Algorithm

The algorithm we designed for this problem is a bandit algorithm based on a Thompson sampling approach. The algorithm estimates, for each bid, the mean and standard deviation of two Gaussian distributions: one based on the observations of the daily number of clicks and the other based on the observations of the cost per click of each customer. Then, always for each bid, it uses the Thompson sampling algorithm presented in section 4.2.2 to estimate the expected conversion rate and the expected number of returns for the fixed price. Since the pricing bandit is initialized with a single price it will only work as an estimator of the parameters for the given price. The decision to use the bandit for this purpose has been taken to make this algorithm extensible to the case of joint bidding and pricing. At each round t a sample is drawn from the distributions of daily clicks and cost per click for each bid b. If we call $dc\_s_{b,t}$ and $cpc\_s_{b,t}$ the samples, the played arm is:

$$b_t \leftarrow \arg\max_{b \in bids} \{dc\_s_{b,t} \cdot \{conv\_rate_{b,t} \cdot p \cdot (1 + \mathbf{E}[returns]_{b,t}) - cpc\_s_{b,t}\}\}$$

where p is the fixed price and $conv\_rate_{b,t}$ and $\mathbf{E}[returns]_{b,t}$ are the values of conversion rate and returns estimated by the price learner for the bid b.

6.2.1 Variants

For this algorithm we tried two variants. They differs in the way mean and standard deviation for the daily clicks of each bid are calculated:

- In the first case the parameters are estimated as the sample mean and standard deviation of the observed daily clicks for each arm.
- In the second case the parameters are estimated by means of a Gaussian Process regressor trained on the daily samples collected.

We didn't implemented the Gaussian Process regressor for the cost per click estimation because the number of samples is very high and training the regressor would have been too computational expensive.

6.2.2 Negative rewards handling

To avoid pulling bids that may provide negative rewards we have to estimate the probability that a reward for a bid will be negative. For a bid b and a price p we can calculate the average reward of a single customer in this way:

$$v_{b,p} = \begin{cases} -cpc_b \ if \ not \ converted \\ p \cdot (1 + \mathbf{E}[returns]_{p,b}) - cpc_b \ if \ converted \end{cases}$$

The total reward for a bid b and a price p can be written as:

$$r_{b,p} = conversions_b \cdot v_{b,p,converted} + (daily\_clicks_b - conversions_{b,p}) \cdot v_{b,p,not\ converted}$$

where $daily\_clicks_b$ is the expected number of daily clicks estimated by the bandit for the bid. We want that $r_{b,p} > 0$ and by solving the inequality we can find the minimum number of conversion in order to have a positive reward:

$$min\_conversions_{b,p} = \frac{-daily\_clicks_b \cdot v_{b,p,not\ converted}}{v_{b,p,converted} - v_{b,p,not\ converted}}.$$

If we have an expected conversion rate $conv\_rate_{b,p}$ the probability distribution function over the number of conversions given $daily\_clicks_b$ daily clicks is a binomial:

$$conversions_{b,p} \sim \mathcal{B}(daily\_clicks_b, conv\_rate_{b,p}).$$

The probability of having a negative reward is:

$$P(conversions_{b,p} \leq min\_conversions_{b,p})$$

This method for estimating arms that may provide negative rewards is used also in the following problems

6.3 Results

We have run experiments for both the variants of the algorithm and averaged the daily results of each experiment. We call GPTS the algorithm that uses Gaussian Processes to estimate the daily clicks and GTS the other. The settings were the following:

- Number of experiments: 10
- Rounds horizon: 365 days
- Returns horizon: 30 days
- Fixed price: 2€
- Bid arms: [0.8€, 1.6€, 2.4€, 3.2€, 4€, 4.8€, 5.6€, 6.4€, 7.2€, 8€]
- Negative probability threshold: 0.1 (10%)

The algorithm using Gaussian Processes seems to perform slightly worst. Both the algorithms converged to the optimal bid of 1.6€ and both the algorithms estimated five playable arms (i.e. arms with negative reward probability under the threshold) [0.8€, 1.6€, 2.4€, 3.2€, 4€].

