POLITECNICO
MILANO 1863

# ADVERTISING

Documentation

## DATA INTELLIGENCE APPLICATION

Academic Year 2018/2019


Made by:
Antoniazzi Matteo (895712)
Bonali Luca (896641)
Chittò Pietro (899045)
Lamparelli Andrea (894005)
Ravelli Leonardo (894222)

# Contents

# Chapter 1

# Introduction

## 1.1 Purpose

This document is basically a **report** document, since its purpose is to report and describe the results obtained by several analysis performed in an advertising scenario, in which we had to daily allocate some budgets to some sub-campaigns given a daily budget constraint. In particular the work that we have done follows these steps:

1. Choose a product to advertise through online channels
2. Choose channel over which advertise the product (e.g. search, display and social channels)
3. Define the sub-campaigns in according to the selected channels
4. Identify the classes we will need to address and describe them
5. Define a cumulative daily budget
6. Run different algorithms that provide us which budget we have to assign for every sub-campaign without exceeding the cumulative daily budget
7. Compare the performance of all algorithms and provide some results in according to this scenario

## 1.2 Product

The product we decided to use for this project is the AMAZON ECHO PLUS. We're basing our project on this version of the product with respect to other ones (like Echo Dot, Echo Sub etc.) because, even if the Echo Dot is the more sold product for its small price, it lacks some features like the hub for connecting more smart devices to make a more advanced "smart home".

It's a smart speaker developed by Amazon. Echo devices connect to the voice-controlled intelligent personal assistant service Alexa, which responds to the names "Alexa", "Echo", or "Computer". The features of this device include: voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, in addition to providing weather, traffic and other real-time information. It can also control several smart devices, acting as a home automation hub like, for example, Smart TV, specific appliances, lamps, windows, doors, temperature control etc. It shares design similarities with the first-generation Echo, but also doubles as a smart

home hub, connecting to most common wireless protocols to control connected smart devices within a home.

We consider it more customizable than its direct competitor "Google Home" and Alexa's encourages faster and broader development and support from third-parties of its skills market.

Figure 1.1: The second generation of Amazon Echo Plus was released in September 2018

## 1.3 Overview

This report document is structured in this way:

- **Introduction**: This chapter introduces the purpose of the document, its structure and it starts describing the product we have chosen.
- **Classes and environment description**: In this chapter we will provide a description of the whole scenario that we have considered performing all the analysis, in particular we will describe the features and how we have identified the users' classes, the considered channels and so also the sub-campaigns.
- **Preliminary concepts**: This is a preliminary chapter that introduces main algorithms that are considered as ground truth for the algorithm adopted during the analysis.
- **Aggregate analysis**: This chapter focuses the attention on the first kind of analysis, in which we have considered only one curve per sub-campaign (obtained as aggregation over the three classes defined in previous chapters).
- **Disaggregation and context identification**: This chapter provides results and consideration about a disaggregation analysis, which is performed along with a context identification algorithm.
- **Tools**: This chapter simply provides tools and libraries adopted in order to perform all analysis and to draw up this document.

# Chapter 2

# Classes and environment description

## 2.1 Features selection

We describe our possible customers by means of 3 main features, with the following values:

- *Age*: Young, Adults, Retires
- *Home status*: Living alone, Living with family
- *Welfare*: Normal, Richer

We decided to divide with respect to the age because younger people are more inclined to accept new technologies to make their lives simpler than older ones. We also thought that a person who lives alone is more worn to have something that can help and fasten the way he approaches some of his daily tasks, especially if young. Of course, we consider that this product is a commodity and so, even if the price is not too large, some class of customer may not be interested in buying our product.

## 2.2 Classes description

In the following tables we show how, using the previous explained features, we have created our main class of customers. For readability, we split the 3D features tensor into 2 tables according to the feature home status. In each cell of the table we reported the probability of a user to belong to that specific class.

Each colour represents one class.

| **FAMILY 0.4** | Young 0.5 | Workers 0.4 | Retires 0.1 |
|---|---|---|---|
| Richer 0.55 | 0.11 | 0.088 | 0.022 |
| Normal 0.45 | 0.09 | 0.072 | 0.018 |

| **ALONE 0.6** | Young 0.5 | Workers 0.4 | Retires 0.1 |
|---|---|---|---|
| Richer 0.55 | 0.165 | 0.132 | 0.033 |
| Normal 0.45 | 0.135 | 0.108 | 0.027 |

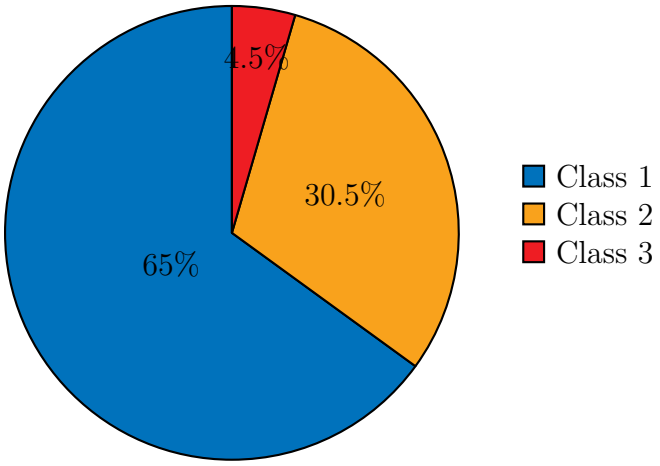



Figure 2.1: Customers classes pie chart

- **Class 1**: this class is characterized by people who live alone, like young student or worker, who have a good economic wellness and are more willing to spend money on this commodity. Young families with economic possibilities are in this class too. We assume that this is the class that we expect to click more on our advertised product.
- **Class 2**: in the second class we have other kinds of families, from younger to older, which are less incline to spend money on a product like that, so they will less probably click on our ad. Considering age , we can include in this class also older families, or retired couples, who have money to spend, and are curious about these new technologies.
- **Class 3**: this class is composed by all those people who, for different reasons, are not so interested in our product, can be for economic reason for example. But advertising the product on those people, may bring some of them to click on the product anyway.

## 2.3 Sub-campaigns definition

- **Search Advertising** This is a method of placing online advertisements on web pages that show results from search engine queries. Through the same search-engine advertising services, ads can also be placed on Web pages with other published content. For this reason we have supposed that these two channels would become the more prolific ones.
  - *Google*
  - *Bing*
- **Display Advertising** This is seen as a disturb many times, so we have assumed that on YouTube, some of the user which are already interested in our product, and knows about it, will click on the ad when it is displayed. But this number will grow very slowly, and it may not be useful regarding the budget cost. This means that we didn't expect to give much importance to this sub-campaign.
  - *Youtube*
- **Social Advertising** This advertising that relies on social information or networks in generating, targeting, and delivering marketing communications. For example, the advertising platforms involve targeting and presenting ads based on relationships articulated on those same services. These two channel represented unknown aspect since we didn't expect anything since their related performance are strictly correlated to the users that populate the specific social networks.
  - *Facebook*
  - *Instagram*

**Google** is the most used search engine, so it seemed mandatory to have our focus on an advertising sub-campaign on it. We assume that all the three classes use Google. **Bing** is the default search engine when dealing with Microsoft OS, and we are assuming that people belonging to the 2nd and 3rd class most probably have a windows based computer, since we assume it to be more user friendly, and that are not interested in changing the default web search engine. The nature of these two search engines brings their curves to be more effective in advertisement, since probably the users go visiting the website to buy some product. Once the invested budget is higher enough the expected number of clicks grows fast, making a sub-campaign on these two channels an interesting choice. For search advertising we are considering keywords like: "assistente vocale", "smart speaker", "home speaker" etc. Searching this kind of keywords will display ads and banners of our product, as form of slot for Google and ads for Bing, that will bring the user to the Amazon link to buy our product. **YouTube** is used for watching videos, and ads are displayed during the video as banners and under the video. Sometimes the ads are related to the video the user is watching, and sometimes it depends on the collected info of that user, like cookies. By displaying our product in some videos as banner, for example during tech videos, which are related to our assistant, we can assume that some user will click on the ad. **Facebook** and **Instagram** are two of the most used social networks. By displaying sponsored ads in the people feed we can increase people's awareness in our product and make them interested in it. We have chosen

YouTube and Instagram, which are more probably used by the first two classes, and Facebook, which is used also by people belonging to the 3rd class.

## 2.4 Average daily budgets/clicks curves

In this chapter we will show the assumption we have made to construct the curves which represent the functions daily budget/clicks.

- *X axis*: daily budget
- *Y axis*: expected number of clicks

We show all the three different curves, for each class of user, and for each sub-campaign, then the aggregated curve for each sub-campaign. The actual numbers on the graphs are our assumptions with respect to the actual behaviour of the users.

We have three levels of clicks limits:

- *High*: 90 clicks
- *Medium*: 60 clicks
- *Low*: 40 clicks

We expect that when a curve has its maximum arrives at one of these three levels, also because being the mechanism to decide which product is placed on the banner, or slot, or ad, an auction ,we have that increasing the budget we increase the probability that our product will be displayed in the best position. But once the product is displayed in the best position the expected number of clicks will remain the same even increasing the budget. (we are implicitly assuming that the bidding is performed automatically by the advertising platform, according to the budget)

The aggregation is achieved by averaging the curves with respect to the probability of belonging to one class (0.65 1st , 0.305 2nd, 0.045 3rd).

- **Google class 1**: This represent the most prolific curve (fig. 2.2), in which we have the most interested users where we expect to have the most clicks, google. The curve grows fast starting with low values of daily budget, and arrives early near to its maximum value, which is the highest level of click we have assumed for the platform, once the product is displayed in the first slot (best position).

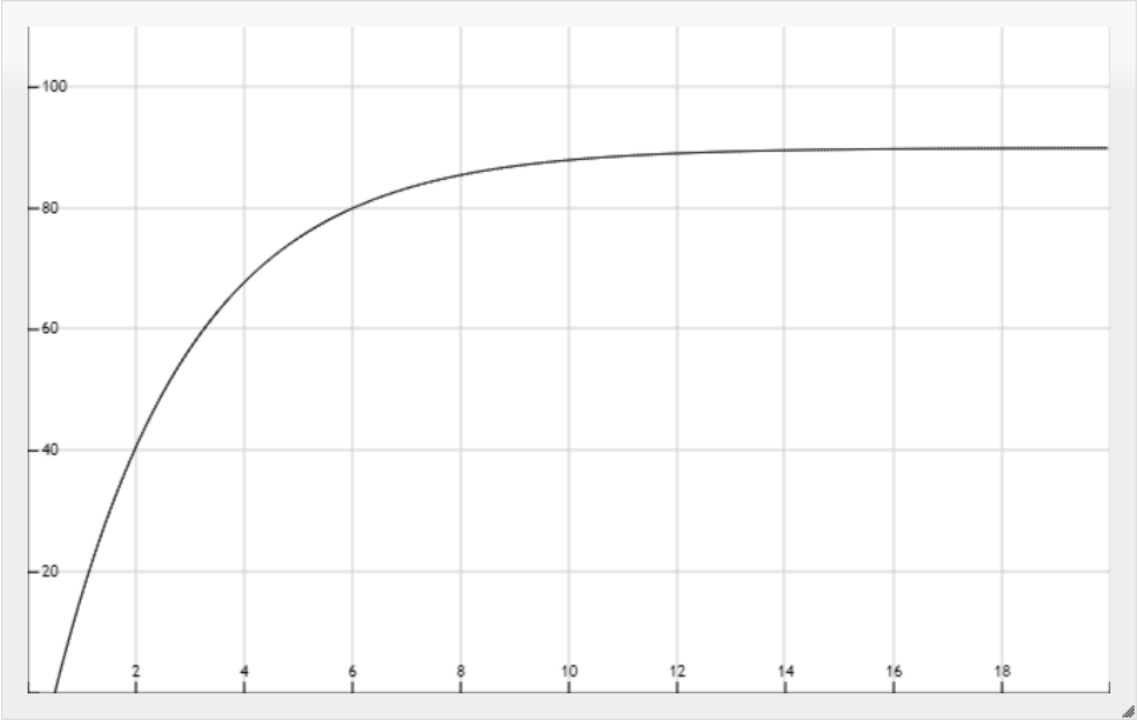



Figure 2.2: 1st users' class of Google channel

- **Google class 2**: Second class of the google users (fig. 2.3), we have a maximum value of clicks which is our Medium value, because we expect this class to be less interested in our advertisement. The curve also has a less increasing behaviour.

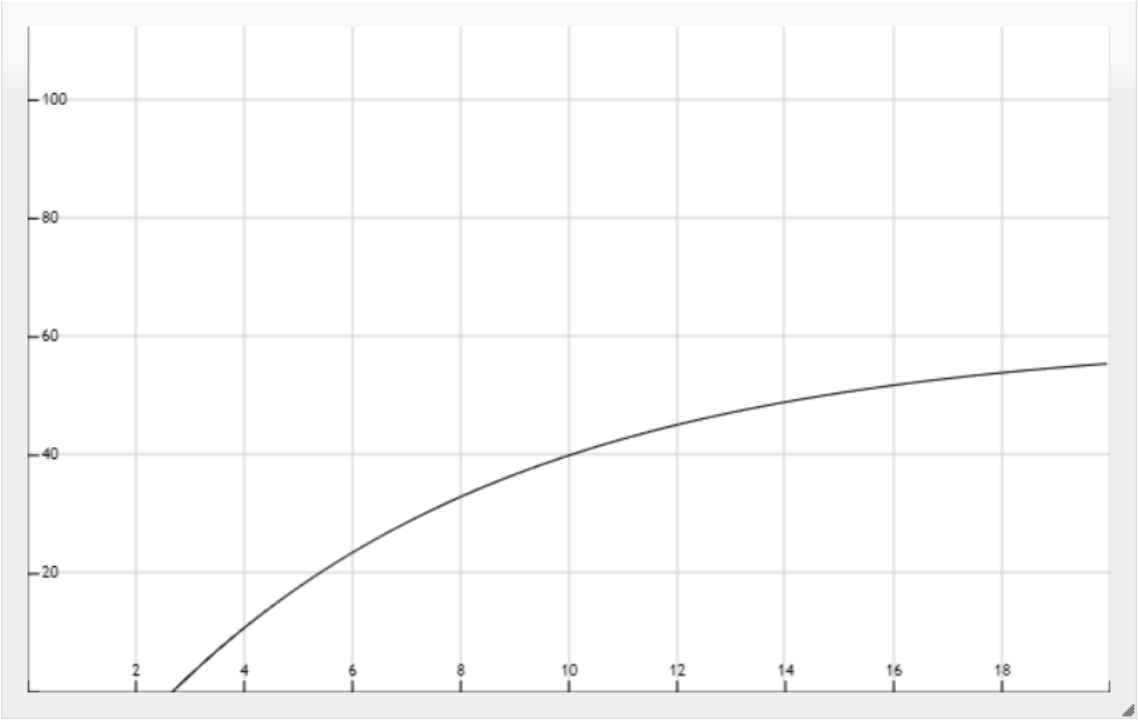



Figure 2.3: 2nd users' class of Google channel

- **Google class 3**: The third class curve ((fig. 2.4)) has a slower increasing behaviour, since this is the class which we assume not so interested in our product. So the limit is the lower level of our definition.

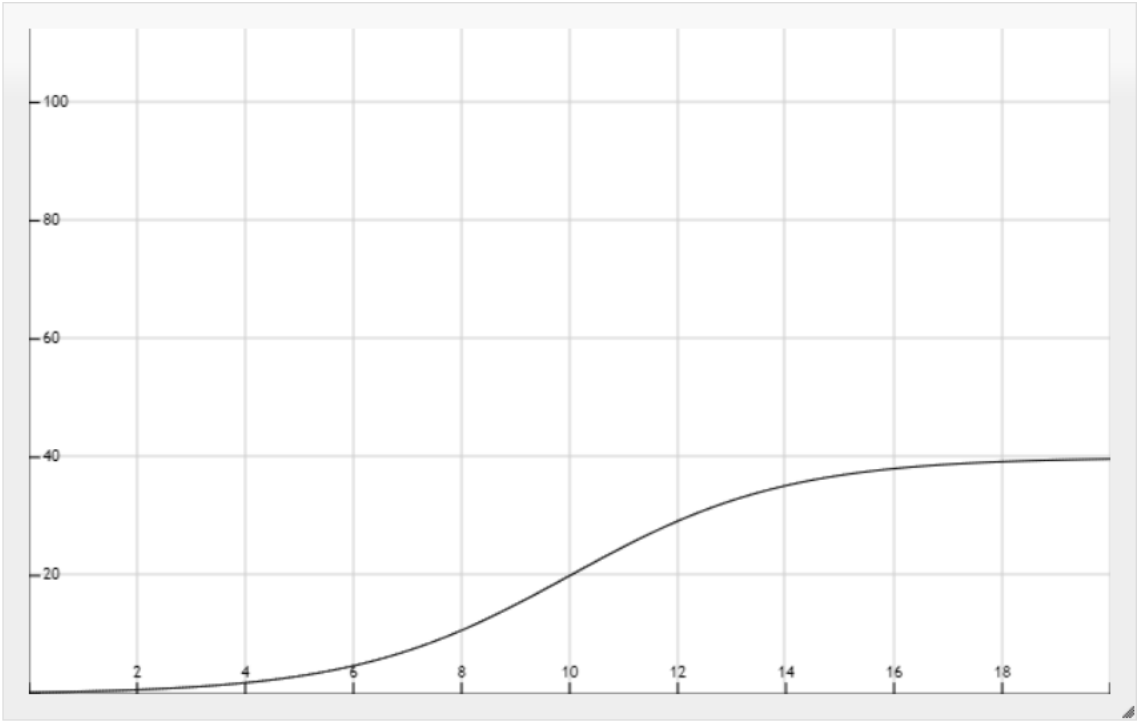



Figure 2.4: 3rd users' class of Google channel

- **Google aggregate curve**: The aggregated curve (fig. 2.5) has the maximum value near 80 and grows quite fast, since the first curve has a bigger influence on the aggregation.

