POLITECNICO
MILANO 1863

ADVERTISING

Documentation

DATA INTELLIGENCE APPLICATION

Academic Year 2018/2019

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

Contents

Chapter 1

Introduction

The product we decided to use for this project is the SAMSUNG GALAXY S10. This product is brand new, so we based our assumption on the previous model, the Samsung Galaxy S9 and, more in general, looking at the past trends in the smartphones market.

1.1 Purpose

The purpose of this project is to choose a product and decide which is the best price to assign to it in order to have the greatest profit, by using different algorithms with different setups. The work that we have done follows these steps:

1. Selection of the product to be sold
2. Identification of the features and the classes of users
3. Definition of market phases and corresponding class of users behaviour (demand curves)
4. Definition of the time horizon and the possible prices/candidates of our analysis
5. Application of several algorithms (K-testing and MAB algorithms) that give us the better prices according to the market phase
6. Comparison of the obtained results

1.2 Product description

The Samsung Galaxy S10 was released on the 8th March 2019. Its an Android smartphone manufactured by Samsung Electronics and, leaving out all the technical specifications, we consider it as a very popular product (Samsung is one of the most popular smartphone brands together with Apple Inc. and Huawei) with a trend like its past models and other competitors products. We consider it as user friendly and less iconic than the Apple products. We hypothesized a production cost of 350 €, relying on the information found in Internet.

Figure 1.1: Samsung Galaxy S10

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.
- **Class and environment description**: In this chapter we provide a description of the features and users' classes that we have identified from them, the market phases that we have supposed in our analysis and the corresponding users behaviour.
- **Time horizon and candidates**: This is a preliminary chapter that introduces the time horizon that we set in our analysis and the candidates/prices that we decided.
- **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 market phase (obtained as aggregation over the three classes defined in previous chapters).
- **Disaggregation ad context identification**: This chapter provides results and consideration about a disaggregation analysis, which is performed along with a context identification algorithm.
- **??**: This is a conclusion chapter that summarises all the consideration done for all algorithms, the scope here is to provide some results for this scenario.
- **Tools**: This chapter simply provides tools and libraries adopted in order to perform all analysis and to draw up this document.

Chapter 2

Class and environment description

2.1 Features selection

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

- *Age*: Students, Workers, Retires
- *Sex*: Male, Female
- *Region*: Advanced economies, Less developed

We indeed assumed that the behaviour of a male customer is different from a female one and, similarly, a customer from an economic advanced country will behave differently form a less developed one *(See chapter: 2.2 Class descriptions)*.

Note: firstly, for the Region feature, we've also considered the "Poor countries", but then we decided to remove them because theyre out of the market were considering.

2.2 Class description

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

MALE 0.5	Students 0.35	Workers 0.45	Retires 0.2
Advanced Economies	0.105	0.135	0.06
Less Developed 0.4	0.07	0.09	0.04

FEMALE 0.5	Students 0.35	Workers 0.45	Retires 0.2
Advanced Economies	0.105	0.135	0.06
Less Developed 0.4	0.07	0.09	0.04

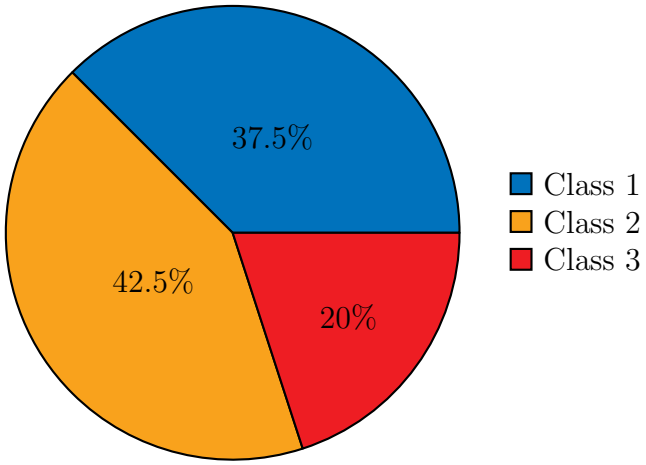

Figure 2.1: Customers classes pie chart

- **Class 1**: this class is characterized by the male students and workers of the economic advanced countries and the female workers of these ones. We assume that the members of this class have no problem in paying higher prices for buying our product that is indeed user friendly.
- **Class 2**: in this class we can find the students and the workers of the less developed countries. Young girls of the developed countries behave similarly to the previous customers because, we assumed, that theyre more interested in a more iconic and famous phone like the Apple ones. Here, the members prefer to pay our product at lower price because, we supposed, that they have not much money to spend.
- **Class 3**: this class is composed by the male and female retires of all the kind of country. This class' members are not very interested in buying expensive phones because, we assumed, they prefer simpler and cheaper phones. In this class there also a small group of particularly rich members that consider the goodness of a phone proportionally to its price, but there are very few.

2.3 Phases and curves

We identified 4 different phases in our scenario:

1. ***Market launch***: this is the first phase, when the product enters the market. We assumed that for the first 3 months the demand for all the classes remains approximately the same, after that, we hypothesized some little smooth changes, in particular in the medium-high price range.
    - *Class 1*: here the demand is overall high for the prices below 1000 €, after which decreases. We assumed that the customers evaluate our product basing on the price of the previous model at the same phase.
    - *Class 2*: in this class the customers have less money than the previous ones, so the demand decreases if the price exceeds 500 €.

- *Class 3*: the demand is generally low since the customers of this class are not very interested in buying our product.

Phase	Period	Duration
Market launch	From February to August	7 months

Demand curves:

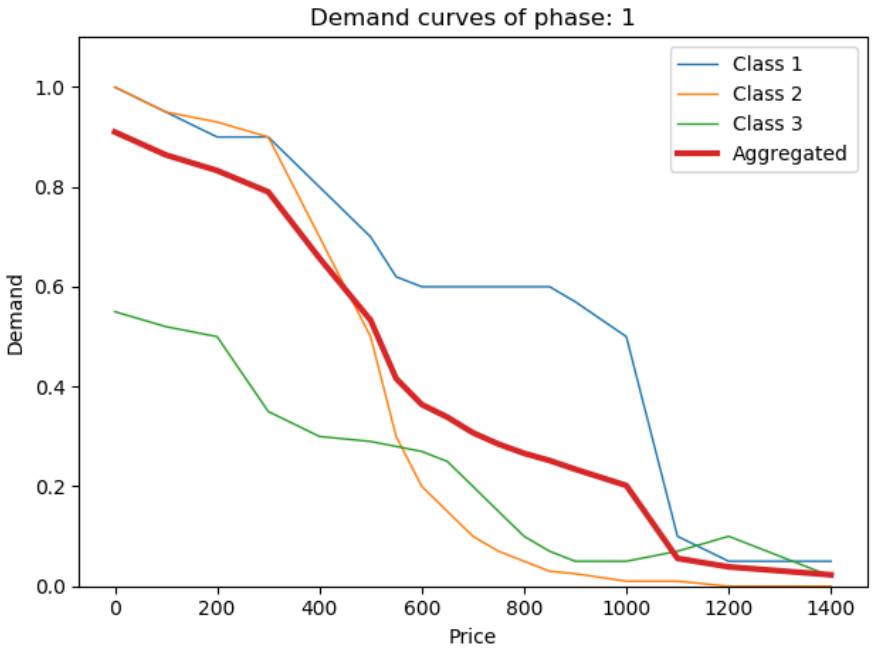

Figure 2.2: Phase 1 aggregated curve

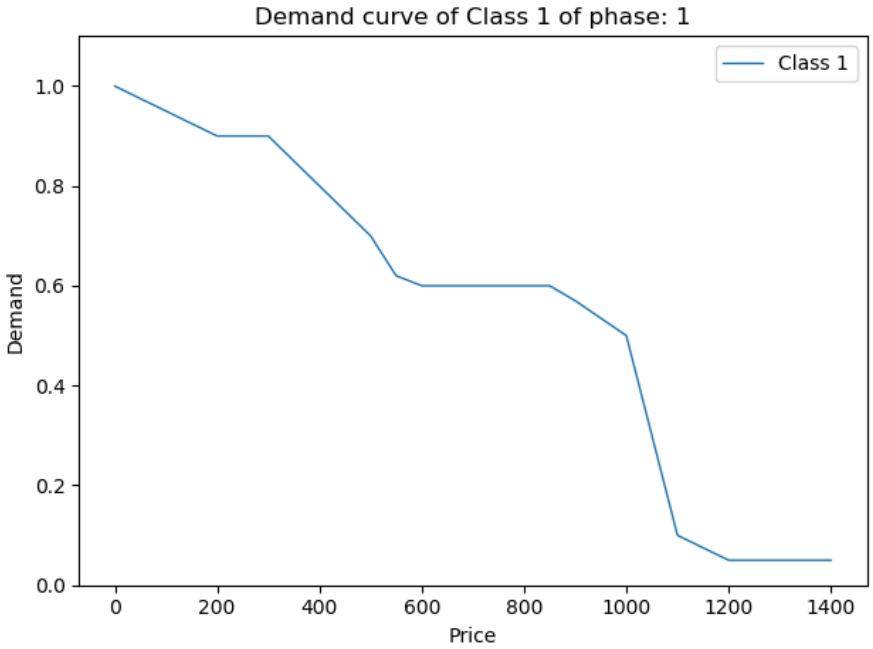

Figure 2.3: Phase 1 class 1 curve

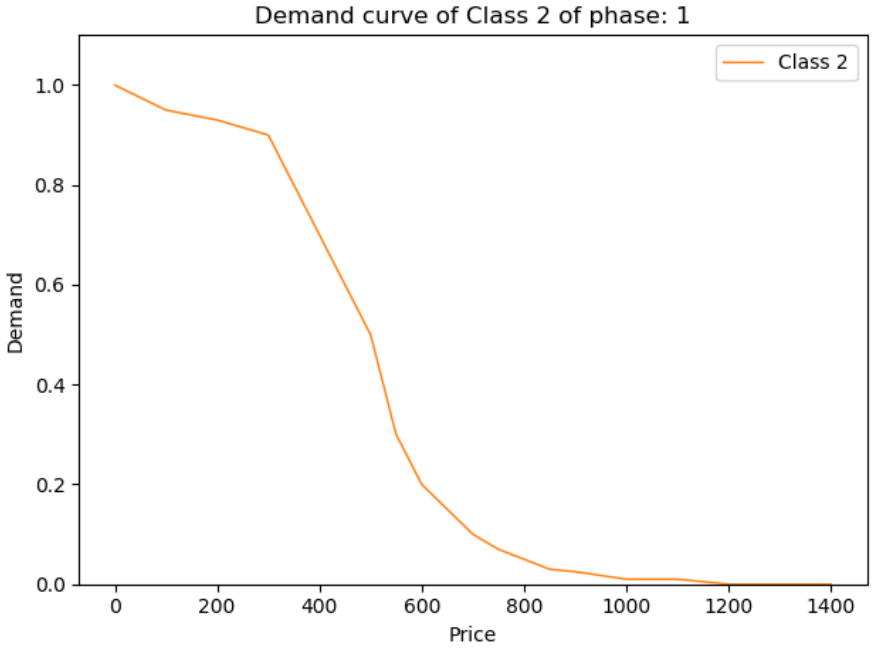

Figure 2.4: Phase 1 class 2 curve

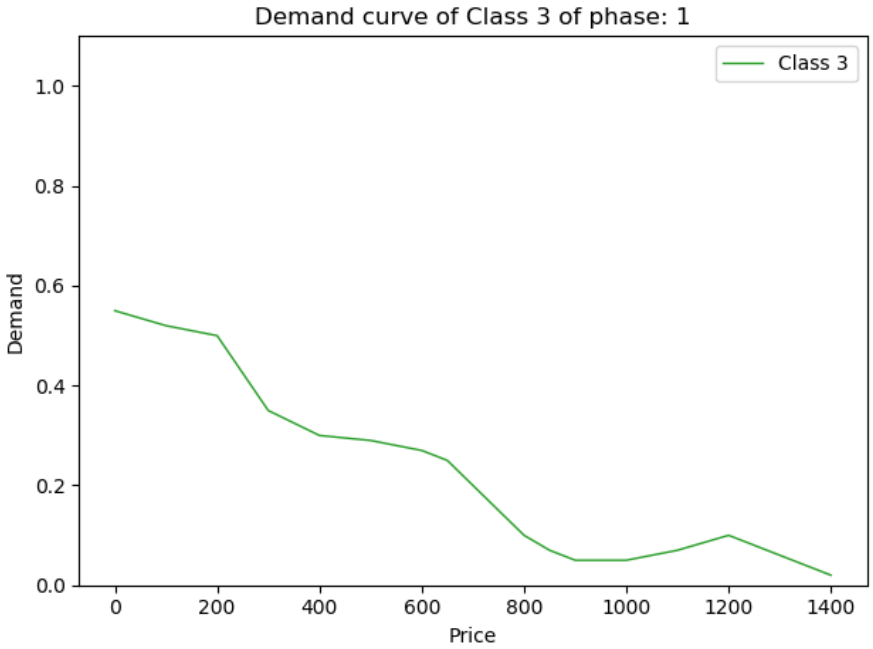

Figure 2.5: Phase 1 class 3 curve

2. ***Competitors' new products***: we assumed that in September the 2 main Samsung's competitors (Apple Inc. and Huawei) decide to release their new products. This leads to an abrupt change in the demand that decrease drastically for high prices.