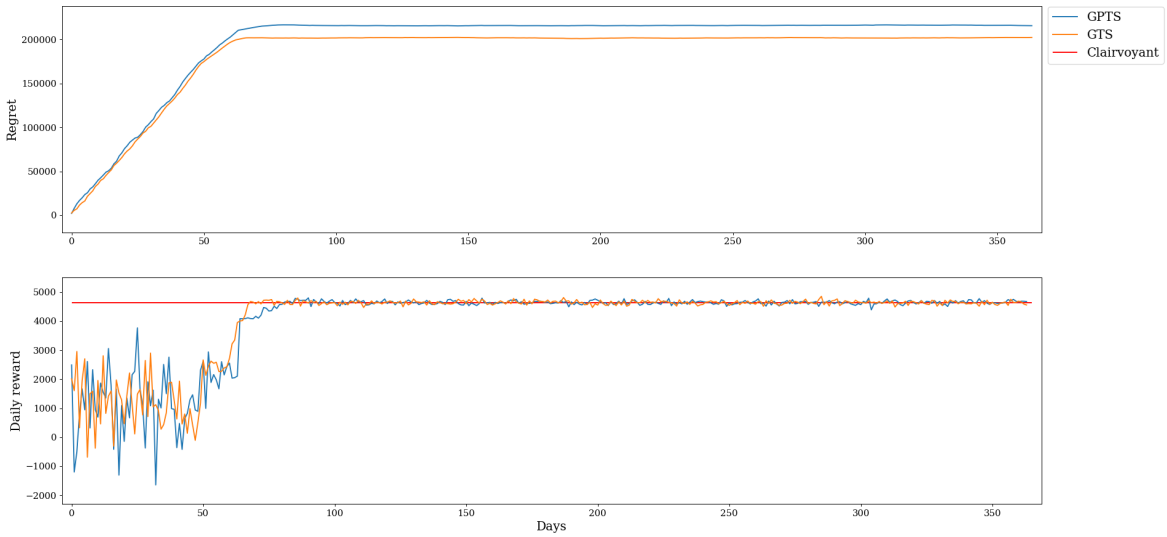

Figure 12: Advertising under fixed price results

7 Joint pricing and bidding

7.1 Problem description

Here the problem is finally jointly optimizing pricing and bidding without discriminating over the customer classes. We have to find a unique pair of bid and price for all the three classes working with aggregate data. The function to maximize is the one shown in section 6.1 and all the considerations done are still valid.

7.2 Algorithm

For this problem we have designed two algorithm. An algorithm optimizes the exact function shown before while the other one makes an approximation in order to reduce the number of samples needed to converge.

7.2.1 Exact algorithm

The exact algorithm is a variation of the algorithm shown in section 6.2 for optimizing the bid given a fixed price. Here instead of initializing the price learners with just a single price they are initialized with the whole set of possible prices. At each time step a price is pulled from the price learner of each bid and so for each bid we will have different expected conversion rates and number of returns depending on the pulled price. The algorithms selects the bid (and the relative best price) with the highest reward calculated using the sampled variables as in section 6.2.

7.2.2 Approximated algorithm

The approximated algorithm assumes that both the aggregated conversion rates and expected number of returns do not depend on the daily clicks per class and so on the bid. This may work in many cases (including our scenario) in which the daily clicks curves of the three classes have a very similar trend and so the proportions of the daily clicks for the three classes are almost constant for each bid. The algorithm treats the pricing problem as disjoint from the bid optimization. It use the Thompson sampling algorithm presented in section 4.2.2 to optimize the price independently from the bid and

to estimate the expected conversion rate and number of returns. Then given these estimations it optimizes the bid as in the case with fixed price.