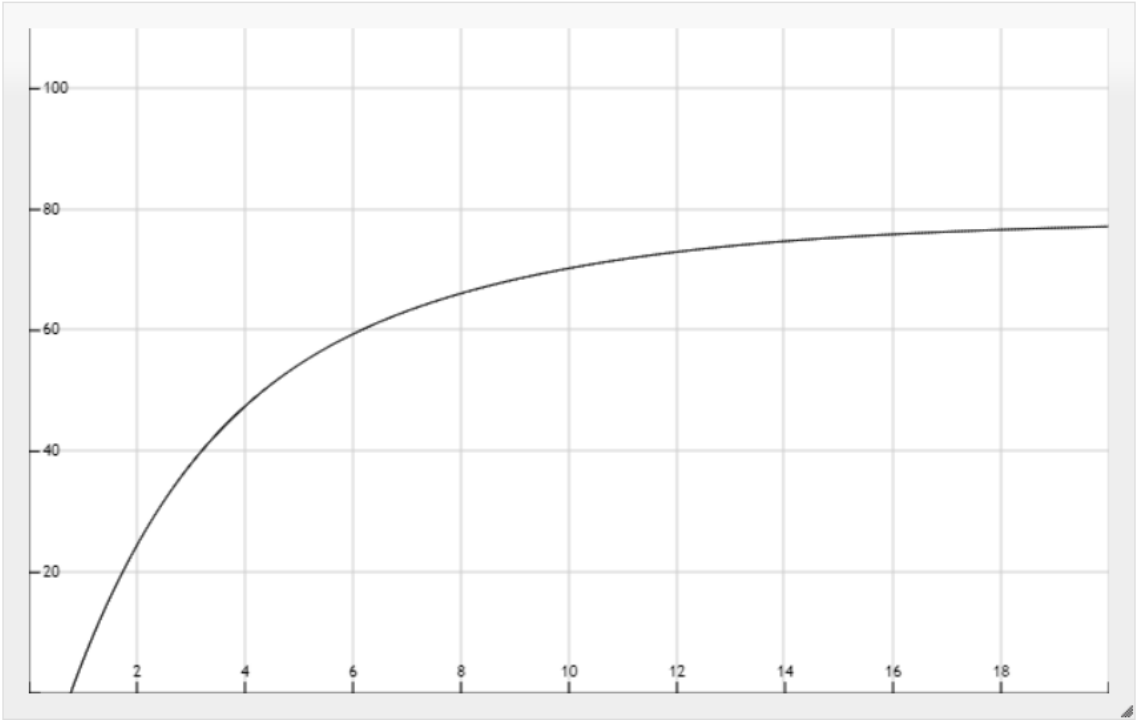



Figure 2.5: Google channel users' aggregate curve

- **Facebook class 1**: First class curve of the second sub-campaign (fig. 2.6), we are assuming that the maximum is lower with respect to the first sub-campaign, and that also grows slower.

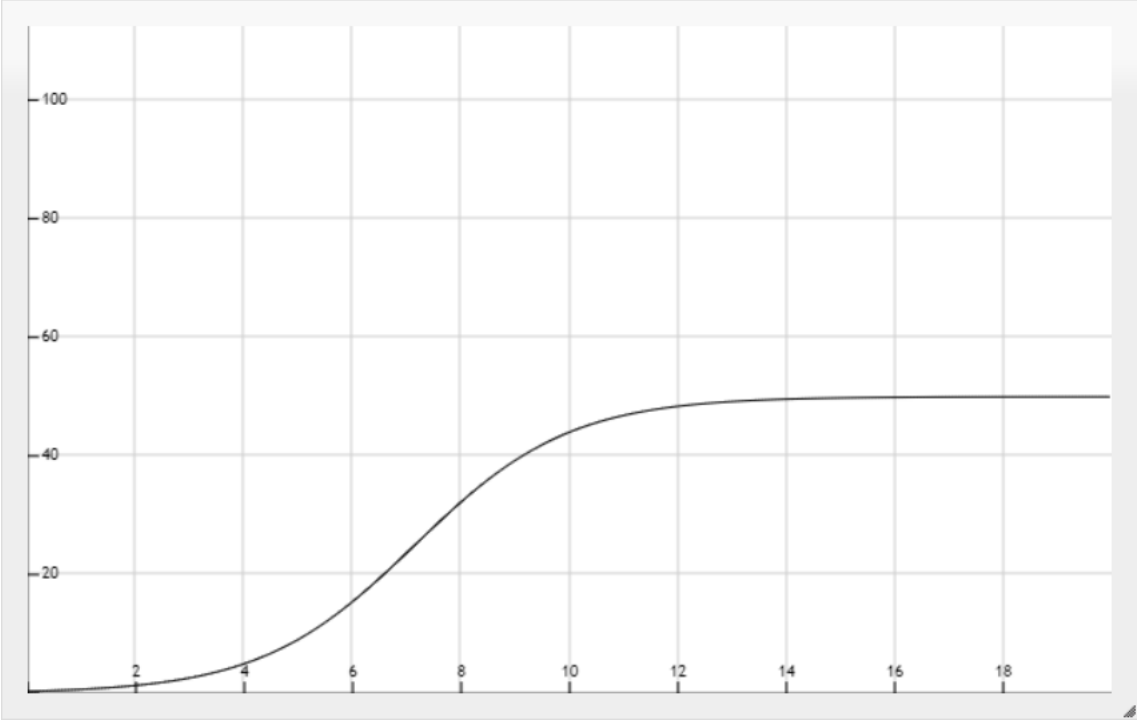



Figure 2.6: 1st users' class of Facebook channel

- **Facebook class 2**: This curve (fig. 2.7) has an higher maximum value than the first class one, basing on our assumption that the 2 class of users are more active on Facebook, and they are more inclined to click on our ad on this platform.

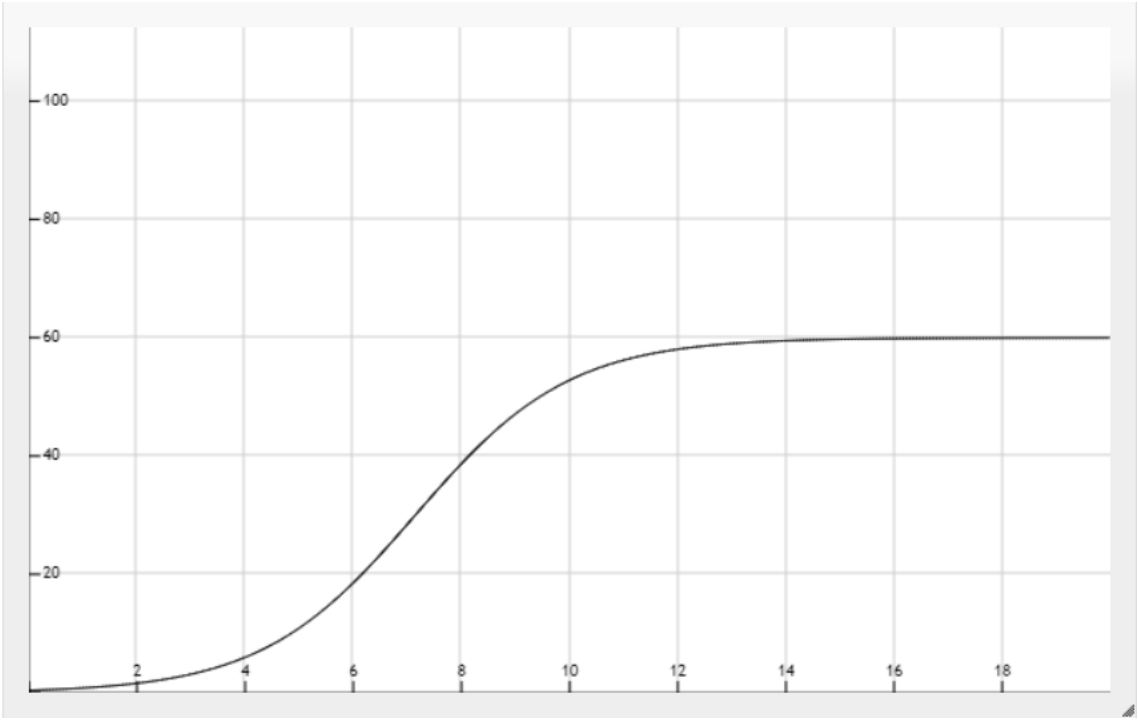



Figure 2.7: 2nd users' class of Facebook channel

- **Facebook class 3**: Third class of user on Facebook (fig. 2.8) are not so interested in our product, and so less likely they will click on our ad.

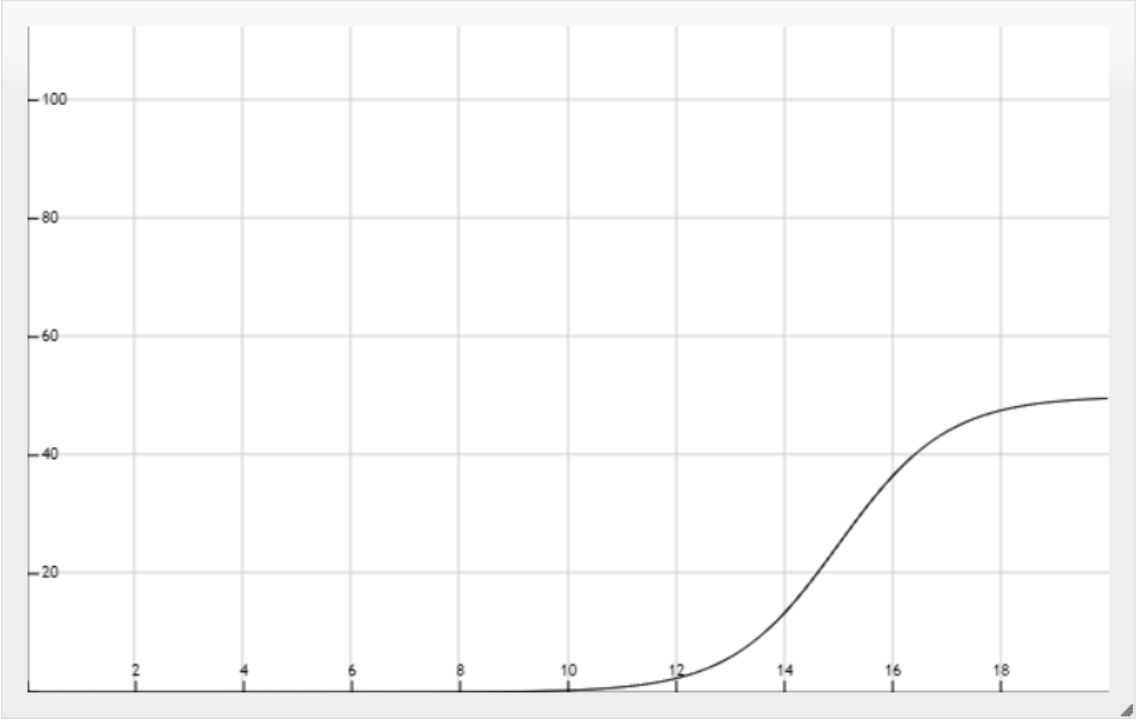



Figure 2.8: 3rd users' class of Facebook channel

- **Facebook aggregate curve**: The aggregation of the three Facebook curves ((fig. 2.9)) is less interesting with respect to the Google aggregation, where we expect to have an higher number of clicks. It may be not interesting in general because the growth is quite slow.

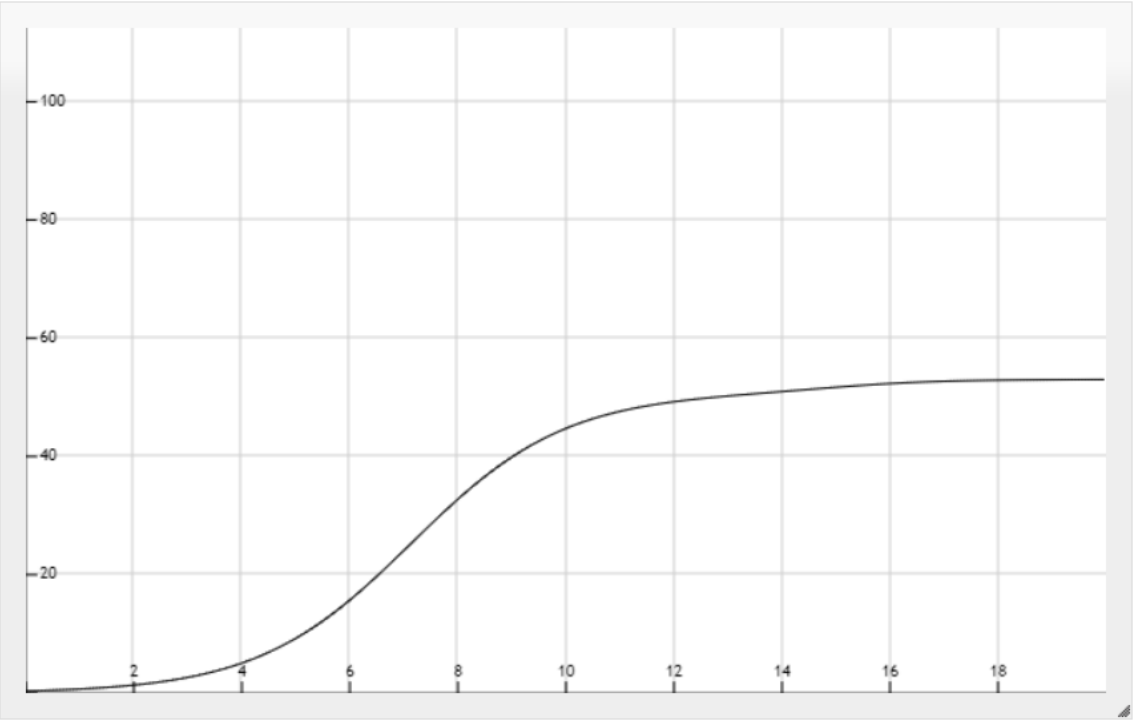



Figure 2.9: Facebook channel users' aggregate curve

- **Instagram class 1**: Instagram first class (fig. 2.10) has a very interesting behaviour, it grows fast with respect to the budget, and its maximum value is high. Moreover a lot of Instagram users belong to our identified first class, and they are more inclined to click on our product.

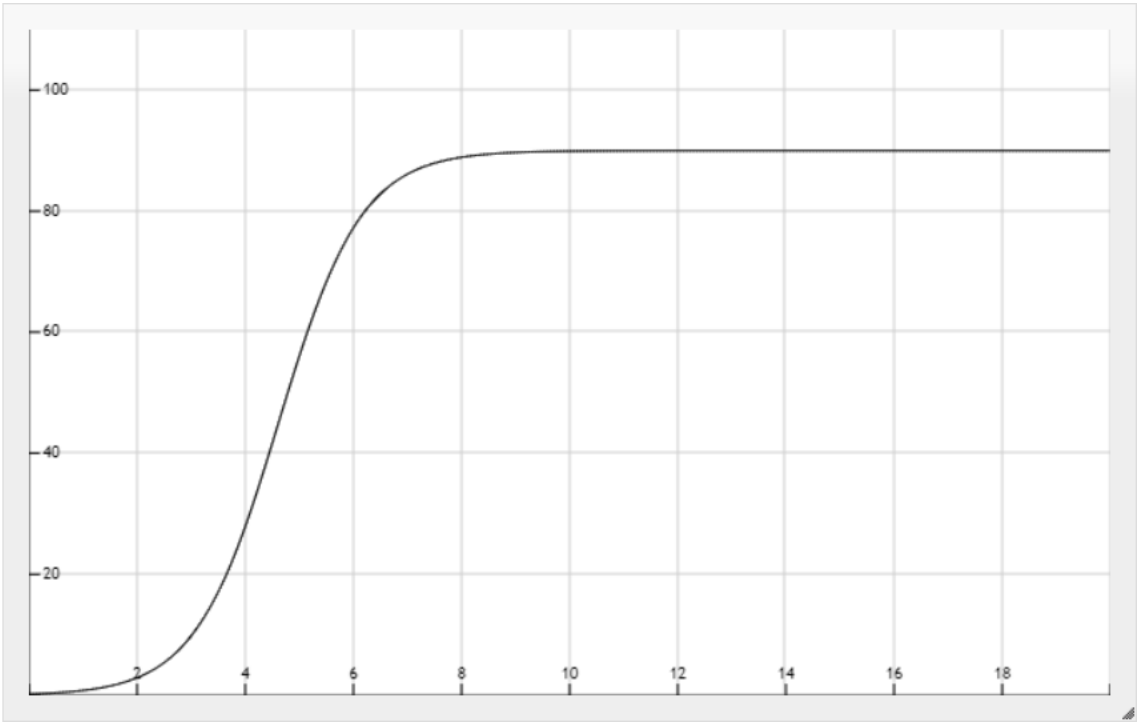



Figure 2.10: 1st users' class of Facebook channel

- **Instagram class 2**: The second class of users on Instagram (fig. 2.11) has a slower growth, regarding budget, but has an interesting behaviour if the ad wins more auction, and it is displayed more times.