   - *Class 1*: the demand falls for prices above 400 €; we assumed that this kind of customers prefers, cost being equal, to buy the new smartphone in the market (we've assumed the new iPhone model).
   - *Class 2*: similar consideration for this type of customers, that prefers to buy a new and cheaper phone (the new Huawei model in this case).
   - *Class 3*: the demand softly decreases and there is a flattening of the demand of what we previously called rich members. As we already said, these few people consider the last released smartphone as the best in the market.

Phase	Period	Duration
Competitors' new products	From September to November	3 months

Demand curves:

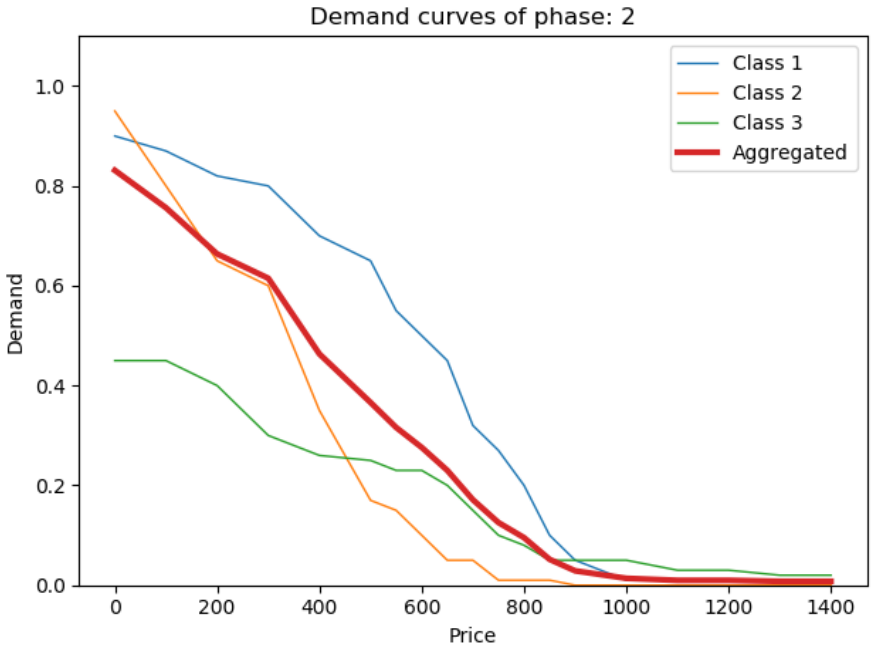

Figure 2.6: Phase 2 aggregated curve

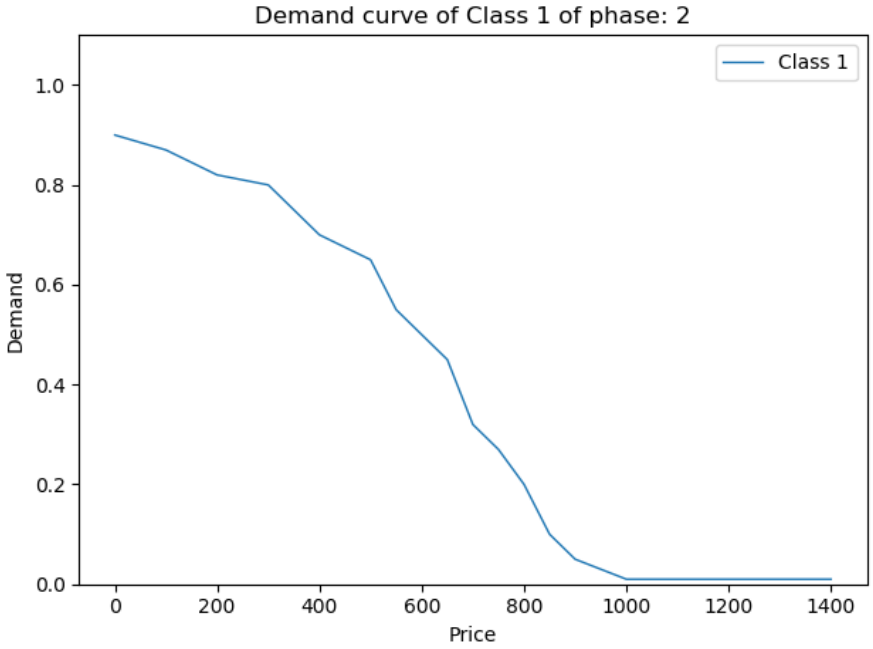

Figure 2.7: Phase 2 class 1 curve

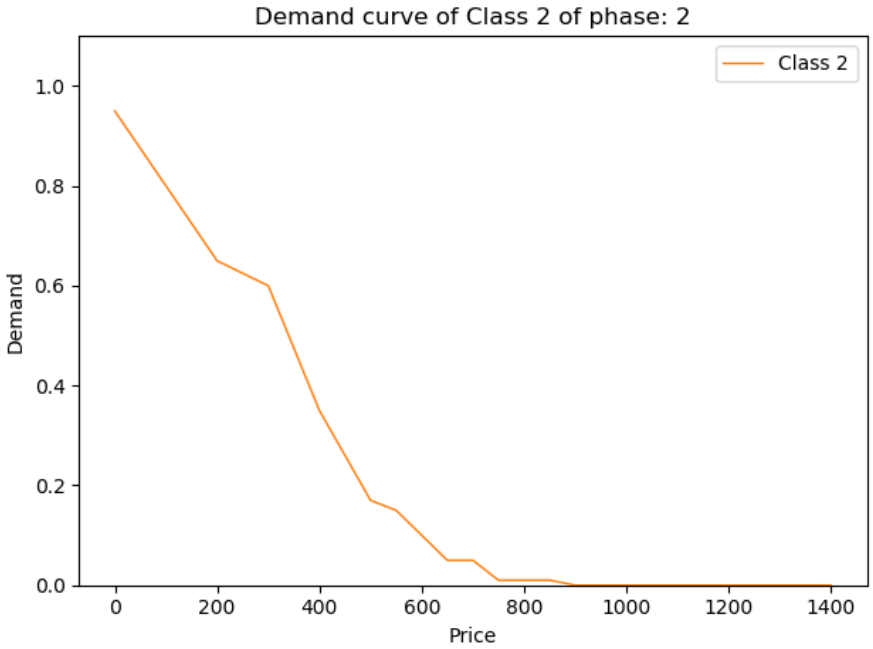

Figure 2.8: Phase 2 class 2 curve

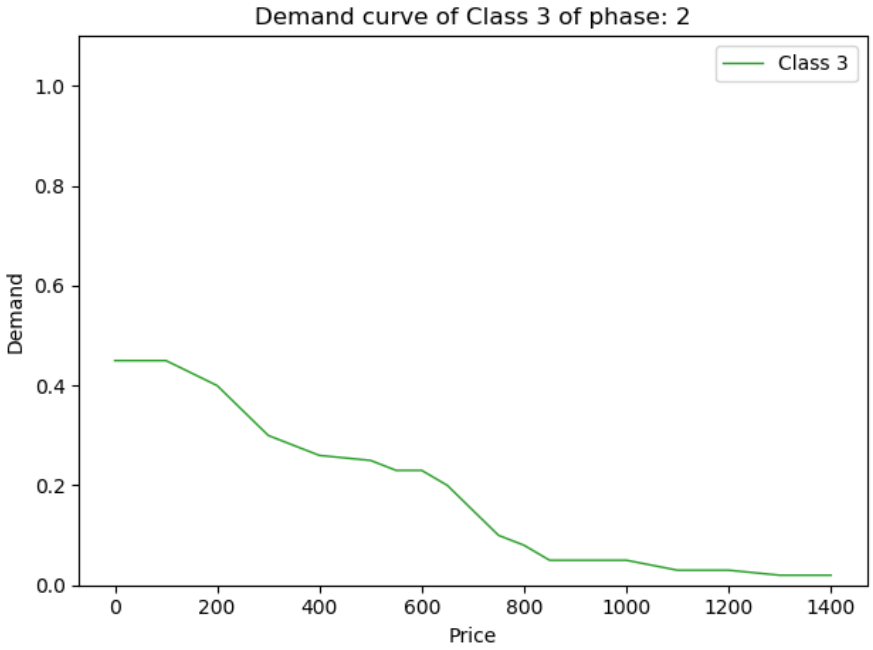

Figure 2.9: Phase 2 class 3 curve

3. ***Holiday***: we hypothesized that during winter holidays, due to the fact that there are several festivities, people increase the demand, because, usually, technological product are a very popular gift.

- *Class 1*: the demand remains more or less the same for lower prices (people take advantage of holiday offers for example) and increase a little bit for medium-higher prices (this kind of customers allows himself to spend a little more for a gift).
- *Class 2*: here the demand increases a little bit for lower prices (again for holiday offers) and remains the same for the others.
- *Class 3*: also in this class the demand increases a little, especially for very high prices, due to the rich members. However, it decreases for medium prices range (we assumed that retirees not the rich group dont spend too much money for a not well-known product).

Phase	Period	Duration
Holiday	From December to January	2 months

Demand curves:

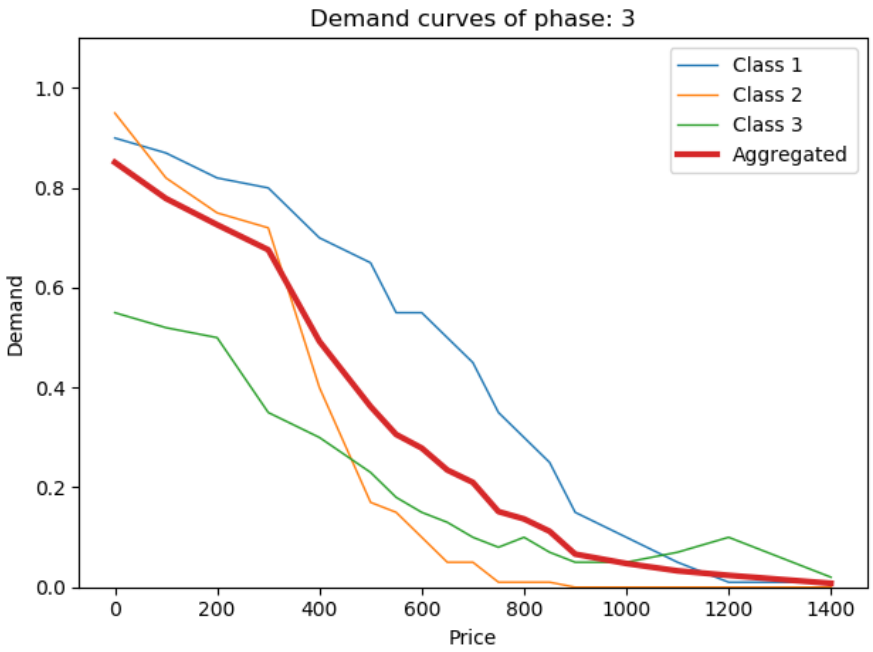

Figure 2.10: Phase 3 aggregated curve

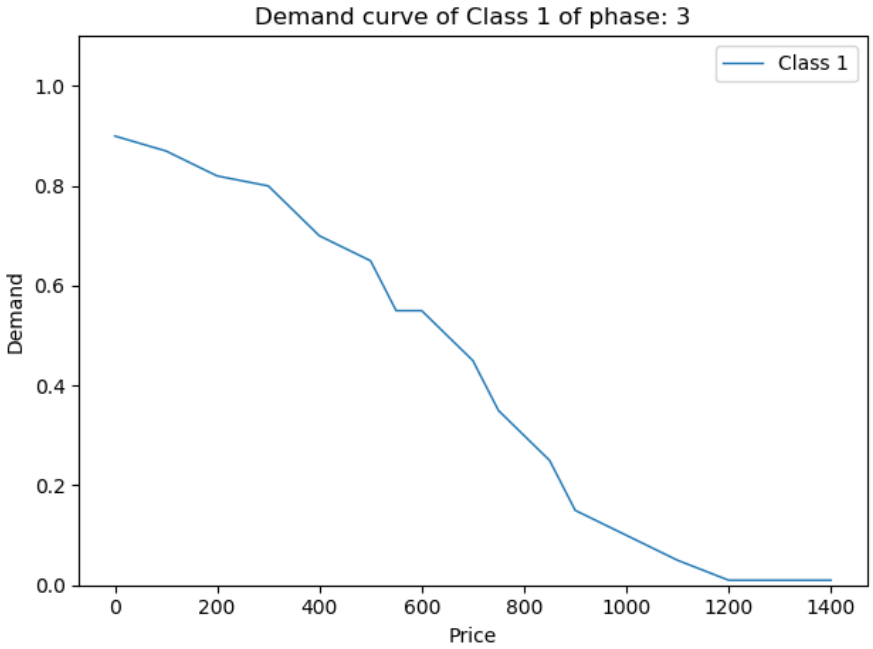

Figure 2.11: Phase 3 class 1 curve

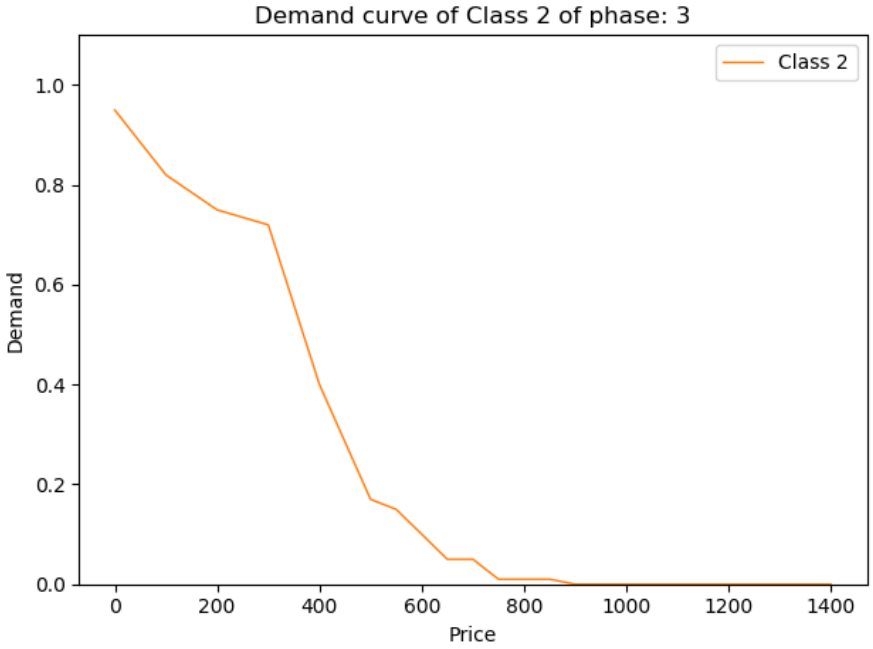

Figure 2.12: Phase 3 class 2 curve

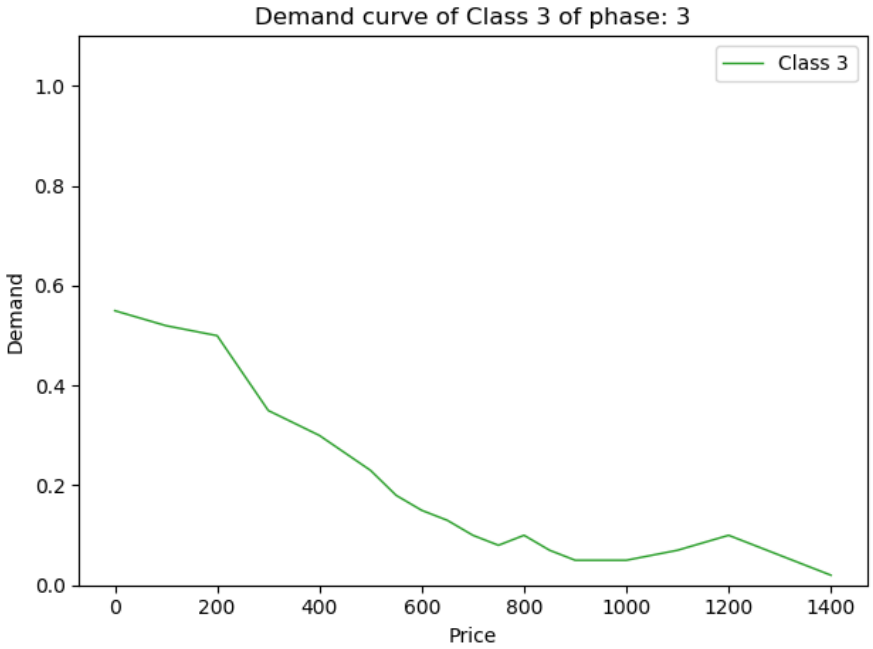

Figure 2.13: Phase 3 class 3 curve

4. ***New model***: Samsung releases the new smartphone model (Samsung Galaxy S11). This is the last phase we decided to consider.

   - *Class 1*: because this class of customer is predominantly composed by wealthy people, as soon as the new model of a smartphone is released, the demand strongly decreases, also due to the fact that, in general, the price of the new model is close to the previous model release price.
   - *Class 2*: the demand increases for low prices because usually, as soon as the new model is released, the prices of the previous models decrease. Instead, for higher prices it decreases due to the fact that this kind of customers prefer to keep their money for the new model.
   - *Class 3*: the demand trend returns as in the competitors new product phase. The motivations are very similar of the ones expressed in the aforementioned phase.