7.3 Results

We have run experiments in the exact and in the approximated case using both the GTS and GPTS variants while we always used TS as price learner. The settings were the following:

- Number of experiments: 10
- Rounds horizon: 365 days
- Returns horizon: 30 days
- Price arms: [0.5€, 1€, 1.5€, 2€, 2.5€, 3€, 3.5€, 4€, 4.5€, 5€]
- Bid arms: [0.8€, 1.6€, 2.4€, 3.2€, 4€, 4.8€, 5.6€, 6.4€, 7.2€, 8€]
- Negative probability threshold: 0.1 (10%)

7.3.1 Exact case

In the exact case the algorithms converged to the optimal pair of price and bid that is 2€ and 1.6€. In this case we have the convergence only after $\sim$ 300 days and this is reasonable if we think that we will need 100 days to pull each arm at least one time.

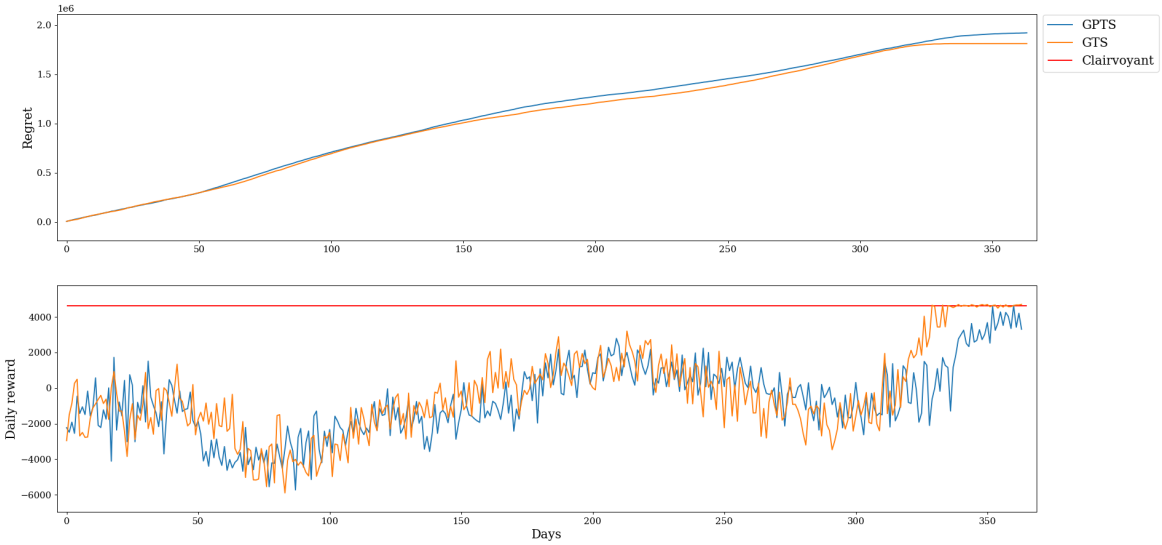

Figure 13: Joint pricing and bidding without approximation results

7.3.2 Approximated case

Also in this case the algorithms converged to the optimal price and bid of 2€ and 1.6€. Here the convergence is very fast and the optimal pair of price and bid reached is the same as in the exact case because the approximation explained in the previous section holds in our scenario.