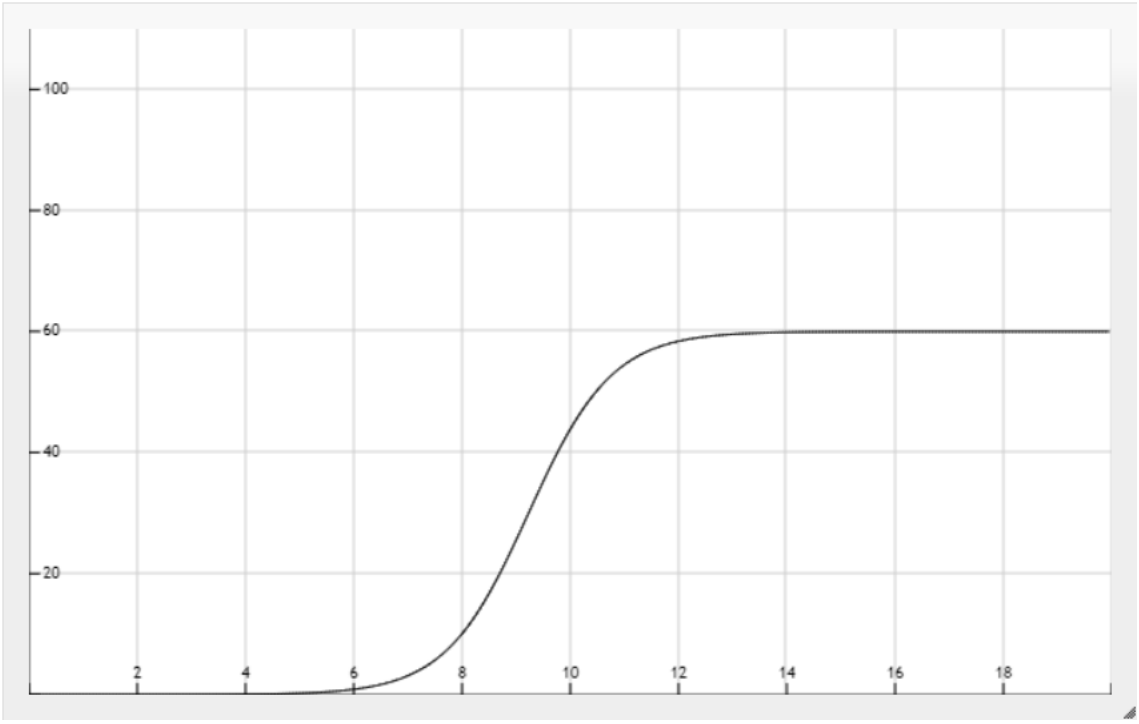



Figure 2.11: 2nd users' class of Facebook channel

- **Instagram class 3**: Instagram is not so used by people belonging to our third class (fig. 2.12), and the curve is very low, but if aggregated can bring to some advantages.

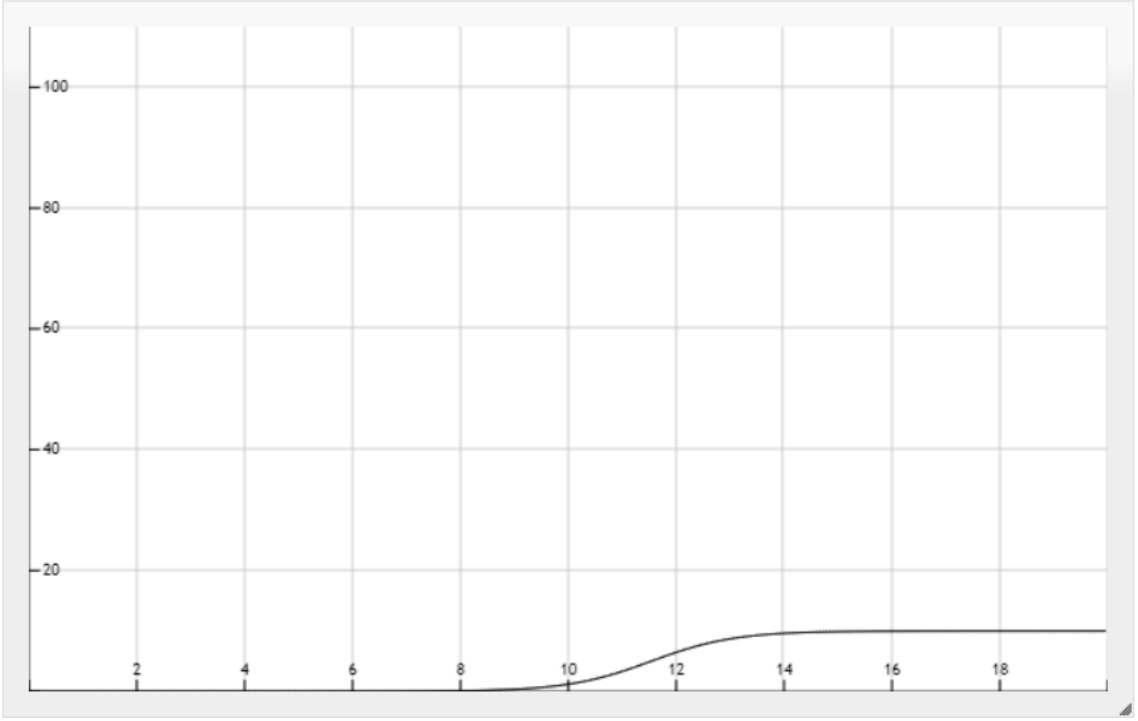



Figure 2.12: 3rd users' class of Instagram channel

- **Instagram aggregate curve**: Aggregating the Instagram curves (fig. 2.13) leads to a interesting result, grows quickly, and has a high maximum value. Also considering the third class of user, advertise our product to a broader audience, which is an advantage.

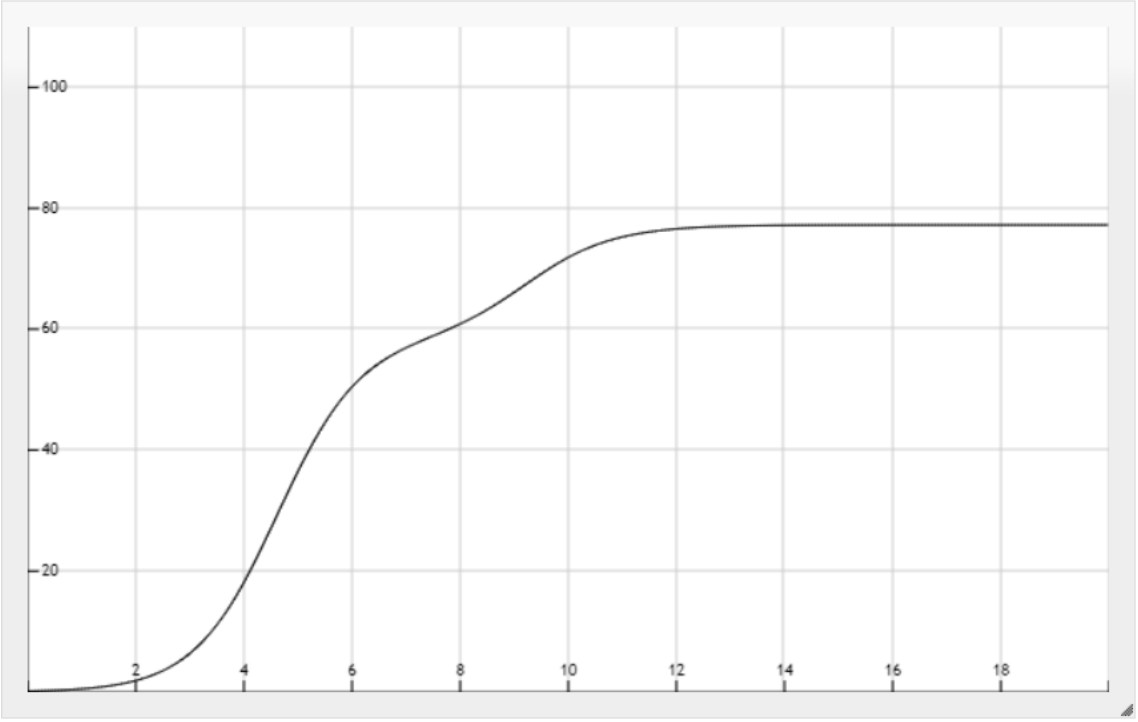



Figure 2.13: Instagram channel users' aggregate curve

- **Youtube class 1**: This channel, as expected, provides us bad results as shown in figure 2.14.

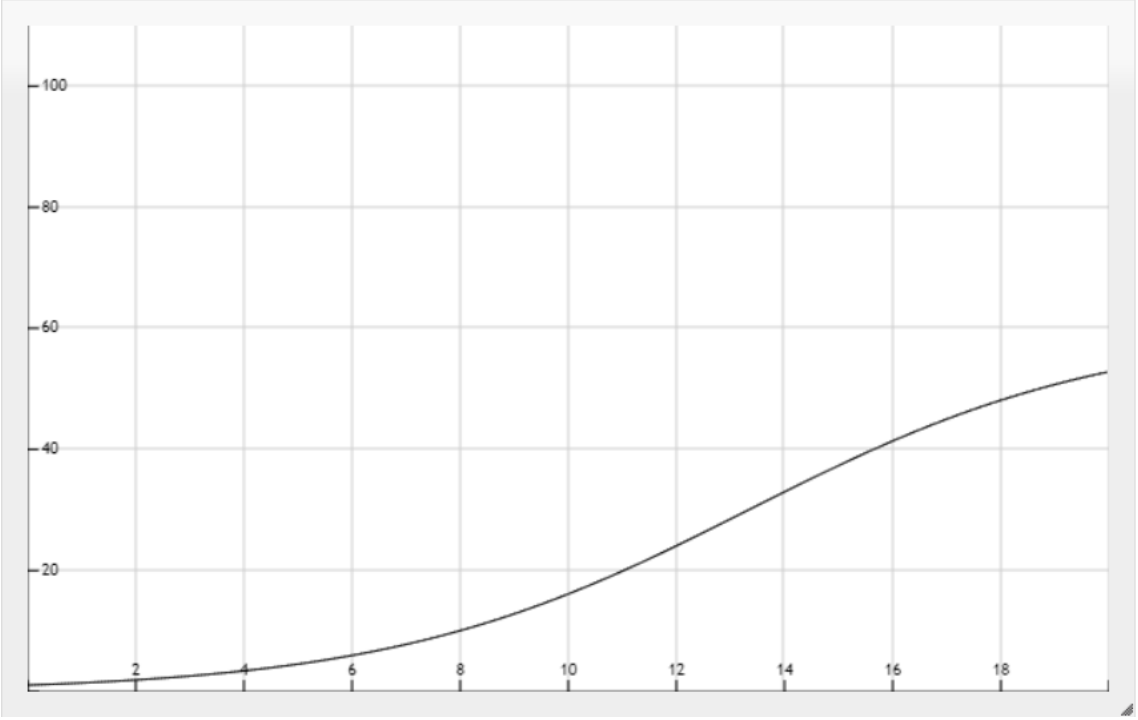



Figure 2.14: 1st users' class of Youtube channel

- **Youtube class 2**: Also considering the second class of users, we had bad results as shown in figure 2.15.

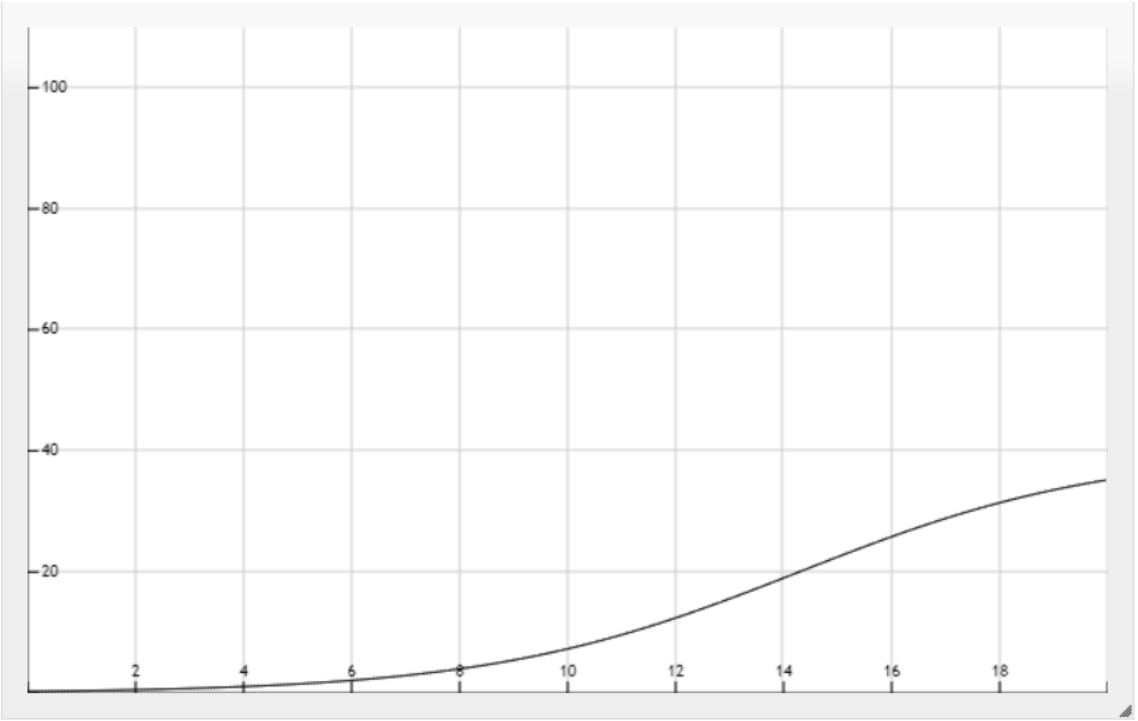



Figure 2.15: 2nd users' class of Youtube channel

- **Youtube class 3**: This 3rd class of user provide the worst result considering the whole scenario (fig. 2.16).

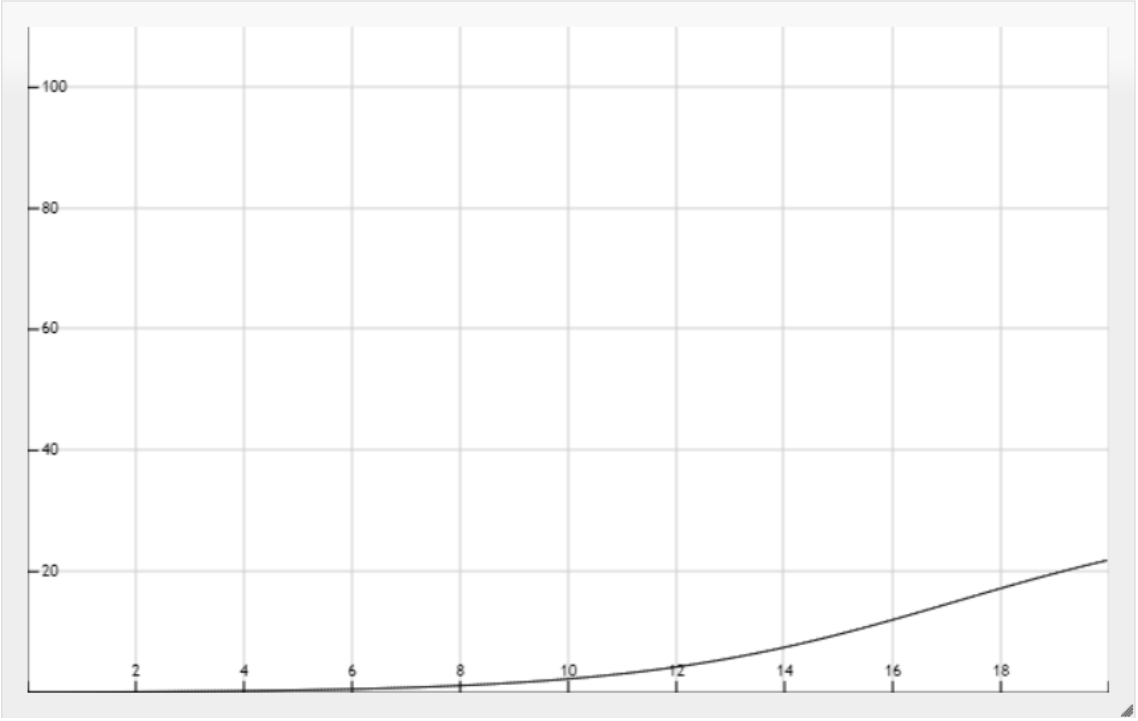



Figure 2.16: 3rd users' class of Youtube channel

- **Youtube aggregate curve**: Using the aggregated curve ((fig. 2.17)) lead to a poor result, we have medium low level clicks with high cost of budget