Phase	Period	Duration
New model	From February to April	3 months

Demand curves:

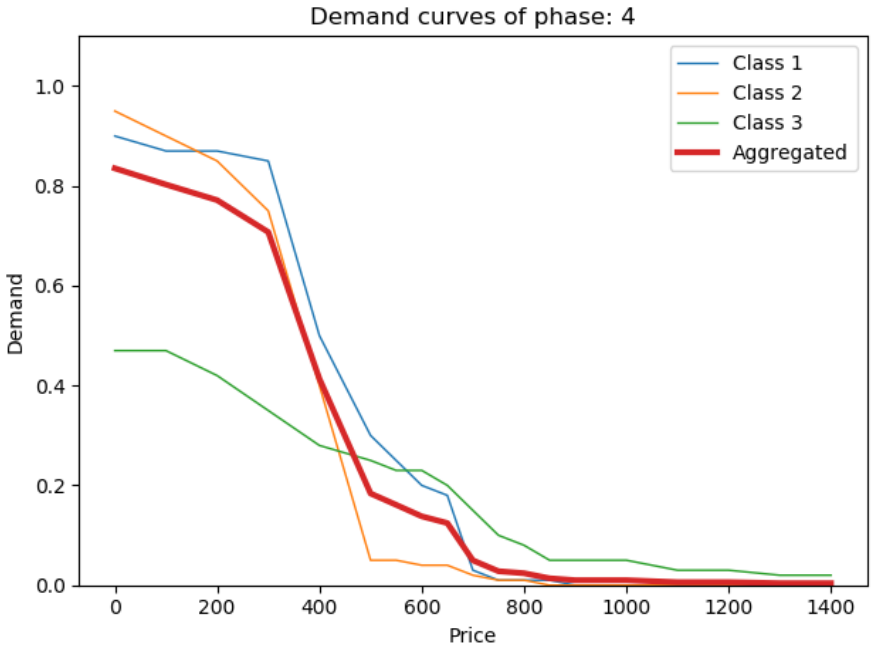

Figure 2.14: Phase 4 aggregated curve

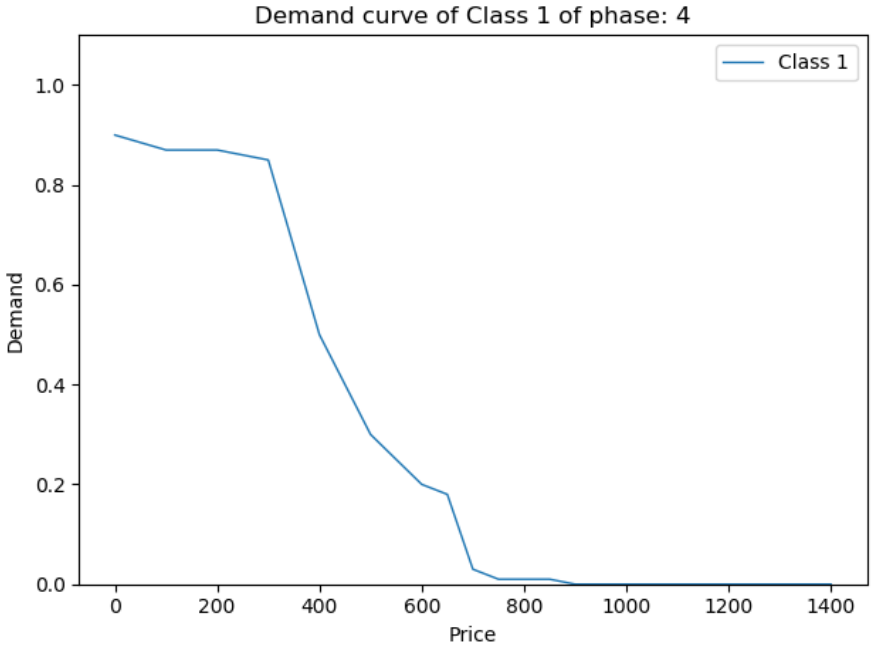

Figure 2.15: Phase 4 class 1 curve

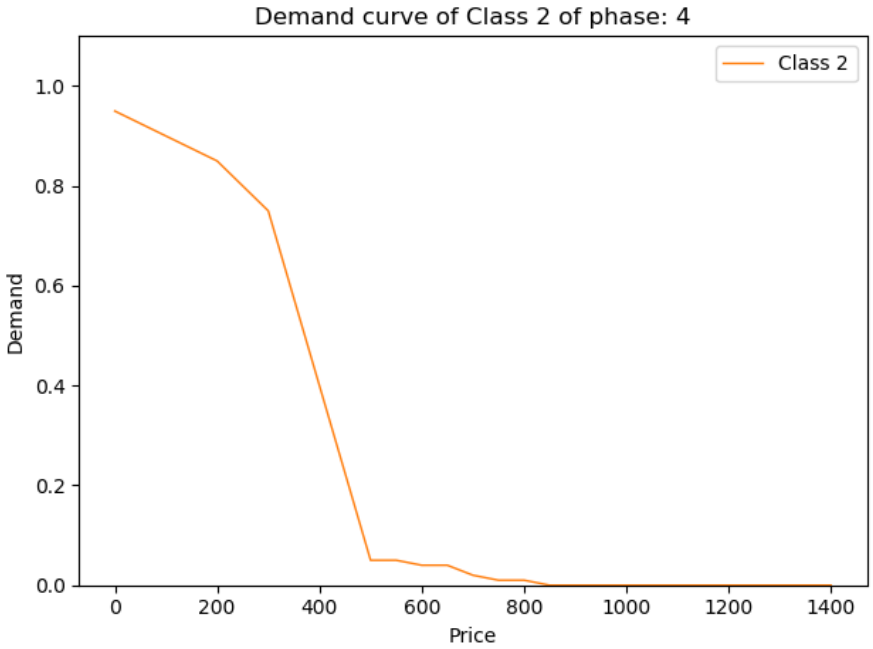

Figure 2.16: Phase 4 class 2 curve

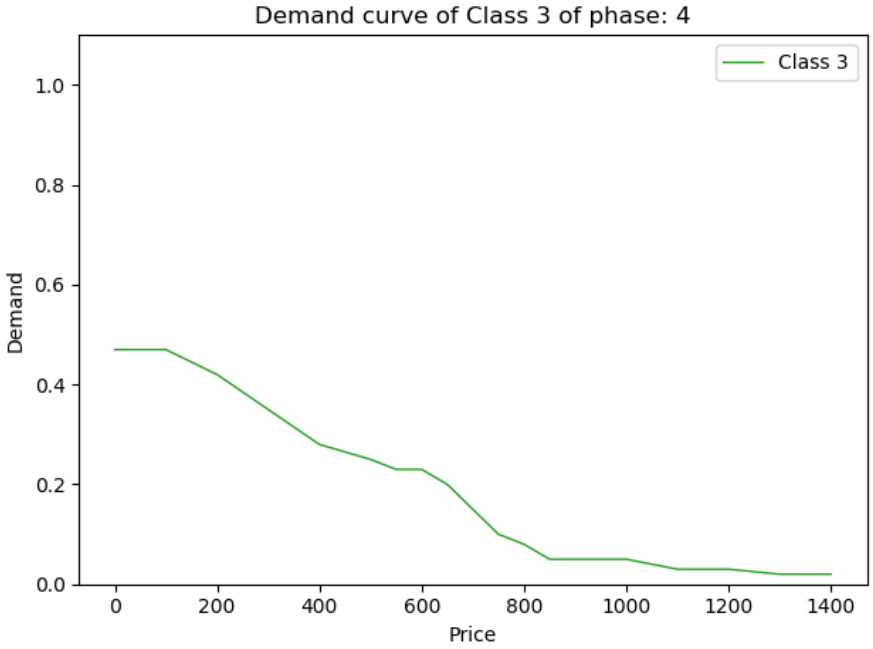

Figure 2.17: Phase 4 class 3 curve

Chapter 3

Time horizon and candidates

3.1 Time horizon

The time horizon we decided to consider (as already introduced in the phases description) starts from February 2019 and it ends in April 2020. Considering an average of 30 days per month and a total of 15 months we obtain a time horizon of 450 days.