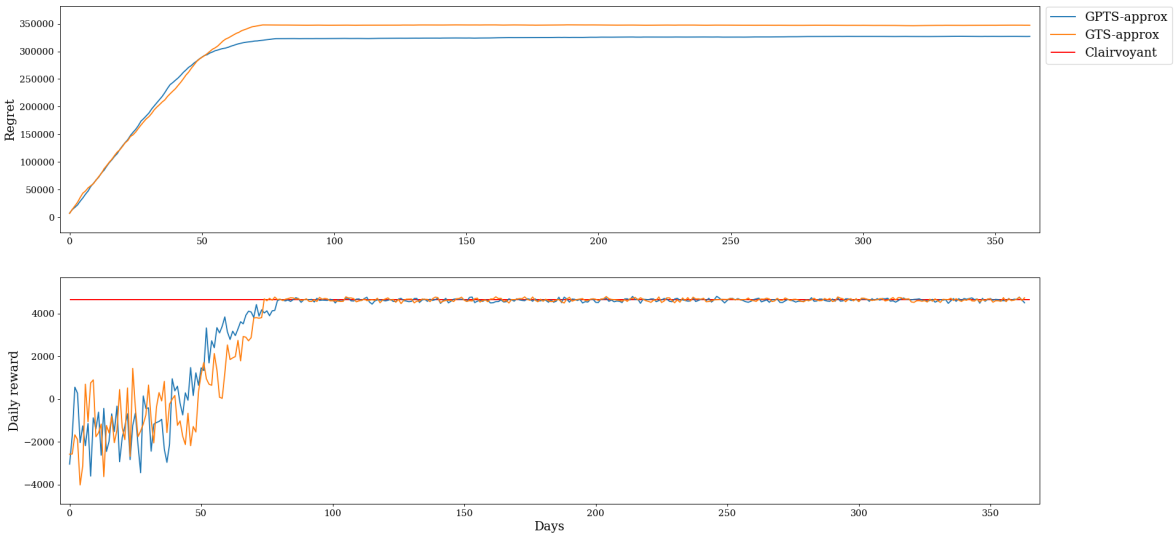

Figure 14: Joint pricing and bidding with approximation results

8 Joint pricing and bidding with discrimination

8.1 Problem description

The problem is similar to the one presented in the previous section with the exception that in this case we have to discriminate over the customer classes for pricing. So we have to find a single bid for the three classes and select a potentially different price for each of them.

8.2 Algorithm

The algorithms are almost the same as the two described for the previous problem. The difference is that in the exact algorithm we must have a context generator for each bid and so the formula for selecting the bid to pull at each time steps (presented in section 6.2) becomes:

$$b_t \leftarrow \arg\max_{b \in bids} \left\{ dc\_s_{b,t} \cdot \left\{ \sum_{c \in C_{b,t}} [w_{c,t} \cdot conv\_rate_{c,t} \cdot p_{c,t} \cdot (1 + \mathbf{E}[returns]_{c,t})] - cpc\_s_{b,t} \right\} \right\}$$

where $C_{b,t}$ is the context structure for bid b and $w_{c,t}$ is the weight of context c calculated as:

$$w_{c,t} = \frac{\#customers_{c,t}}{\#customers_{b,t}}.$$

Here $\#customers_{c,t}$ is the number of customers belonging to the context and $\#customers_{b,t}$ is the total number of customer for the bid b for which the context is generated.
In the approximated case we have a single context generator and so the prices for the three classes are optimized independently from the bids. The formula above for selecting the best bid is used also in this case with the difference that all the bids have the same $C_{b,t}$ because the context structure is unique.

8.3 Results

We have run experiments in the exact and in the approximated case using both the GTS and GPTS combined with greedy and brute-force context generation (the price learner used is always TS). We

decided to use only the incremental context generation since in the previous problem of section 5 the context generation from scratch performed worse. The settings were the following:

- Number of experiments: 5
- Rounds horizon: 365 days
- Returns horizon: 30 days
- Price arms: [0.5€, 1€, 1.5€, 2€, 2.5€, 3€, 3.5€, 4€, 4.5€, 5€]
- Bid arms: [0.8€, 1.6€, 2.4€, 3.2€, 4€, 4.8€, 5.6€, 6.4€, 7.2€, 8€]
- Negative probability threshold: 0.1 (10%)
- Hoeffding confidence: 95%

8.3.1 Exact case

The situation is similar to the one of the previous problem. In 365 days the sample we collect are too few to provide a good result. However at the very end of the time horizon all the algorithms seemed to converge to the optimal prices for all the three classes.