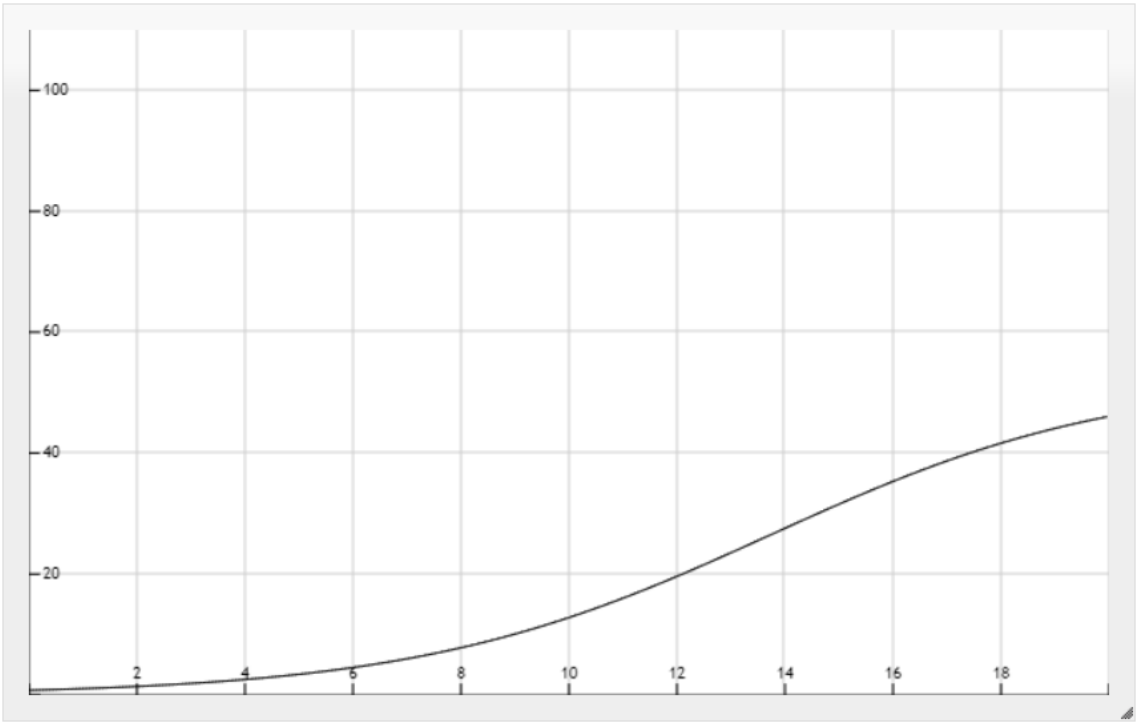



Figure 2.17: Youtube channel users' aggregate curve

- **Bing class 1**: As you can notice in figure 2.18 the curve stays on zero until some point (approximately 4€) where it has a sudden surge until it reaches its limit.

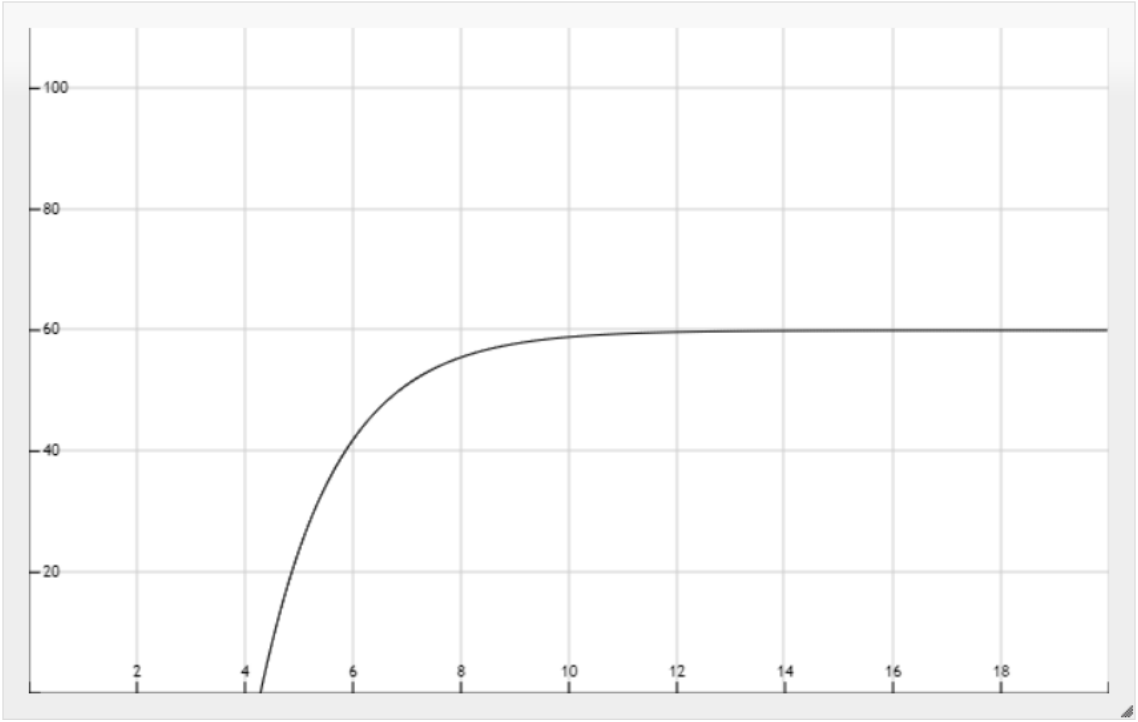



Figure 2.18: 1st users' class of Bing channel

- **Bing class 2**: For the second class (fig. 2.19) we have a quick increase in the function, since more user of the second class will click on the product on this platform, and the maximum number is the same as the first one.

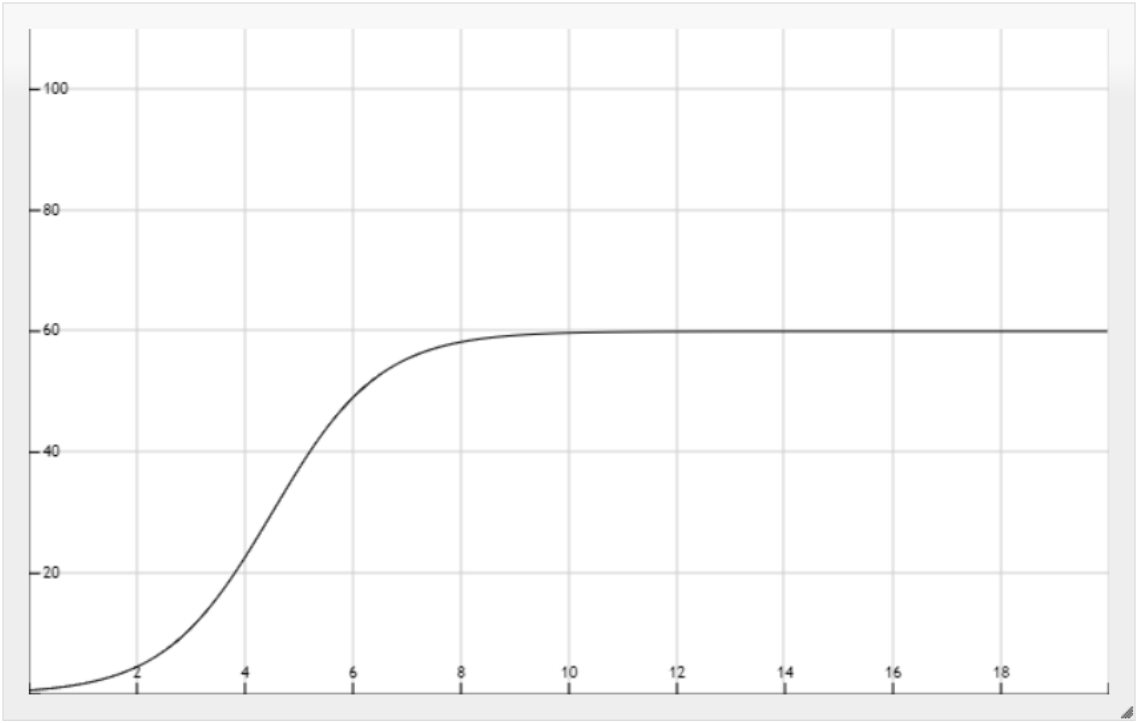



Figure 2.19: 2nd users' class of Bing channel

- **Bing class 3**: In the third curve (fig. 2.20) there is less interest in the product, and the curve grows slowly.

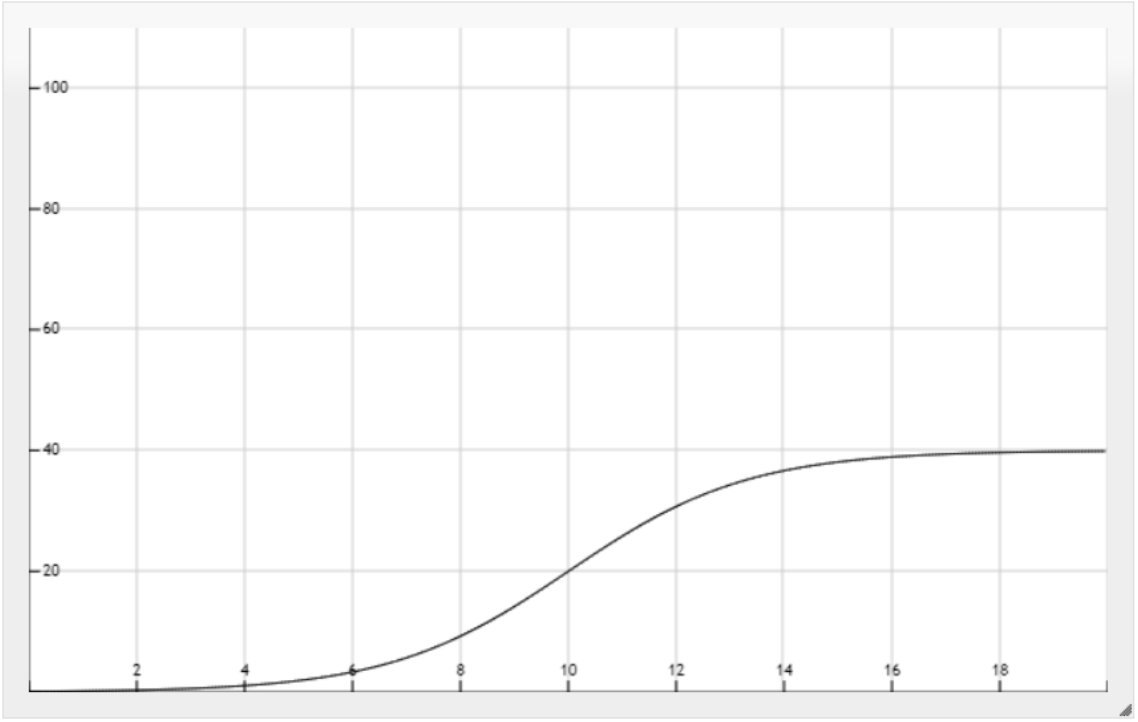



Figure 2.20: 3rd users' class of Bing channel

- **Bing aggregate curve**: Once aggregating the curves (fig. 2.21) we have that in order to get a good number of clicks, we would need to spend budget on this sub- campaign, since the curve grows very quickly and with a considerable maximum amount of expected number of clicks.

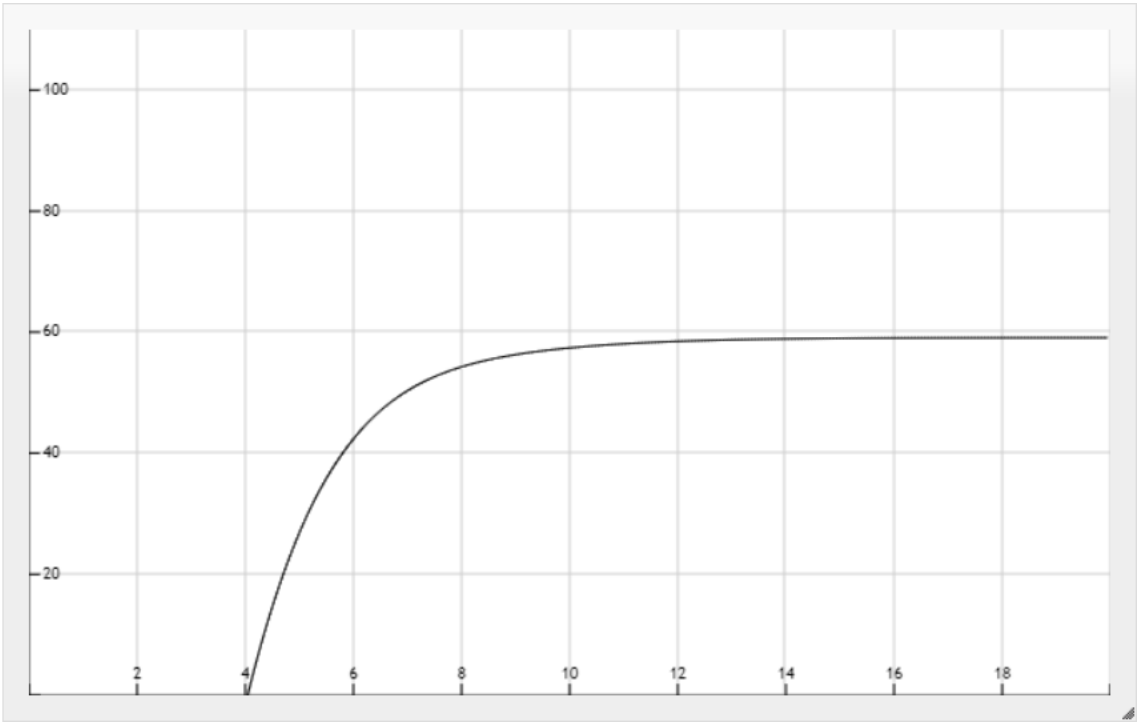



Figure 2.21: Bing channel users' aggregate curve

# Chapter 3

# Preliminary concepts

In this chapter we provide all preliminary tools that we have used for performing all the analysis provided in the following chapters, in particular we provide a theoretical description of adopted algorithms in order to provide a ground truth that allows the reader to well understand what is behind all algorithms that are deeper described in the specific analysis. From this point on we will refer to all sub-campaigns in the order in which are presented in section 2.3.

## 3.1 Thompson Sampling

The *Thompson Sampling* is a heuristic for choosing actions that addresses the exploration-exploitation dilemma in the multi-armed bandit problem. It consists in choosing the action that maximizes the expected reward with respect to a randomly drawn belief. One of the most simple Thompson Sampling implementation sample from the environment a Bernoulli distribution and has as prior a Beta distribution. Once an arm is played the update only affects the Beta distribution of that arm. The pseudo-code of how the TS works (considering Bernoulli and Beta distribution) is the following:

1. at every time $t$, for every arm $a$

$$\widetilde{\theta_a} \leftarrow Sample(P(\mu_a = \theta_a))$$

2. at every time $t$, play arm $a_t$ such that

$$a_t \leftarrow arg\max_{a \in A}\{\widetilde{\theta_a}\}$$

3. update the Beta distribution of arm $a_t$ as

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

## 3.2 Gaussian Process Thompson Sampling

A Gaussian Process Thompson Sampling is quite similar to the classical Thompson Sampling the only difference is the instead of having a prior distribution for each arm, here we have a Gaussian Process that is used to learn the whole associated curve. As you can see in the following pseudo-code is very similar to the previous

one, where instead of sampling from a Beta distribution, here we sample from the GP and the the update is performed inside the GP. A crucial point is that here once we have played an arm and then collected its reward from the environment, the update does not affect only that arm but the GP in the neighbourhood of the played arm.

1. at every time $t$, for every arm $a$
$$\widetilde{\theta_a} \leftarrow Sample(P(\mu_a = \theta_a))$$
2. at every time $t$, play arm $a_t$ such that
$$a_t \leftarrow arg\max_{a \in A}\{\widetilde{\theta_a}\}$$
3. update the GP according to the observed reward

## 3.3 Combinatorial algorithm

Due to the fact that we have different learners since we have different sub-campaigns, and in according to how Thompson Sampling learners work we have to select an arm for each of them, so we need an algorithm that allowed us to pick an arm for each learner when there exist a constraint on the arms (in this case budget constraint). this algorithm is called *Combinatorial Algorithm* which is used, in this particular scenario, to select which budget we have to choose in order to maximize our global expected number of clicks (considering all sub-campaigns) and such that the whole budget does not exceed the cumulative daily budget that we have. This problem problem is an extension of the well known knapsack problem, the only difference is that, here, we have to select exactly one item (i.e. budget) for each sub-campaign (i.e. five items) such that the sum of all selected budgets does not exceed the cumulative daily budget and such that the total amount of expected number of clicks is maximized. The combinatorial algorithm works as follow:

1. for each learner (each of one represent a single sub-campaign) compute a sample for each arm (budget), the result is a matrix $n \times m$ where n is the number of sub-campaigns and m is the number of considered budgets. A generic element in position (i,j) represent the sampled expected number of clicks for sub-campaign i with budget j. Call this matrix *sample matrix*
2. create an empty matrix with the same size of the previous one, initialize the first row with the first row of the *sample matrix*, then incrementally fill the row i as follow, for each budget j compute which is the highest expected number of clicks that we have considering all partition of budget j using the i row of the *sample matrix* and the (i-1) row of the current matrix. This step is performed keeping track for each cell which is the partition that provides the highest number of clicks.
3. the result of the previous step is new matrix of size $n \times m$ where a generic row i provides for each budget which is the expected number of clicks considering all sub-campaigns from the first to the $i^{th}$. Now consider the last row (since we want to include all sub-campaigns) and choose the cell with the highest value, from this retrieve the partition of this last sub-campaign and all others. Then return these array of partitions (one for each sub-campaign)

# Chapter 4

# Aggregate analysis

This chapter focuses the attention on the application and analysis of algorithms applied to the same environment but with different settings, trying to compare them in order to provide some conclusion about their performance in our scenario.