3.2 Candidates selection

We considered a price range from 0 € to 1400 € and we identified 19 possible prices/candidates: we divided the range into intervals of 100 €, except for the prices from 500 € to 900 € where we reduced the interval to 50 €, because we assumed that in this range a small variation of the price would lead in a significant variation in customers behaviour and so in the demand curve.

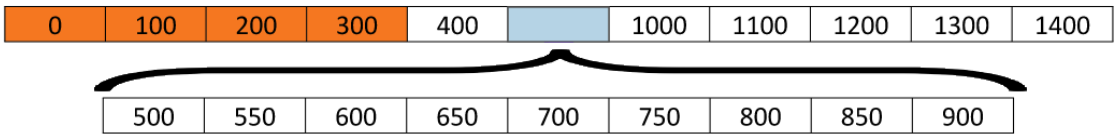

Figure 3.1: Candidates

Since we have assumed a production cost of 350 € for unit, the **highlighted prices** would get us a negative reward. Given this situation, we decide to perform our experiments considering only the prices that would get us positive rewards *(no highlighted prices)*, in order to project us in a more realistic analysis.

Chapter 4

Preliminary concepts

In this chapter we provide a technical description about the algorithm that we used in our analysis.

4.1 Sequential A/B testing

In A/B testing we have 2 groups of users and 2 possible candidates (prices in our case) that are associated to 2 different configurations: A (already used configuration) and B (new configuration). With this kind of testing we want to verify which one of the two configurations is better collecting samples from the groups of users. In the Sequential A/B testing we repeat this operation sequentially until all the candidates are evaluated. At the end we will obtain the best candidate/price.

Here we report the main phases that characterize Sequential A/B testing:

- *Hypothesis definition*: here we define the hypothesis to verify for the single A/B test.

Hypothesis		Action performed
H0	u1 = u2	Select the new price/candidate
H1	u1 > u2	3 Keep the old price/candidate

- *Test statistic selection*: here we set the statistic that we used for our testing.

$$z = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\bar{y}\,(1-\bar{y})\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}} \qquad \bar{y} = \frac{n_1\,\bar{x_1} + n_2\,\bar{x_2}}{n_1+n_2}$$

- *Accuracy selection*: before collecting the data we set the accuracy of our test and the relaxing coefficient:

  - α: the significance level.
  - β: the power level.
  - δ: the alternative hypothesis relaxing coefficient.

- *Number of sample selection*: Each time we perform the single A/B testing we compute the minimum number of samples needed to take a good decision. This is done at every comparison because the number of samples is dependent by the standard deviation of the two current candidates:

$$n = \frac{\left(z_{(1-\alpha)} + z_{(\beta)}\right)^2 \sigma^2}{\delta^2}$$

  With:

$$\delta^2 = \left(\frac{p_1 + p_2}{2}\right)\left(1 - \frac{p_1 + p_2}{2}\right)$$

- *Statistic computation*: computation of the chosen statistic with candidates probabilities and the computed minimum number of samples.
- *Decision taking*: once the statistic is computed, we check if reject or not the null hypothesis (H_0). If z is greater than $z_{(1-\alpha)}$ then the null hypothesis is rejected, otherwise not.
- *Repeat/End*: if there are other candidates to test, repeat the statistic computation with the new candidate, otherwise stop.

4.2 UCB1

The *Upper Confidence Bound* algorithm is a multi-armed problem algorithm. For each arm, it computes a confidence bound and uses it for selecting the most promising candidate; these bounds are update during time. The pseudo-code of how the UCB1 works (considering Bernoulli distribution for the environment) is the following:

1. Play each arm one time.
2. At every time t play arm a_t such that:

$$a_t \leftarrow \underset{a \in A}{argmax} \left\{ \bar{x_a} + \sqrt{\frac{2log(t)}{n_a(t-1)}} \right\}$$

4.3 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

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

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

$$a_t = arg\max_{a \in A}\{\widetilde{\theta}_a\}$$

3. update the Beta distribution of arm a_t as

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

4.4 Sliding Window UCB1

This kind of algorithm introduces a sliding window that considers only the last τ samples to compute the upper confidence bound. This is the pseudo code of the UCB1 considering this fact:

1. Play each arm one time.

2. At every time t play arm a_t with $n_{a_t}(t-1,\tau) = 0$ if any, otherwise play arm such that:

$$a_t \leftarrow \underset{a \in A}{argmax} \left\{ \bar{x}_{a,\tau} + \sqrt{\frac{2log(t)}{n_a(t-1,\tau)}} \right\}$$

4.5 Sliding Window Thompson Sampling

This kind of algorithm introduces a sliding window that considers only the last τ samples for updating the beta distributions. This is the pseudo code of the TS considering this fact:

1. at every time t, for every arm a

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

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

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

3. update the Beta distribution of arm a_t as:

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

if $t > \tau : (\alpha_{a_t}, \beta_{a_t}) \leftarrow max\big\{(1,1), (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t}) - (x_{a_{t-\tau},t-\tau}, 1 - x_{a_{t-\tau},t-\tau})\big\}$

Chapter 5

Aggregate analysis

In this chapter focuses the attention on the application and analysis of algorithms applied to the same environment but with different settings. For this analysis we are going to use the aggregated demand curve.

5.1 Aggregated demand curve

This curve is computed as the weighted sum of the demands of the single classes, that we've introduced in the previous chapter *(See the section: 2.3 Phases and curves)*:

$$p_{agg}(i) = \sum_{c \in classes} p_c(i) * w_c \qquad \forall i \in prices$$

Where $p_{agg}(i)$ is the percentage of customers that would buy our product at price i. In our experiment we use these percentage as probabilities for the conversion rate.