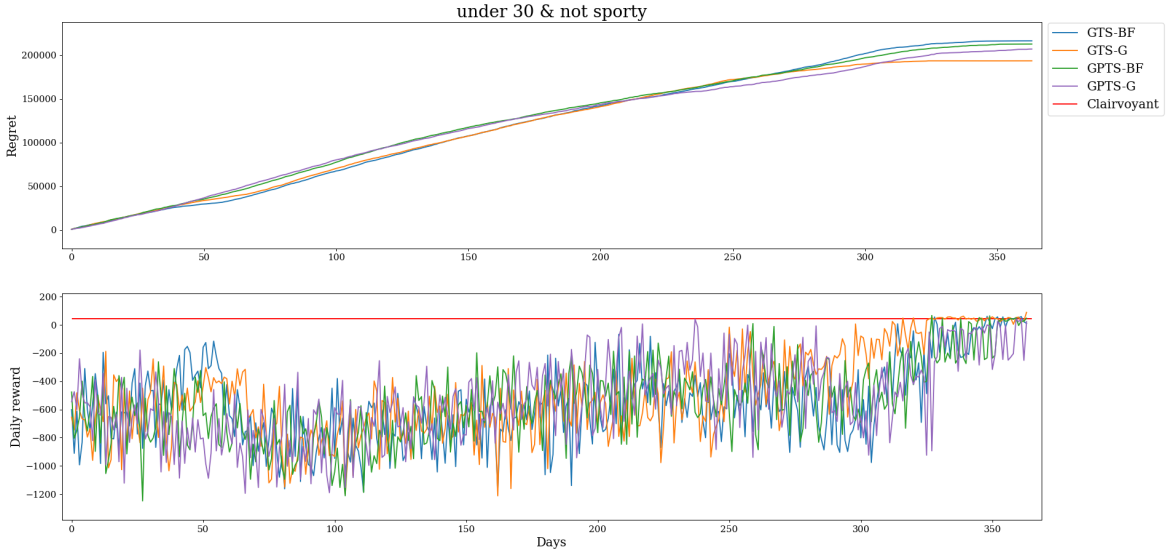

Figure 15: Joint pricing and bidding with incremental context generation results for class *Under 30-Not sporty*