## 4.1 Environment setup

In this case the environment is composed by 5 different sub-campaigns, already defined in section 2.3, since initially we have considered that for each channel we don't know which are the three classes to which we have to advertise our product, hence each sub-campaign point to a unique class that is made by an aggregation of the three classes (introduced in section 2.2), which we don't know. The goal of the following analysis is to apply different variation of Thompson Sampling algorithm in order to create a model that is able to predict the behaviour of the different curves (one for each channel) such that it provides us how we have to split the total budget among all sub-campaigns. For each algorithm we analyse how the reward (in terms of number of clicks perceived from a modeled environment) evolves during time, we also provide how the regret evolves (the regret is computed supposing to know which is the optimum, that we can compute since we have the true curves) and the we provide additional information about the single learners, such as how the learners learn their curves and which are the errors generated.

In particular we have analysed two different algorithms: Combinatorial Gaussian Process Thompson Sampling and Combinatorial Gaussian Thompson Sampling, the former is tried in different settings.

## 4.2 CGP-TS (disallow empty budgets)

### 4.2.1 Implementation

In according to how a GP-TS works (defined in section 3.2) and how the combinatorial algorithm was implemented (defined in section 3.3), this Combinatorial GP-TS launches five GP-TS in parallel (one for each sub-campaign), where at each day for every GP-TS instead of choosing the arm the maximize the reward, we have to choose the arm that maximize the overall reward (obtained as sum over all GP-TS) such that the sum of all selected arms (among all GP-TS) does not exceed the budget that we have available. the pseudo-code is the following:

For every day $t$:

- for every sub-campaign $s$ and for every arm $a$ sample a value from the GP associated to that sub-campaign.

$$\widetilde{\theta_{a,s}} \leftarrow Sample(P(\mu_{a,s} = \theta_{a,s}))$$

- for every sub-campaign $s$, play the arm $a_t$ such that it is the best arm obtained solving the combinatorial problem, forcing to have at least 1 budget for every sub-campaign.

$$a_t \leftarrow arg\max_{a \in A}\{\sum_{s \in S} \widetilde{\theta_{a,s}}\}$$

subject to:

$$\sum_{s \in S} \widetilde{\theta_{a_t,s}} < Budget$$

- for every sub-campaign $s$, update the GP in according to the observed reward

In particular the settings of this algorithm are represented in the table 4.1:

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Time horizon | 100 days | Number of days for which the algorithm runs |
| Number of experiments | 100 | Number of times that we have repeated the algorithm in order to have more accurate results |
| Number of sub-campaigns | 5 | since we were considering, for each channel the aggregation of the three classes, so we have one sub-campaign per channel |
| Cumulative daily budget | 20€ | this represent the total amount of budget available among all sub-campaigns, for each day |
| Sigma | 5.0 | Standard deviation used sampling from the environment |
| Alpha | 100 | GP parameter, value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. |
| Allow empty budgets | false | boolean value that specifies whether allowing to have sub-campaigns with 0 budget associated or not |

Table 4.1: CGPTS settings

### 4.2.2 Performance

In this section we provide the results obtained launching the previously described algorithm as the time evolves. In according to the scenario previously described, hence forcing the algorithm to give at least 1€-budget for every sub-campaigns, we initially provide the optimum values in terms number of clicks and sub-campaigns partition obtained by the clairvoyant algorithm. The best partition was computed solving the combinatorial algorithm using the true values rather than the sampled ones, the optimum results are the following:

Number of clicks obtained choosing the best partition: **149 clicks**

Best partition:

- **Google** = 4 €
- **Facebook** = 1 €
- **Instagram** = 6 €
- **Youtube** = 1 €
- **Bing** = 7 €

**Reward analysis** As you can notice in the figure 4.1 the Combinatorial GP-TS is able to reach its maximum value in few days (approximately 20 days), after these 20 days the algorithm stabilizes to the maximum, which is approximately around 137 clicks.

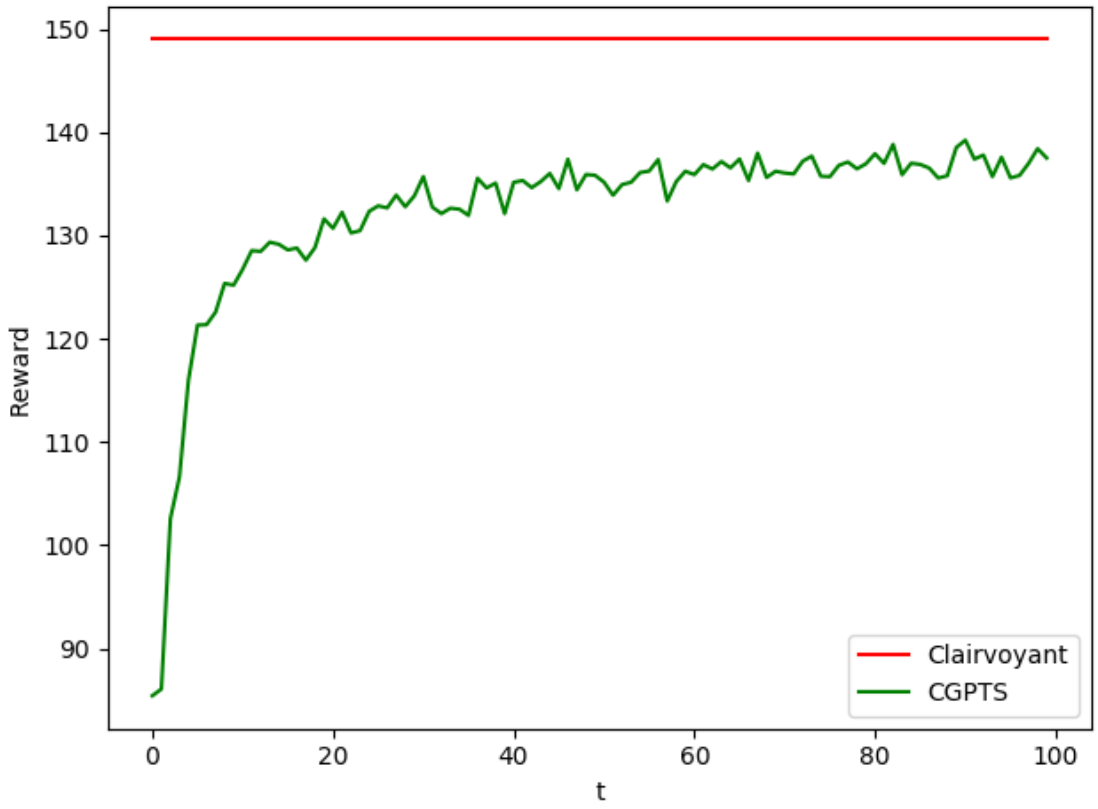



Figure 4.1: Comparison between the reward obtained from the CGPTS and the optimum of clairvoyant algorithm

**Regret analysis** In according to the reward analysis, well described in figure 4.1, the algorithm is not able to reach the optimum partition, this means that, since the algorithm is an online algorithm, we have regret (which is obtained as difference between the collected reward and the optimum one day by day). How the cumulative regret evolves during time is well explained in the figure 4.2: the graph is coherent with respect to the reward graph because initially the regret is higher due to the fact that we have not yet collected enough samples to learn in a good way the curves, this means that initially the algorithm find several different partition. The regret decreases day by day due to the fact that the algorithm is learning well day by day until it reaches a stable point (approximately 20 days) that correspond to the maximum reward obtained by the algorithm. From this point on the regret is kept quite constant so the graph increases linearly with time. The regret is expressed in terms of lost clicks.

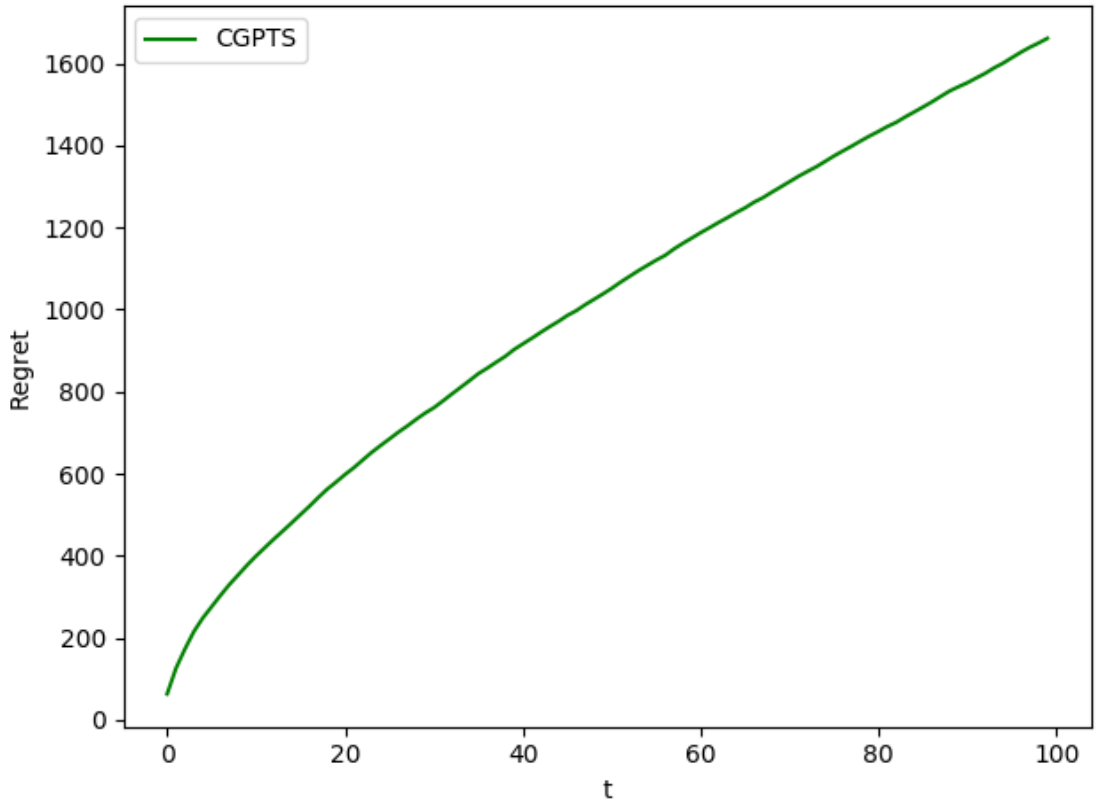
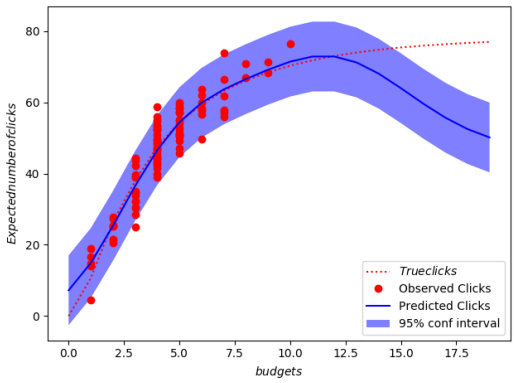


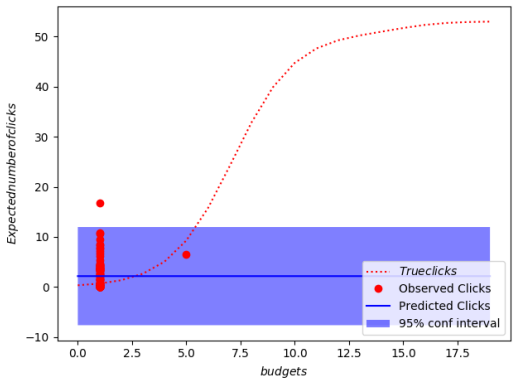

Figure 4.2: Combinatorial GP-TS cumulative regret

**GP regression** In this paragraph we will focus on the Gaussian Processes that compose each GP-TS, in particular we'll analyse how the GPs learn their associated budget/clicks curves and how their average regression error evolves as the number of samples (i.e. days) increases. The average regression error is the highest error among all the arms for that sub-campaign. Due to the fact that the job of the GPs is not yet to learn the budget/click curves in all points but only in the points that have the highest interest, hence all points (i.e. budgets) that are returned by the combinatorial algorithm. This means that each GP will learn the curve in all points associated to the most important budgets. As you can notice in the following figures which represent the GP regression result, taken from a generic experiment the GP associated to Google (4.3a), Instagram (4.3c) and Bing (4.3e) channel were able to learn quite well the curve because during the time evolution several budget are

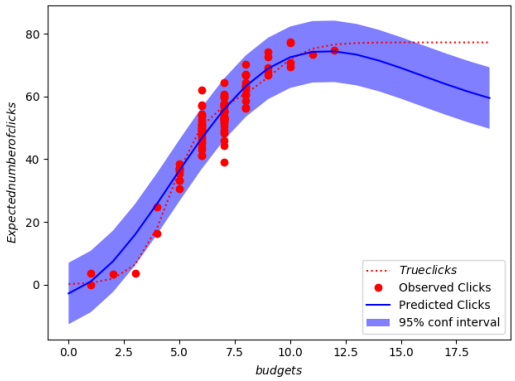
sampled from the environment, while for those associated to Facebook (4.3b) and Youtube (4.3d) were learnt very bad since most of times these two sub-campaign less-considered with respect to all others (i.e. their budget is around 1 €) for these reason these latter GPs approximate their curves as straight lines.



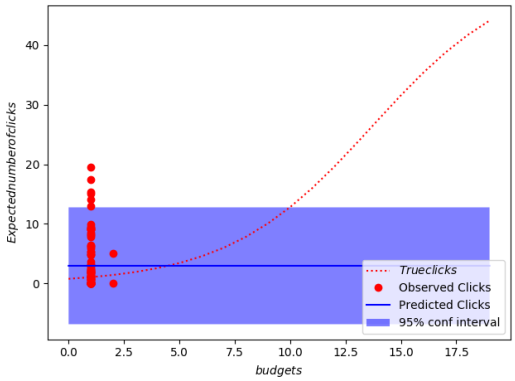

(a) Google channel.



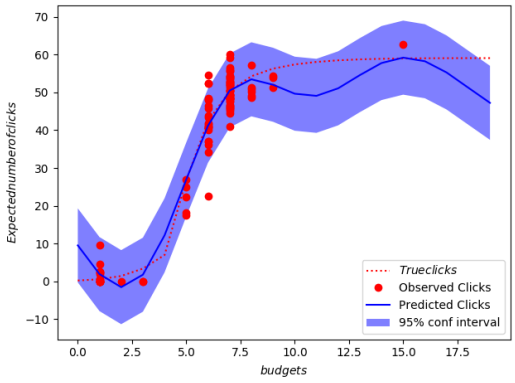

(b) Facebook channel.




(c) Instagram channel.




(d) Youtube channel.




(e) Bing channel.

Figure 4.3: GP regression of all sub-campaigns.

Moreover in figure 4.4 you can see the average regression error of all GPs, as you can notice this is coherent to the previous analysis because the GPs that have the highest average regression error are those of Facebook and Youtube since they approximate the curve as straight lines, while the best ones (which have the lower average regression error) are those of Google and Bing which are able to well approximate the whole budget/click curves, the former reaches a very small error (about 10 clicks) slowly, the latter reaches an higher error, anyway small (about 28 clicks),

much faster. Considering the remaining GP (i.e. Instagram channel) you can notice that the average regression error decreases during time, hence the GP was learning the curve better day by day until it reaches a stable point the is far from 0, this is because the GP has learnt the curves only in points associated to the more prolific budgets, in according to the sampled values, hence this error may avoid our algorithm to reach the optimum.

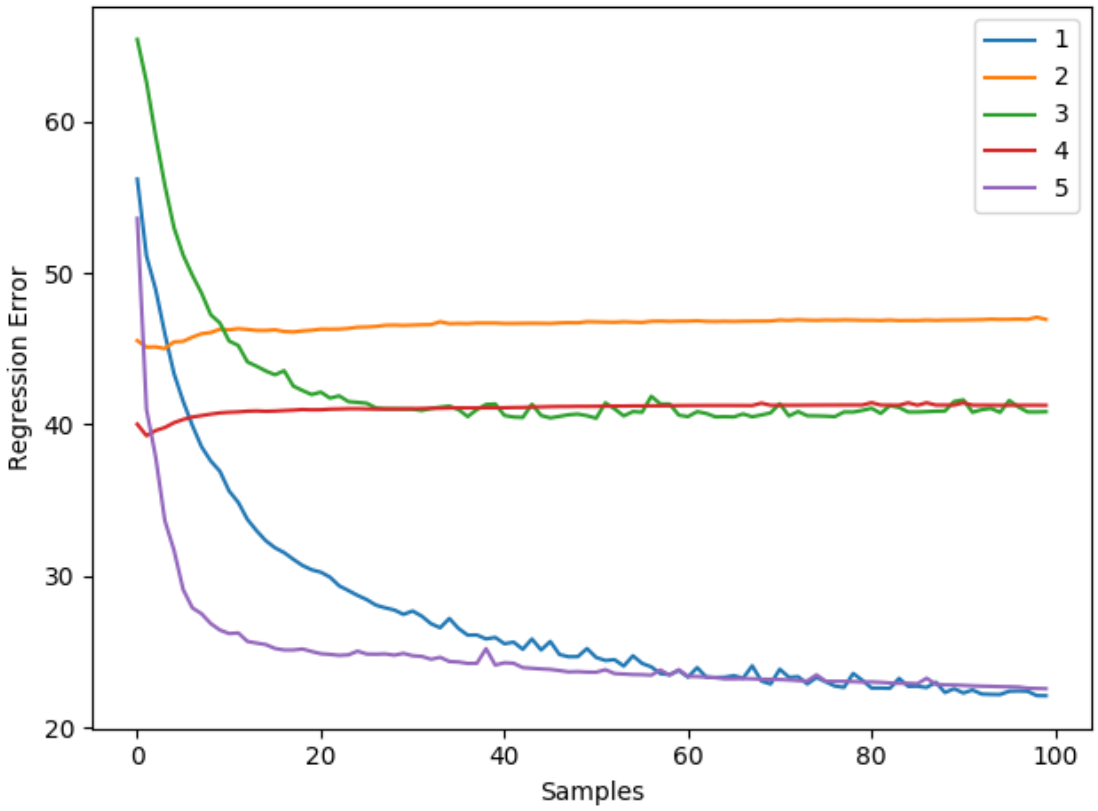



Figure 4.4: Average regression error of the GPs as the number of samples increases

### 4.2.3 Consideration

The first consideration is that, in according to the clairvoyant algorithm, the Combinatorial GP-TS performs quite well, it's learning rate is very high since it requires only about 20 days to reach its maximum reward, and moreover the maximum reward (about 125 clicks) is not so far from the optimum (i.e. 136 clicks). Remember that this analysis was performed forcing the algorithms (Combinatorial GP-TS and Clairvoyant) to assign at least 1 €-budget for every sub-campaigns. Considering the complete analysis, in particular the GPs regression errors, new question have arisen:

- If we allow the algorithms to assign 0€-budget to some some-campaign, are we able to reach an highest optimum?
- In according to the GP regression analysis we have notice that two sub-campaigns are quite always less considered, so are there some useless sub-campaigns that we may prefer to remove?

In order to answer to these new question we have decided to perform another analysis applying the same algorithm but with different settings, this analysis is discussed in section 4.3

## 4.3 CGP-TS (allow empty budgets)

### 4.3.1 Implementation

The implementation of this algorithm is the same described in subsection 4.2.1 except for the fact that here the algorithm can assign 0€-budget to those sub-campaigns that it considers as useless. In particular the settings of this algorithm are represented in the table 4.2:

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Time horizon | 100 days | Number of days for which the algorithm runs |
| Number of experiments | 100 | Number of times that we have repeated the algorithm in order to have more accurate results |
| Number of sub-campaigns | 5 | since we were considering, for each channel the aggregation of the three classes, so we have one sub-campaign per channel |
| Cumulative daily budget | 20€ | this represent the total amount of budget available among all sub-campaigns, for each day |
| Sigma | 5.0 | Standard deviation used sampling from the environment |
| Alpha | 100 | GP parameter, value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. |
| Allow empty budgets | true | boolean value that specifies whether allowing to have sub-campaigns with 0 budget associated or not |

Table 4.2: CGPTS settings

### 4.3.2 Performance

As for the previous section, here we put all analysis performed to this current Combinatorial GP-TS algorithm. Due to the fact that here we have allowed the algorithms to assign 0€-budget to some sub-campaigns, we have found that thanks to this permission we were able to reach an highest optimum, hence this means that there are some sub-campaigns for which it is useless give some budget, because it mainly represent a lost in terms of money, which moreover avoid us to reach more clicks. In this scenario the *clairvoyant* algorithm has produced the following results:

Number of clicks obtained choosing the best partition: **162 clicks**

Best partition:

- **Google** = 5 €
- **Facebook** = 0 €
- **Instagram** = 7 €
- **Youtube** = 0 €
- **Bing** = 7 €

As you can notice now the optimum is 162 which is 13 clicks higher than applying the algorithm forcing to give some budget to all sub-campaigns, consequently even the clairvoyant algorithm determines that there are two sub-campaigns that are useless for our scope. Now we have to test how the algorithm run in this scenario and whether it will be able to find out the same results perceived with the clairvoyant algorithm.

**Reward analysis** The behaviour of the algorithm in terms of reward is quite similar to the previous one, since also here the algorithm is able to reach its maximum in approximately 20 days (figure 4.5), but here the maximum value is around 145 clicks that is an average of 10 clicks higher than the previous one.

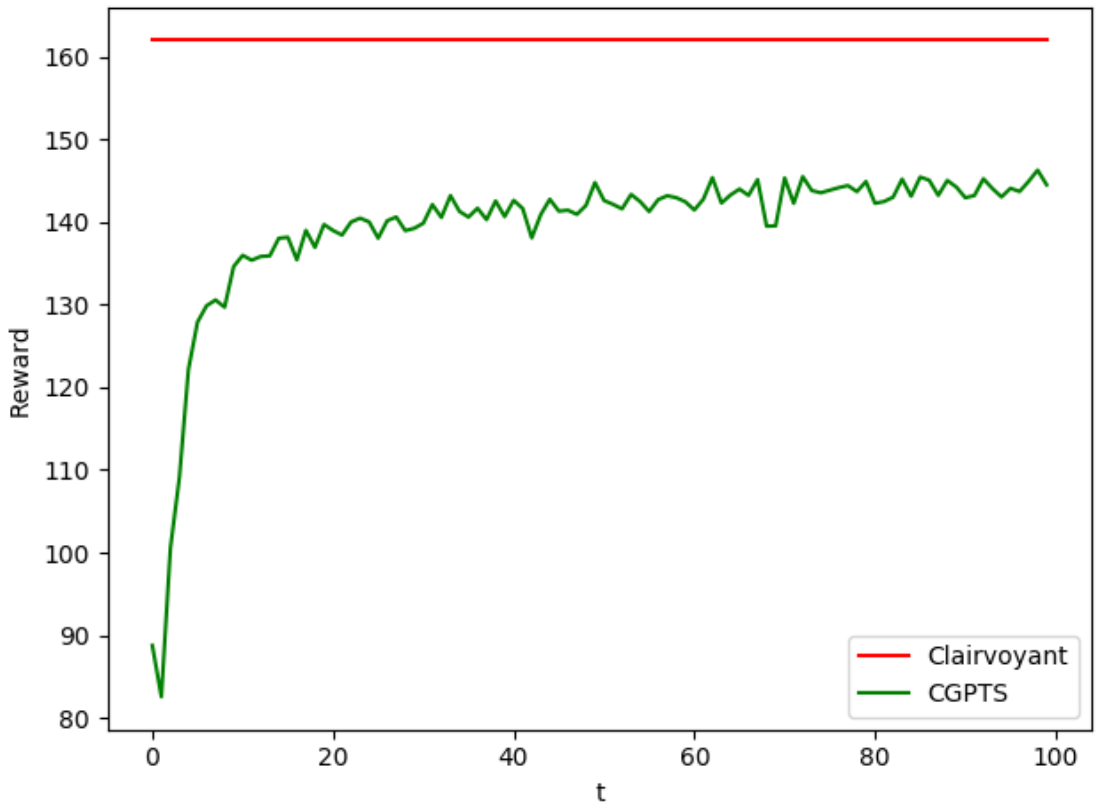



Figure 4.5: Comparison between the reward obtained from the CGPTS and the optimum of clairvoyant algorithm

**Regret analysis** Due to the fact that here we have found that the true optimum of 162 clicks is higher than the 136 clicks obtained in the previous settings we have launched both algorithm in order to compare the regret of these two different settings. As you can notice in figure 4.6 the new settings allow the algorithm to lose less clicks during the execution of the algorithm, hence reaches a cumulative regret of about 1400 clicks lost, instead the algorithms in the previous settings lost much more clicks during time, reaching an higher cumulative regret.

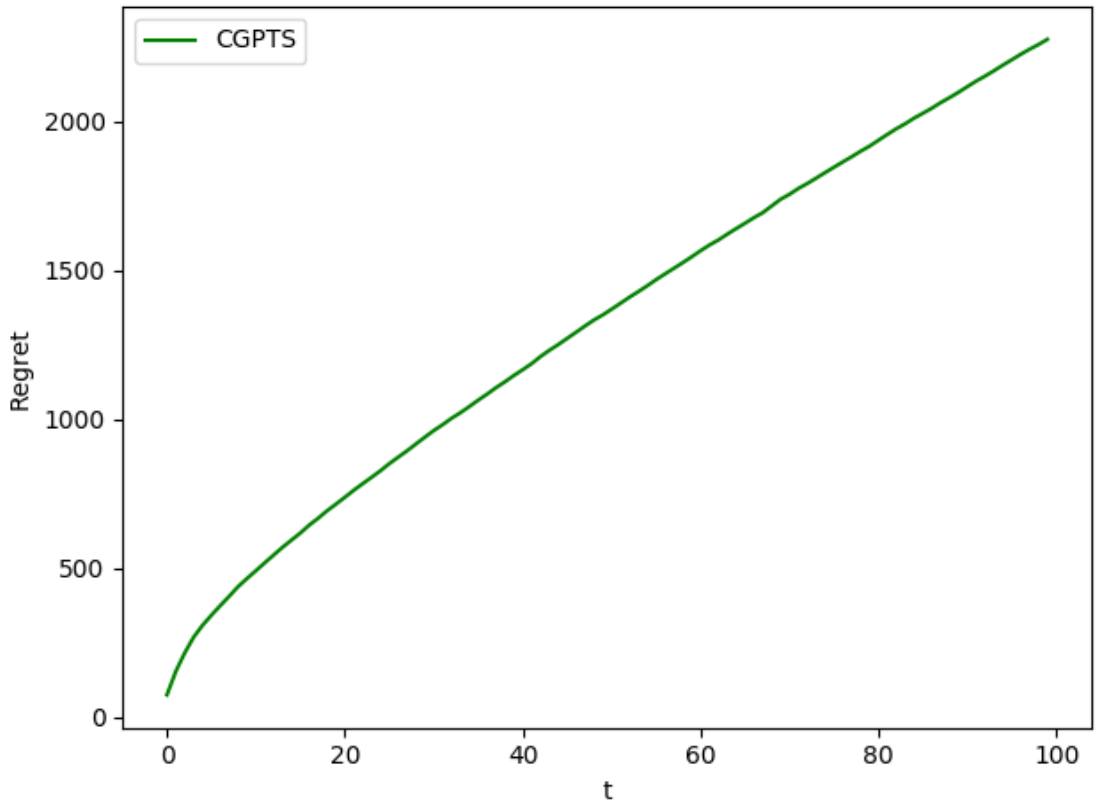
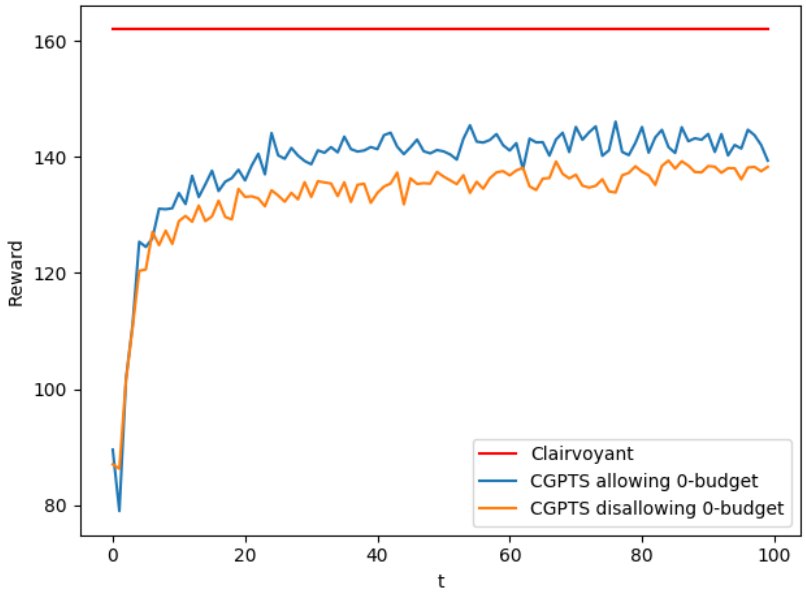
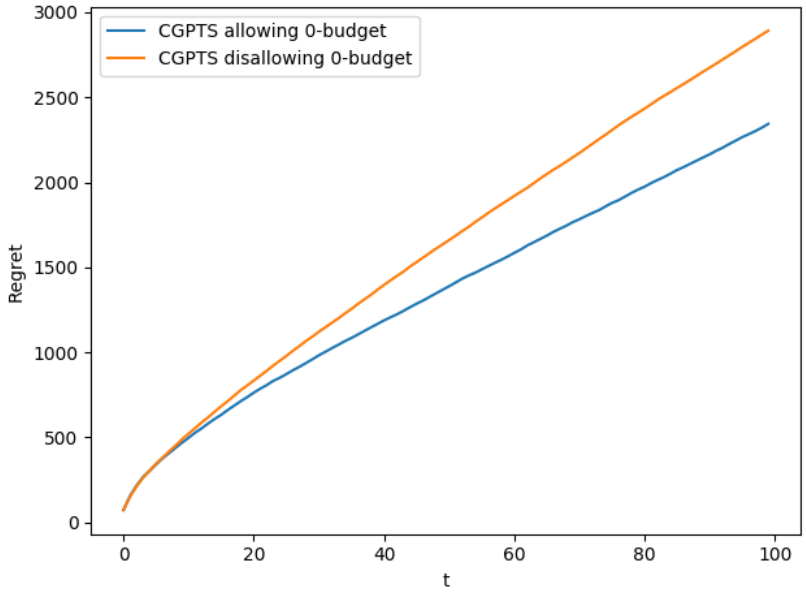



Figure 4.6: Combinatorial GP-TS cumulative regret

**Comparison** In order to show the improvement we have performed an additional analysis that compares the performance of the previous scenario with respect to this one. For computational reason that analysis was performed with 60 experiment per algorithm, this choice reduces the accuracy but we have found that was enough to show the improvement. As you can notice in figure 4.7 the behaviour of both algorithms is quite the same except for the upper bound, since the new algorithm (i.e. the one that assigns 0€-budget) is able to reach an higher stable point, hence saving more regret.



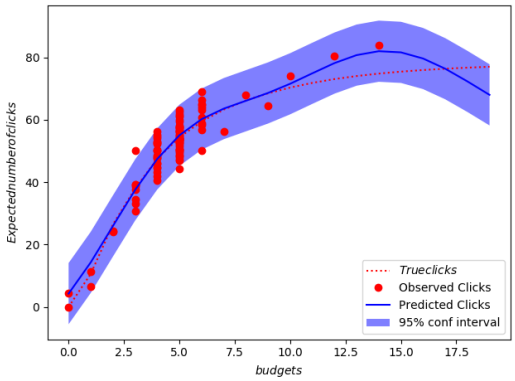

(a) Reward comparison.



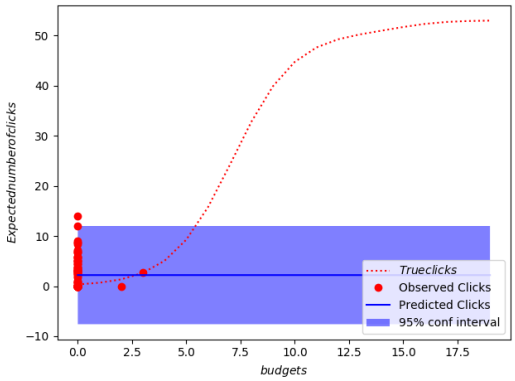

(b) Regret comparison.

Figure 4.7: GP regression of all sub-campaigns.

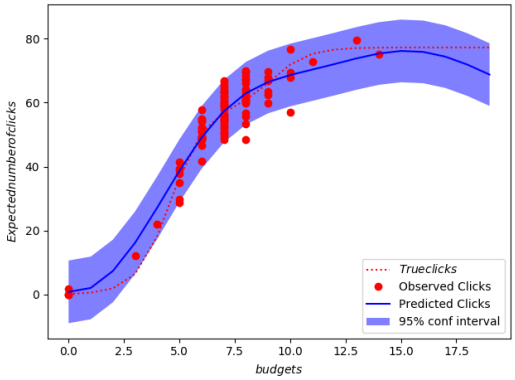
**GP regression** The results of this analysis are quite similar to ones obtained in the application of the algorithm in the previous settings, differently from before the GPs of Facebook (fig. 4.8b) and Youtube (fig. 4.8d) are quite always assigned to have 0€-budget, since they are useless. Hence giving saving budgets from these two GPs we were able to learn well also the Bing curve, allowing us to reach an higher maximum reward.