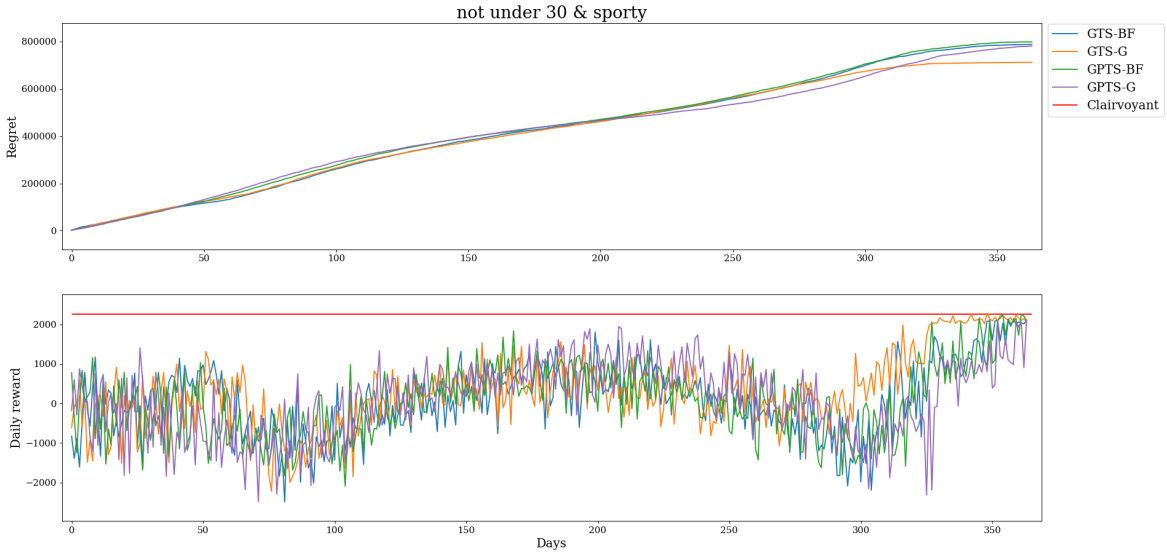

Figure 16: Joint pricing and bidding with incremental context generation results for class *Over 30-Sporty*

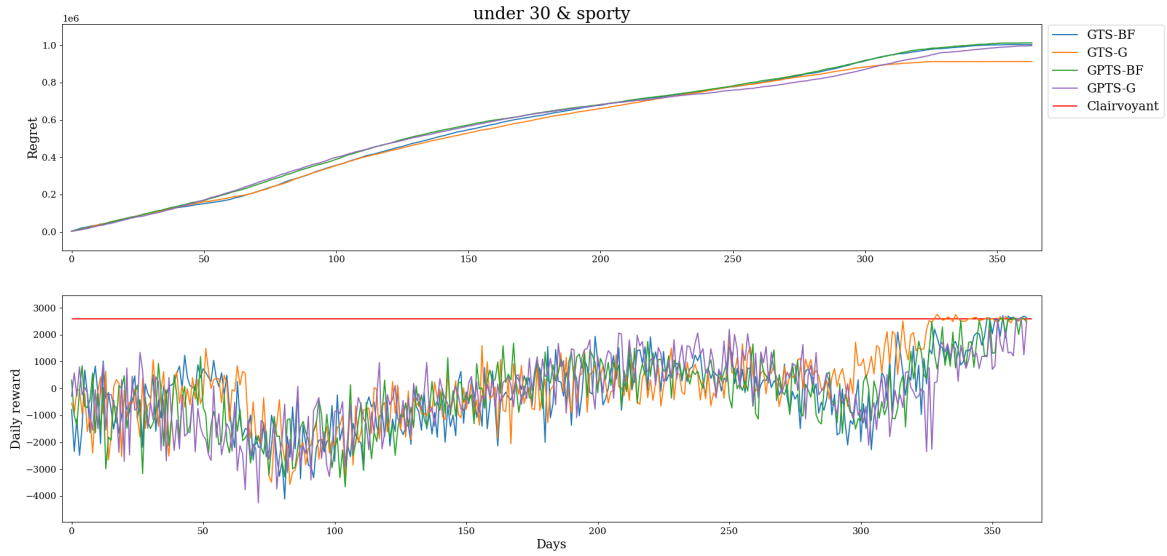

Figure 17: Joint pricing and bidding with incremental context generation results for class *Under 30-Sporty*

8.3.2 Approximated case

In the approximated case the convergence is very faster and all the classes converged to the optimal price and bid in almost all the experiments. As happened also for problem of section 5 some experiment didn't converge for the second class *Under 30-Not sporty*.