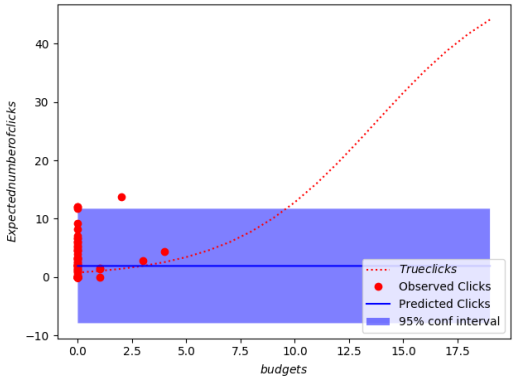

(a) Google channel.



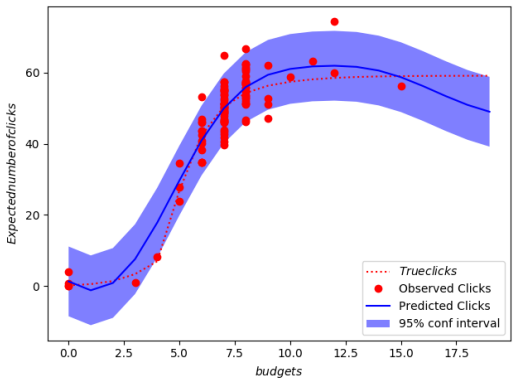

(b) Facebook channel.




(c) Instagram channel.




(d) Youtube channel.




(e) Bing channel.

Figure 4.8: GP regression of all sub-campaigns.

The average regression error of all sub-campaigns is the same obtained in the previous settings except for the Bing GP, that in this case is able to reduce error with respect to the previous one.

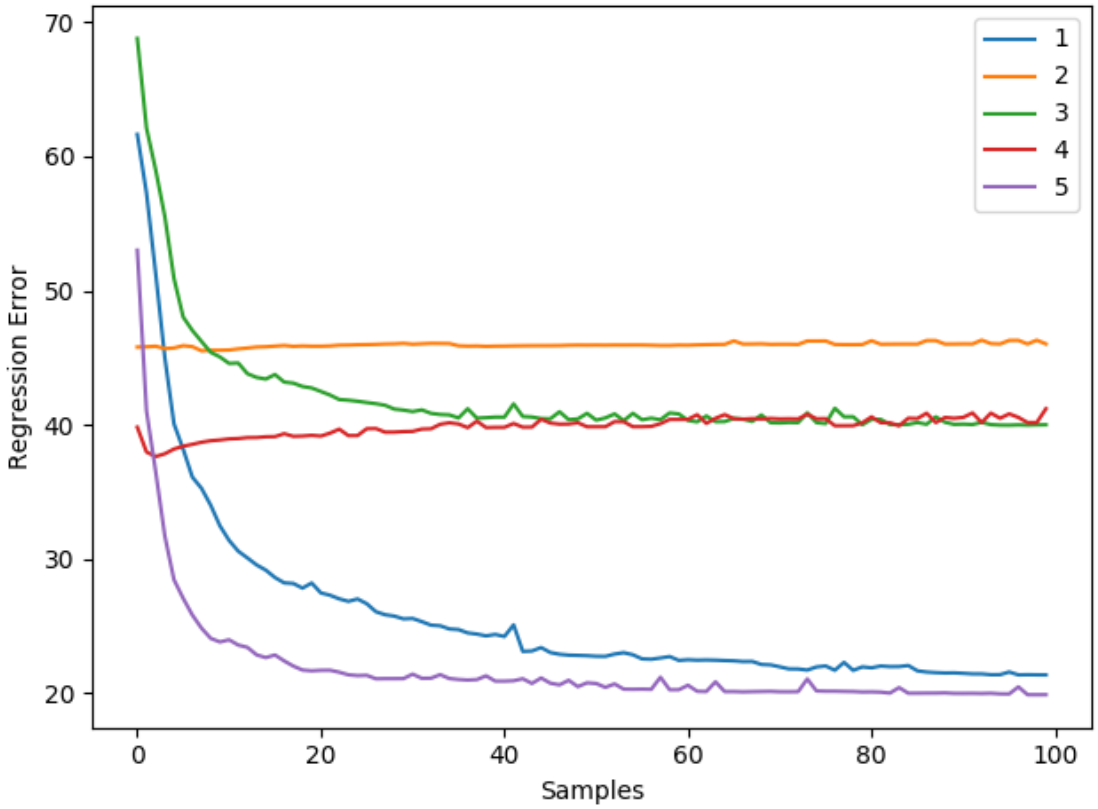
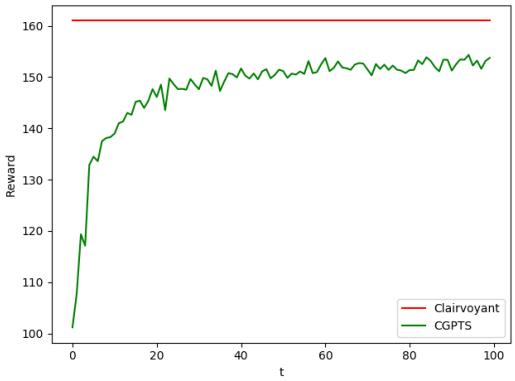


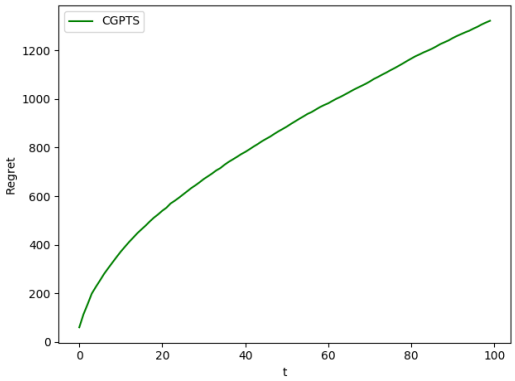

Figure 4.9: Average regression error of the GPs as the number of samples increases

### 4.3.3 Consideration

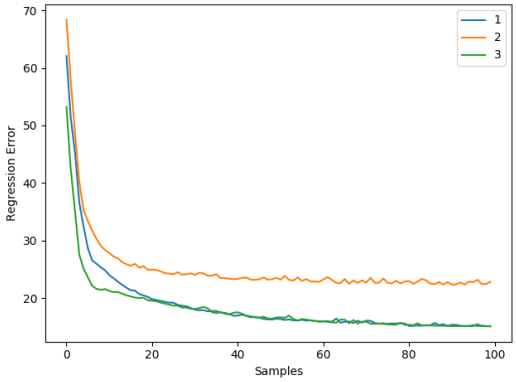
The first consideration is that, as we thought in the consideration of the previous settings, there are some sub-campaigns that are useless for our goal, moreover forcing to assign them some budgets reduces the potentiality of the Combinatorial GP-TS, avoiding to reach higher rewards. In particular the useless sub-campaigns are those of Facebook and Youtube, this can be caused by the fact that their curves stay approximately around zero for too much budget values, and their maximum number of expected clicks is lower with respect all other sub-campaigns. this means that in order to obtain much clicks we need to give them too much budget, which we cannot later assign to other more prolific sub-campaigns (Google, Instagram and Bing). From this point on all the analysis that we will perform using a Combinatorial GP-TS will have these settings, hence the algorithm will assign 0€-budget to those sub-campaigns that it considers as useless. The result obtained completely discarding the useless sub-campaigns, hence considering only Google, Instagram and Bing channels are the following:




(a) Reward.



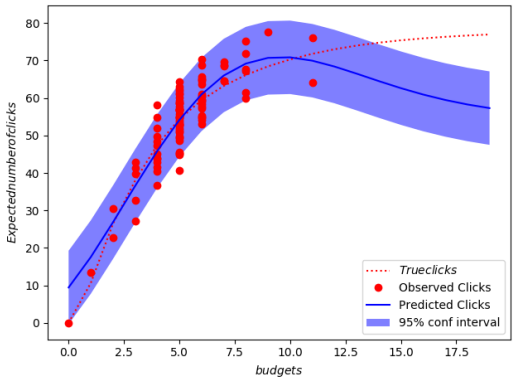

(b) Regret.



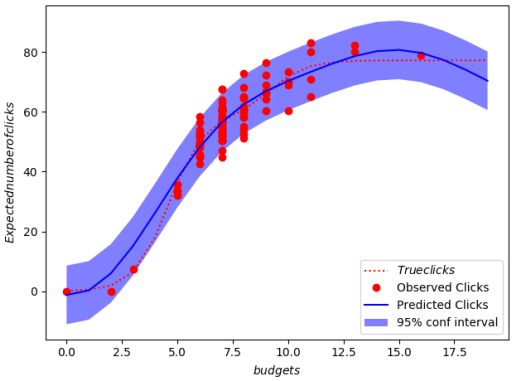

(c) Average regression error.

Figure 4.10: Combinatorial GP-TS performance considering only Google, Instagram and Bing

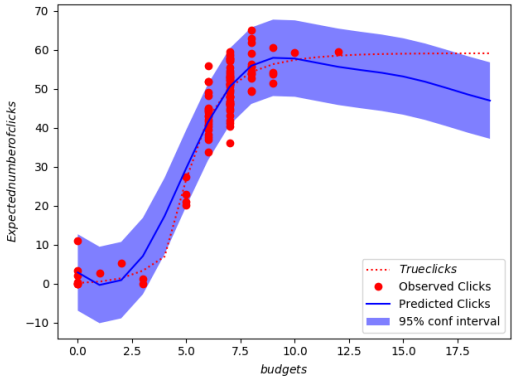
As you can notice from the Combinatorial GP-TS performance, showed in figure 4.10, if we suppose to restart the algorithm without considering useless sub-campaigns, the algorithm is able perform well, reaching an higher maximum value that is quite near the optimum, hence the cumulative regret is less than before because here there is no risk to assign some budget to useless sub-campaigns and finally the GPs are able to learn very well all budget/clicks curves. This last comment can be also seen all images contained in figure 4.11 which show, how the curves are learnt by the GPs, in a generic experiment.




(a) Google channel.




(b) Instagram channel.




(c) Bing channel.

Figure 4.11: GP regression of all sub-campaigns.

## 4.4 Combinatorial G-TS

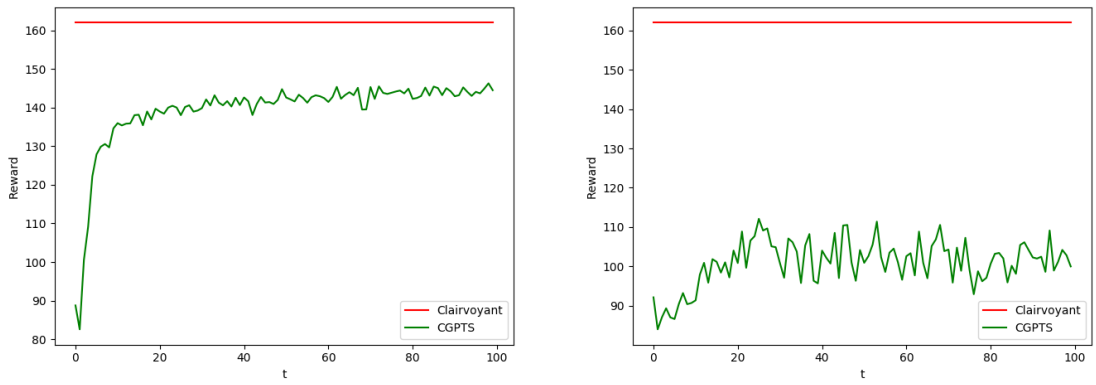
For a complete analysis we have decided to try another algorithm, whose implementation is a combinatorial fashion of a classic Thompson Sampling that uses Gaussian as prior distribution. Before started this comparison we have already known that the G-TS performs worst than the GP-TS, hence our analysis focuses on testing whether introducing a combinatorial scenario increases the performance of the G-TS or it has kept worst than the Combinatorial GP-TS.

### 4.4.1 Implementation

The implementation is quite similar to the Combinatorial GP-TS because we also have five learners here too, where each of one is a Gaussian Thompson Sampling rather than a GP-TS, which works exactly as described in section 3.1. Each day for every G-TS we have to sample a value from each arm (there exists a Gaussian for each arm) and then compute the Combinatorial algorithm (described in section 3.3) in order to choose which arm has to be pulled for all learners (i.e. for all sub-campaigns). Then pull all best arms inside the environment and after that collect all observed rewards and update the Gaussian associated to the pulled arm.

### 4.4.2 CGPTS vs CGTS

In this subsection we provide a comparison between the results obtained using the Combinatorial GP-TS and the Combinatorial G-TS in terms of rewards and regret. As you can notice in the figure 4.12 the Combinatorial G-TS performs very bad with respect to the Combinatorial GP-TS, this can be due to the fact this algorithm during their execution tries too much times bad partition (this is strictly correlated to how a TS works) that disallow the algorithm to reach the same result obtained by the CGPTS.




(a) CGPTS rewards.

(b) CGTS rewards.

Figure 4.12: Rewards comparison between CGPTS and CGTS.

The previous described bad performance cab be easily shown also in the regret comparison, figure 4.13.

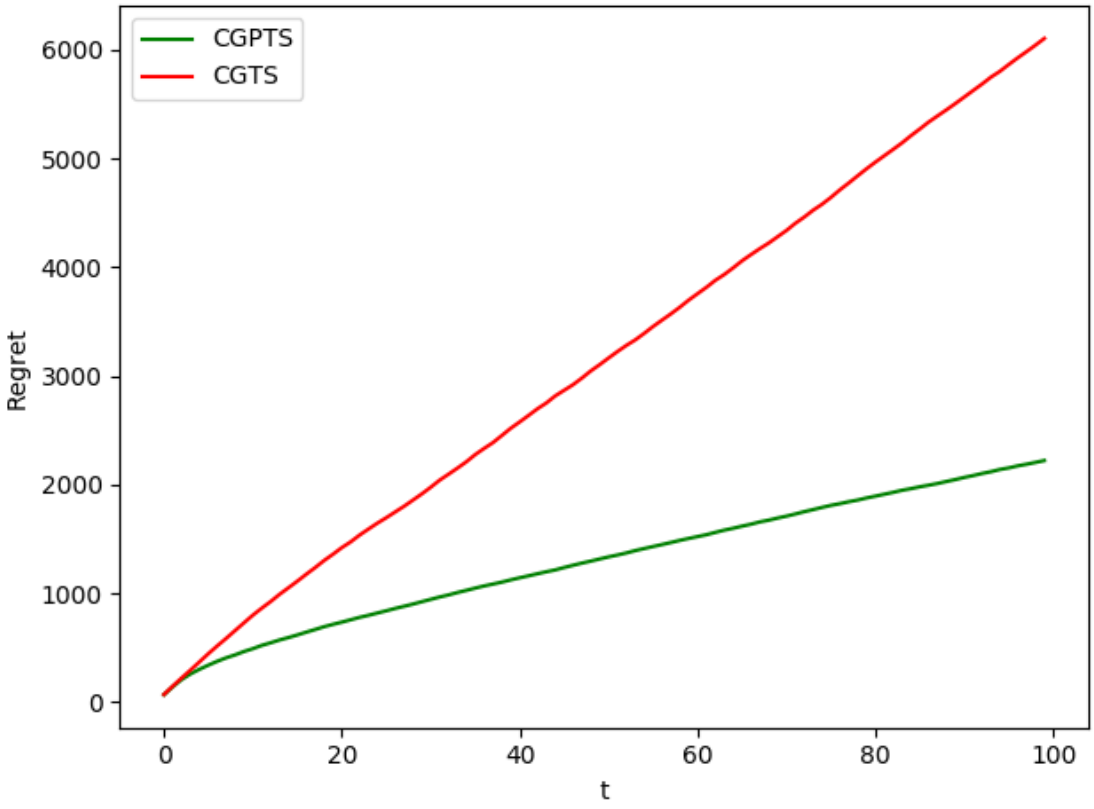



Figure 4.13: Regrets comparison between CGPTS and CGTS

## 4.5 Additional Combinatorial GP-TS analysis

Given the Combinatorial GP-TS described before, the one that allows assigning 0€-budget, in this section we have done a deeper analysis that shows how the algorithm perform over different Gaussian Processes parameters settings. In particular we have provided a deeper analysis of how the Combinatorial GP-TS perform by changing the value of *alpha*.

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Time horizon | 100 days | Number of days for which the algorithm runs |
| Number of experiments | 30 per alg. | Number of times that we have repeated the algorithm in order to have more accurate results |
| Number of sub-campaigns | 5 | since we are performing the aggregate analysis, hence for each channel we have a single sub-campaign. |
| Cumulative daily budget | 20€ | this represent the total amount of budget available among all sub-campaigns, for each day |
| Sigma | 5.0 | Standard deviation used sampling from the environment |
| Alpha | 9, 25, 100 | GP parameter, value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. |
| Allow empty budgets | true | boolean value that specifies whether allowing to have sub-campaigns with 0 budget associated or not |

Table 4.3: CGPTS settings

As evidenced by the graphs in figure 4.14 and figure 4.15 reducing the alpha parameter increases the reward and hence reduces the cumulative regret. This could be done due to the fact that we have less noise than what we expected. (Notice that in the figures we have alphas of 3,5 and 10 instead of 9,25 and 100 because in the GP we have input their power of 2)

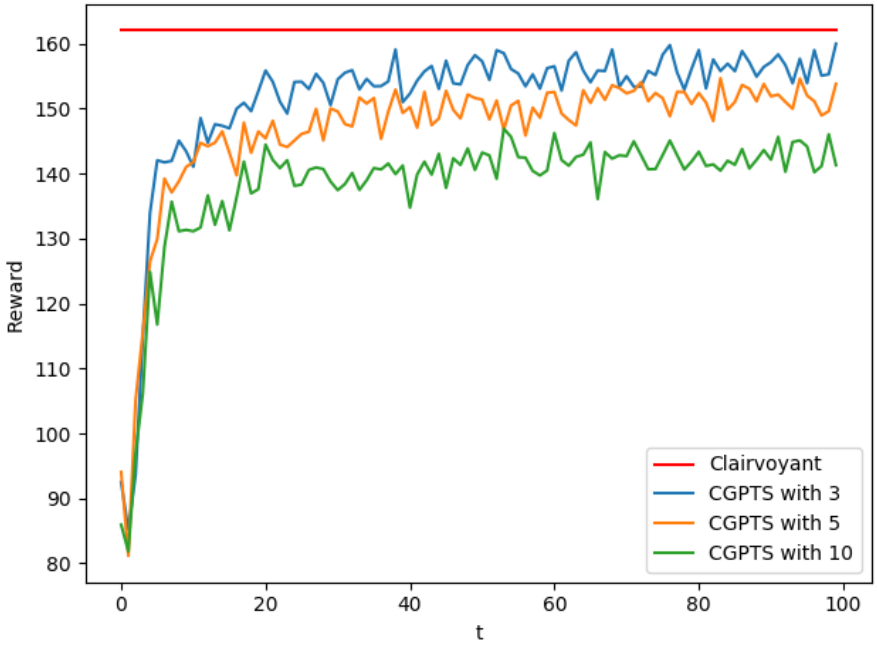



Figure 4.14: Reward analysis with different values of alpha

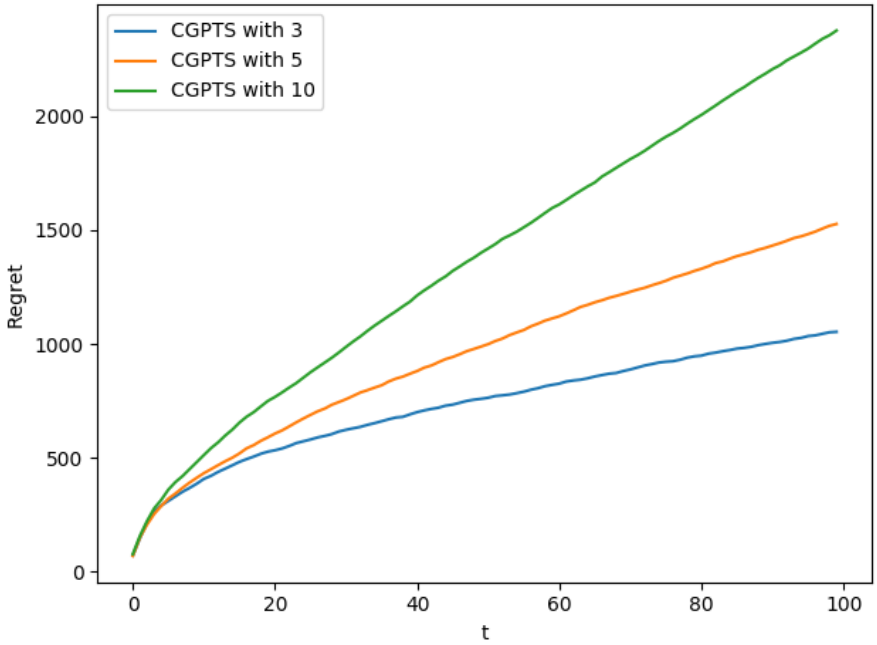



Figure 4.15: Regret analysis with different values of alpha

From this point on, all the analysis that we will perform adopt an alpha of 9 since it was the best value that we have found, that allows the algorithm to quite reach its optimum.

# Chapter 5

# Disaggregation and context identification

This chapter analyses the scenario in which there exist an algorithm, in parallel to the Combinatorial GP-TS, that identifies the context, such that it is able to provide us from which classes the clicks come from, hence the goal of this analysis is to check whether at some point is better disaggregate the aggregate curves and so create new sub-campaigns or not. The check has to be performed the first day of every week.

## 5.1 Context identification

This algorithm basically works identifying from which class (in according to the classes defined in section 2.2) the clicks, observed from the environment, come from.

- *Input:* Takes as input $c$, the number of observed clicks for a given sub-campaign, which is associate to an aggregation of $n$ curves. (our case $n = 3$)
- *Output:* Returns an $n$ length integer array, where $i^{th}$ value correspond to $i^{th}$ class of the $n$ classes and it tells how many clicks of $c$ come from that class.

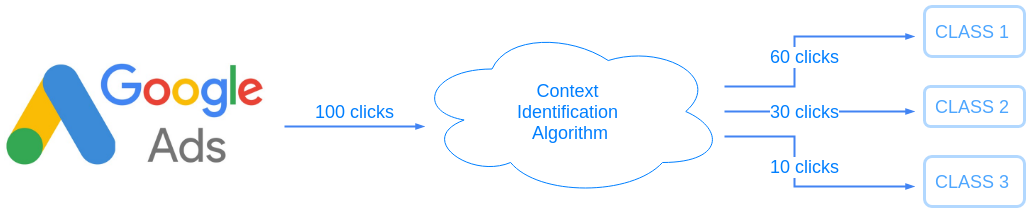



Figure 5.1: Context identification algorithm diagram

## 5.2 Implementation

The algorithm is a Combinatorial GP-TS, as described in section 4.3, initially the scenario consists of five sub-campaign, where each of one is associated to the aggregated curve. The difference between this algorithm and the classic Combinatorial GP-TS is that here during the execution of this online algorithm, thanks to the context identification algorithm that provides us from which classes the observed clicks come from, it learns in parallel other combinatorial GP-TS considering all the possible disaggregation combination (e.g. (class1, class2, class3), (class1+class2, class3), (class1+class3, class2) and so on) and then the first day of every week the algorithm checks whether which Combinatorial GP-TS provides the highest reward. If the algorithm finds out that is better to disaggregate then it removes the aggregated sub-campaign which is substituted by the new sub-campaigns (two or three in according to which disaggregation is chosen) and from this point on it keeps this new settings. Notice that this algorithm continues checking whether disaggregate or not until the sub-campaign is completely disaggregated in the three classes. Notice that for computational reasons we have decided to apply the context identification and disaggregation check only on the first sub-campaign (i.e. Google), but even we this simplification we have noticed a huge improvement in the performance.

**Update** During the execution of the Combinatorial GP-TS with five sub-campaigns, as already said, we need to steadily update all other combination of Combinatorial GP-TS, considering all the possible classes partition of the Google sub-campaign. The update works as follow:

1. Once we have pulled all arms, that we have found out through the combinatorial algorithm, consider the reward obtained by the Google sub-campaign (i.e observed number of clicks), thanks to the *context identification* algorithm we were able to identify from which classes those clicks come from. Once we have identified the observed number of clicks for every classes we have to update all GPs related to the Combinatorial GP-TSs that we were running in parallel.

2. As result of the context identification algorithm we have now the number of clicks for each class (i.e. class 1, 2 and 3) and the aggregate pulled arm (i.e. pulled budget). Our approach was to split the budget (one for each class) in according the percentage of people who belong to that class, so for the class 1 we assign 0.65% of budget, for the class 2 we assign 0.305% of budget and finally for the class 3 the 0.045% of the budget

3. If it is not the first day of every week then go to the point number 1, otherwise check whether it is better disaggregate or keep the curve aggregated, this last operation is performed solving the combinatorial algorithm in all possible combination, hence considering all possible classes partition (i.e considering all parallel Combinatorial GP-TS as substitute of the Google GP-TS) once solved the combinatorial algorithm for each combination choose the ones that provides the highest combinatorial reward. If the best combination is not the aggregate one, substitute the latter with the best one then if the best one is the complete disaggregate combination stop checking and continue the algorithm with these new three sub-campaigns, otherwise go to point 1 with the new two sub-campaigns.

The following results were obtained given this configuration:

| **Parameter** | **Value** | **Description** |
|---|---|---|
| Time horizon | 100 days | Number of days for which the algorithm runs |
| Number of experiments | 60 | Number of times that we have repeated the algorithm in order to have more accurate results |
| Number of sub-campaigns | initially 5 | this parameter is changed during learning whether the alg. finds out that is better to disaggregate |
| Cumulative daily budget | 20€ | this represent the total amount of budget available among all sub-campaigns, for each day |
| Sigma | 5.0 | Standard deviation used sampling from the environment |
| Alpha | 9 | GP parameter, value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. |
| Allow empty budgets | true | boolean value that specifies whether allowing to have sub-campaigns with 0 budget associated or not |

Table 5.1: CGPTS settings

## 5.3 Performance

In order to consider the disaggregated learners reliable, we have supposed that there was need to wait at least two weeks before start checking whether disaggregate or not, since we have thought that after this time enough samples were collected and so the disaggregated learners would be sufficiently reliable. Given this settings we found that, as we already thought, the algorithm finds out that is better disaggregate as soon as it was allowed. In particular the algorithm perform the first disaggreagtion around the $14^{th}$ and $21^{th}$ day and the second disaggregation (whether could be performed) around the $28^{th}$ and $42^{th}$ day. The first disaggregation can be easily seen in the figure 5.2 because there is a great difference between the *no disaggregated* reward and the *disaggregated* one, while for the second disaggregation is quite harder to see the improvement because there exist a small difference between a partial disaggregation and total one, furthermore approximately for the 40% of the experiments the algorithm completely disaggregate the first time, hence without performing the partial disaggregation step. The best results, associated to the Clairvoyant algorithm and obtained completely disaggregating the first sub-campaign, are the following:

Number of clicks obtained choosing the best partition: **183 clicks**

Best partition (complete):

- **Google C1** = 5 €
- **Google C2** = 0 €
- **Google C3** = 0 €
- **Facebook** = 0 €
- **Instagram** = 7 €
- **Youtube** = 0 €
- **Bing** = 7 €

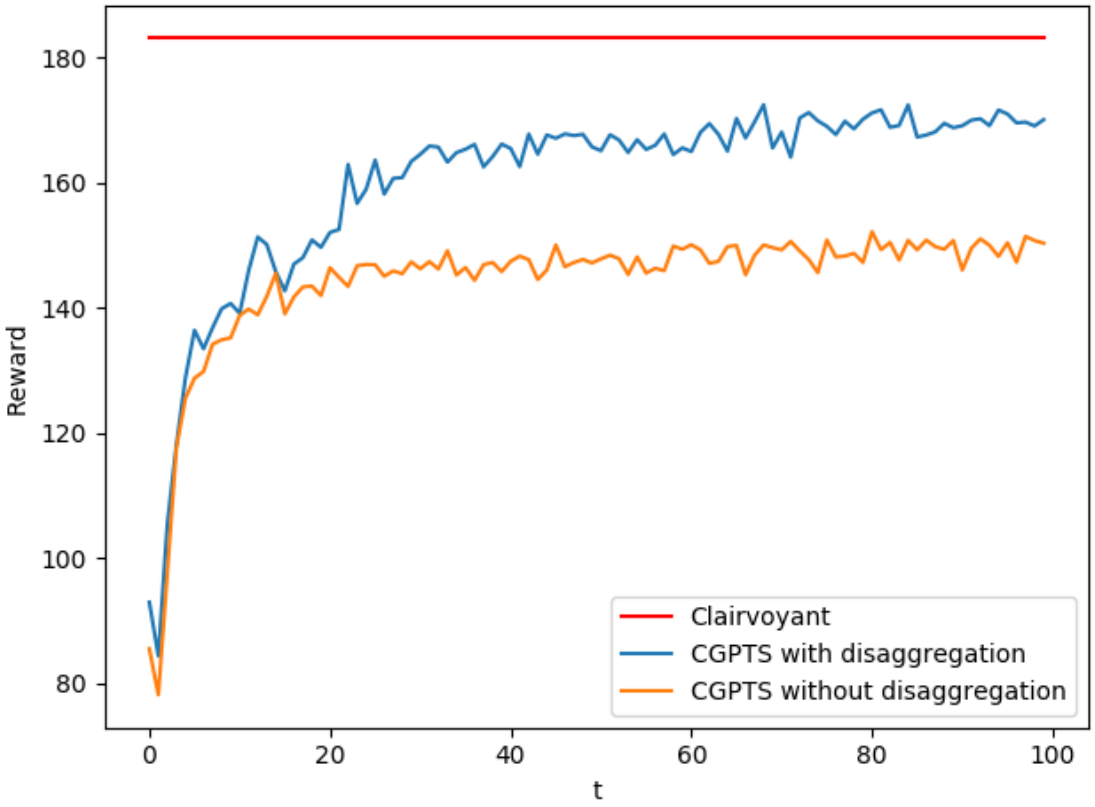



Figure 5.2: Reward comparison between disaggregation and not disaggregation.

In particular we have reached the complete disaggregation in the 75% of the experiments, in the 22% of the experiments we have partially disaggregate the Google sub-campaign but in this percentage the algorithm was not able to disaggregate again the curve, in the remaining part, approximately 3% of the experiments, the algorithm did not perform the disaggregation. This results highlight the fact that the algorithm is quite always able to recognize the fact that disaggregate is better than keep the curve aggregated, hence this is a good performance result. As showed in figure 5.2 the Combinatorial GP-TS that disaggregates is much better than the

aggregate one, in particular with this new algorithm we were able to gain much reward as soon as it starts disaggregating.

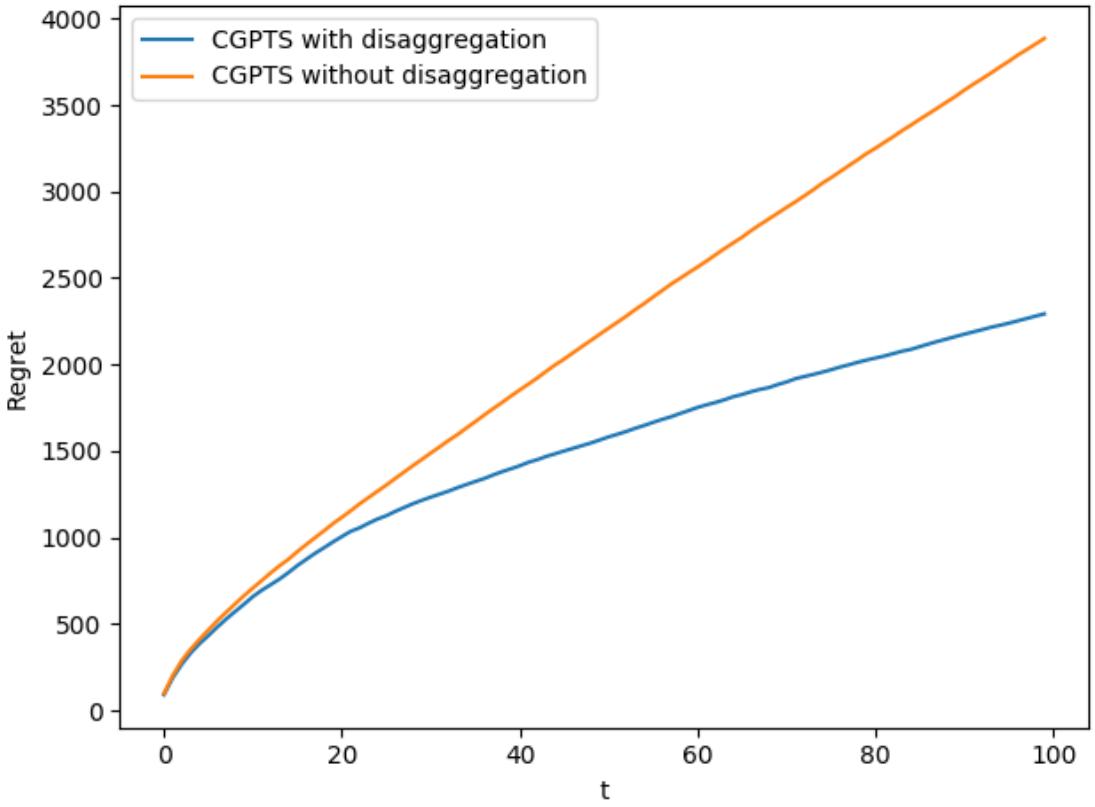



Figure 5.3: Regret comparison between disaggregation and not disaggregation.

In according to the reward analysis just described, the regret (figure 5.3) follows exactly what we have already said, around the first disaggregation point the regret start increasing less then before the disaggregation, this is due to the fact that we are observing higher rewards from the environment, hence the slope of the cumulative regret reduces. After this improvement the regret continues increasing almost linearly with time.

# Chapter 6

# Tools

**Analysis** All discussed analysis were performed modeling the environment and the algorithms with python programming language, in particular we have used the following modules:

- *Numpy*: used for numerical computation.
- *Sklearn*: this module was used to implement the Gaussian Processes, in particular we have used the GaussianProcessRegressor for the GP model, RBF and ConstantKernel for the kernels used to implement the GP.
- *Matplotlib*: 2D plotting library which is used to produce all the graphs that we have provided in this document.

The whole project was written using the PyCharm Integrated Development Kit and it is shared on GitHub (repository), which is used for code versioning and sharing.

**Documentation** The documentation is fully written using LaTeX, which is a high-quality typesetting system. The curves represented in section 2.2 are taken from FooPlot, which is an online function generator, while all graphs reported in the analysis are directly results of the matplotlib modules adopted.