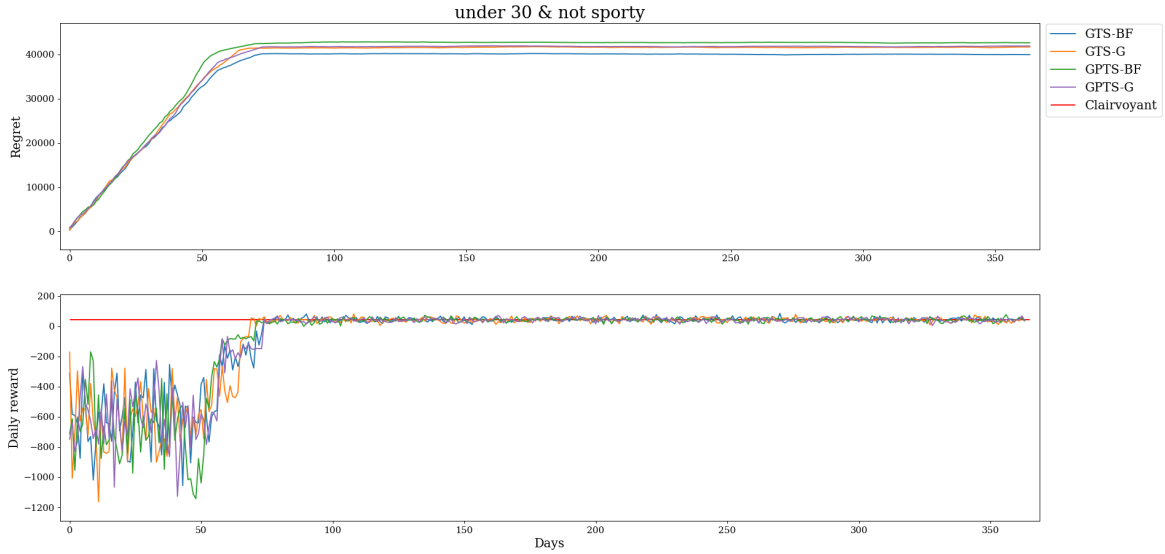

Figure 18: Joint pricing and bidding with incremental context generation results for class *Under 30-Not sporty*

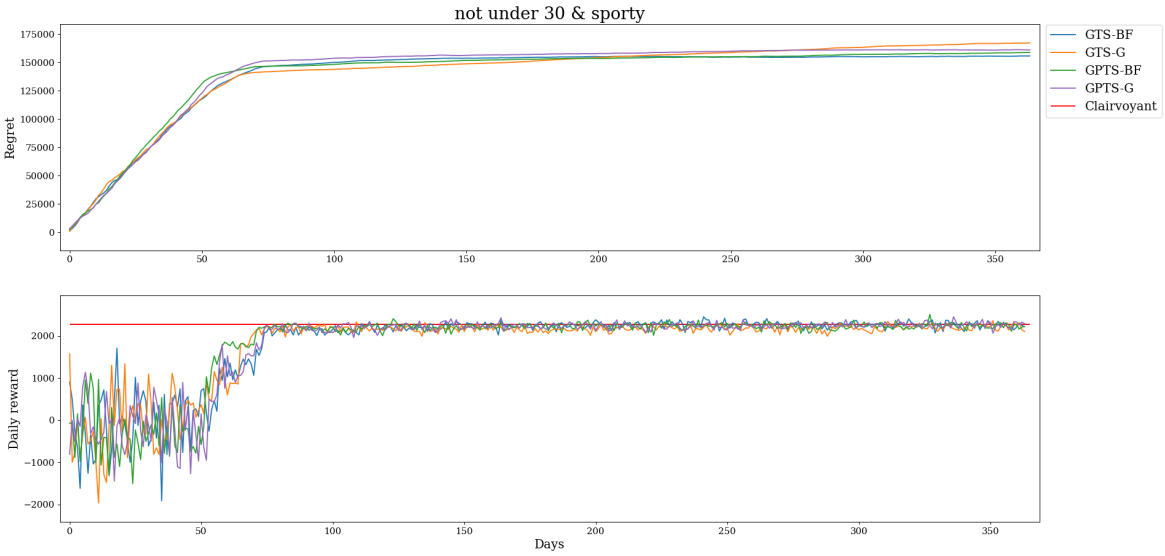

Figure 19: Joint pricing and bidding with incremental context generation results for class *Over 30-Sporty*

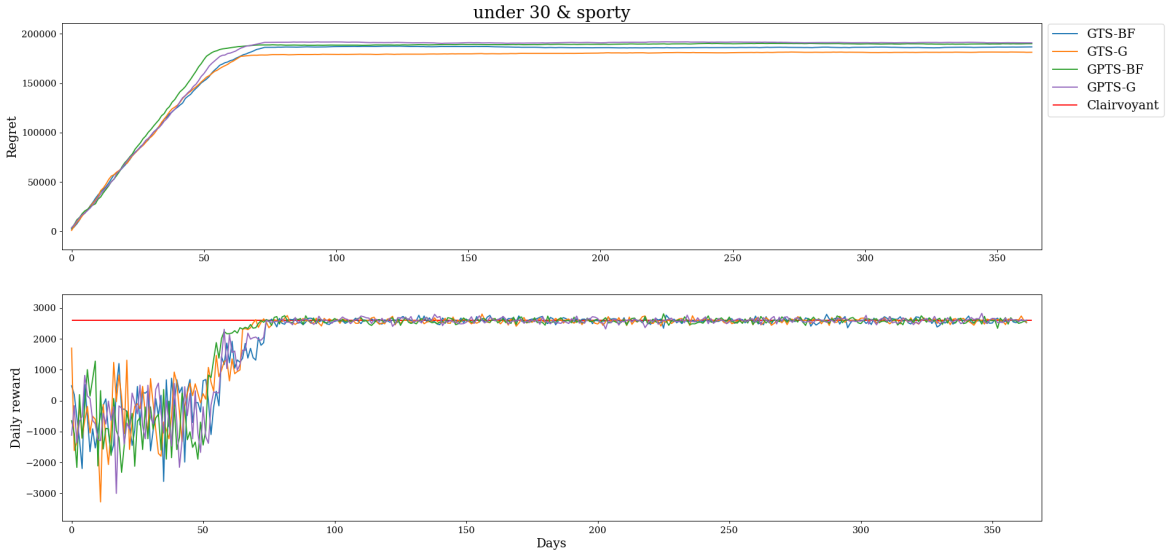

Figure 20: Joint pricing and bidding with incremental context generation results for class *Under 30-Sporty*

9 Running the experiments

All the experiments described in the report can be reproduced by running the file *main.py* and setting the desired parameters between the followings:

Parameter	**Value**	**Default**	**Description**
–exp	{price, bid, joint}		Choose if maximizing price, bid or both.
–scen	SCENARIO		The scenario JSON name located in *main/environments/scenarios*, the one used by us is *scenario_example*.
–disc	{F, T}	F	Choose wether performing or not class discrimination for pricing (default *F*).
–bid	FIXED_BID		Value for the fixed bid, only if –exp is set to *price*. The bid should be one of the bids decided in the scenario.
–price	FIXED_PRICE		Value for the fixed price, only if –exp is set to *bid*. The price should be one of the prices decided in the scenario.
–ne	N_EXP	1	Number of iterations of the experiment to perform, the daily results will be averaged.
–npt	THRESHOLD	0.2	Reward negative probability threshold under which an arm can't be pulled.
–incgen	{F, T}	T	Choose wether the context generation should be incremental or not, only if –disc is set to *T*.
–genrate	RATE		Frequency (in days) for context generation, only if –disc is set to *T*.
–conf	CONFIDENCE	0.05	Confidence for Hoeffding lower bound in context generation, only if –disc is set to *T*.
–approx	{F, T}	T	Choose wether the joint algorithm should be approximated or not, only if –exp is set to *joint*.
–learners	{UCB, TS, GTS, GPTS}		Select the learners to use, UCB and TS can be selected only if –exp is set to *price*; GTS and GPTS can be selected only if –exp is set to *bid* or *joint*. More than one learner can be selected at the same time.
–cgens	{G, BF}		Choose the type of context generation between Greedy and Brute-force or both, only if –disc is set to *T*. More than one generator can be selected at the same time.
–save	{F, T}	F	Choose whether to save or not the results as JSON file.