5.2 K-testing

Because the Sequential A/B testing is not practical and difficult to perform in a non-stationary environment, we decided to consider only the first phase demand curve during all the time of the test. We perform the sequential A/B testing comparing the candidates 2 by 2 from the lower to the higher.
We decided to perform 100 experiments in order to have an average result (different experiments iterations might give slightly different results).

These are the value of accuracy and relaxing coefficient that we set before starting the Sequential A/B testing:

Parameter	Value
Significance level	0.005
Power level	0.85
Alternative hypothesis relaxing coefficient	0.05

During the computation of $\bar{x_1}$ and $\bar{x_2}$ for $\bar{y}$, if for some reason, one between $\bar{x_1}$ or $\bar{x_2}$ is 0, we directly use the probability associated to each candidate (respectively p_1 and p_2), otherwise we have a division by 0 in the computation of the statistic.

At the end of the experiment the Sequential A/B testing gave us that the average best-selling price for the first phase is 949 €, but since is not one of our candidates, we approximate it and we select as best price: 900 €.

5.3 MAB algorithms

Here we report the results that we have obtained from UCB1, Thompson Sampling algorithms and their Sliding window versions considering the candidates we have explained before *(See the section: 3.2 Candidates selection)*.

Setup:

- *Number of experiments*: 10000
- *Samples per day*: 20

5.3.1 UCB1

Here we report the reward and the regret of the UCB1 algorithm. As can be seen from the picture, the algorithm does not perform very well, especially because of the non-stationary environment that we have assumed. As a matter of fact, the algorithm can learn how to reach one optimum, but it is not able to handle the fact that this optimum can change. Moreover, it can be seen that in the first phase the algorithm is not able to well learn the optimum, before the environment changes. This is probably due to the quite pessimistic number of samples per day we have assumed in our analysis that are not enough for the UCB1 algorithm for learning the optimum.

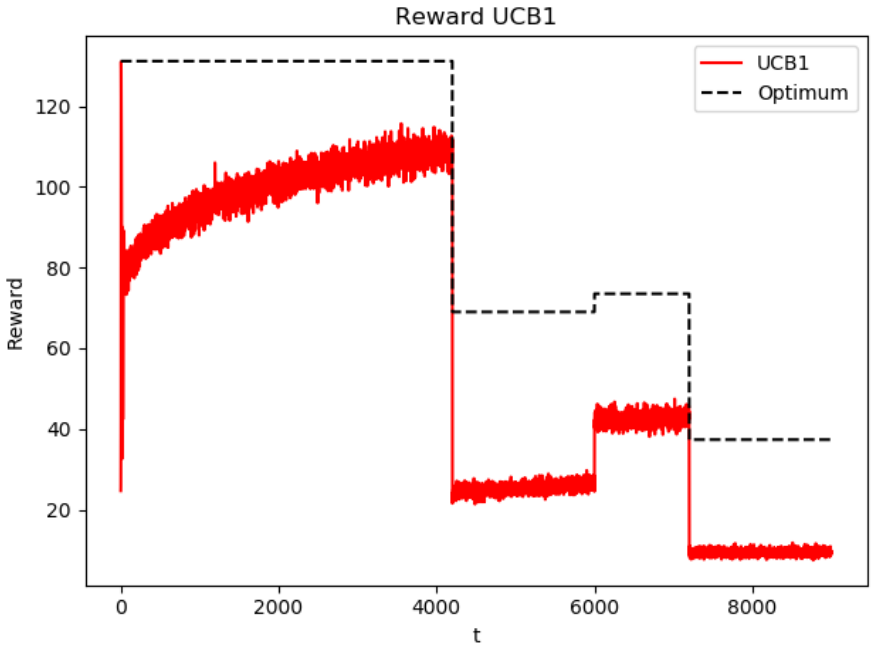

Figure 5.1: UCB1 reward

As explained from the following graph, the regret (which is obtained as difference between the collected reward and the optimum one day by day) in the first phase decreases gradually as the time flows. However, it's very high since the different between the reward and the optimum is huge also at the end of the first phase *(See the picture above)*. Moreover, when the environment changes (and consequently the optimum) the regret strongly increases since, as already explained, the algorithm does not perform well in a non-stationary environment.

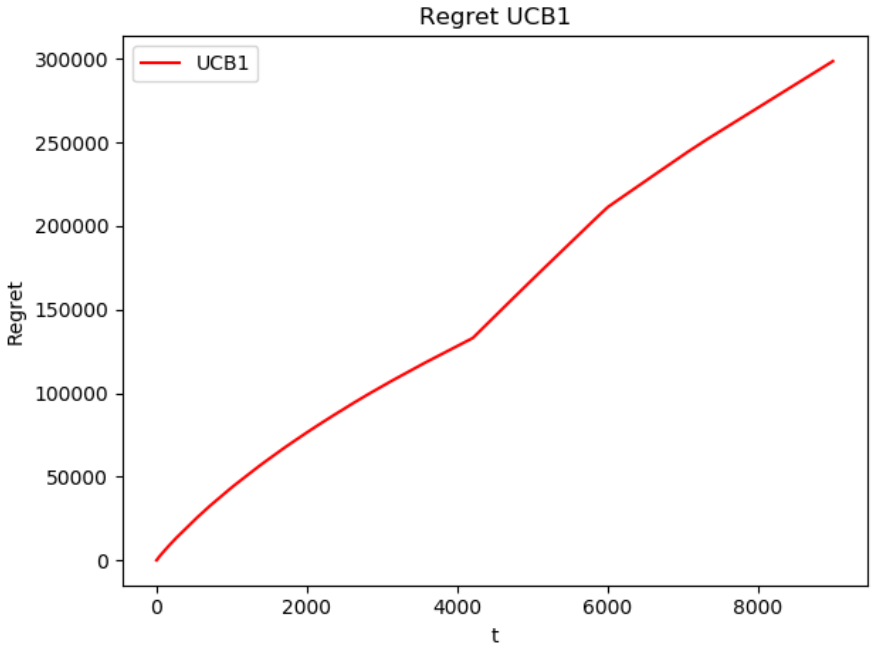

Figure 5.2: UCB1 regret

5.3.2 Thompson Sampling

Here we report the reward and the regret of the Thompson Sampling algorithm. In the first phase the algorithm is very close to learn the optimum, but then the environment changes and, since that moment, the algorithm perform badly because it is not able to handle optima changing.

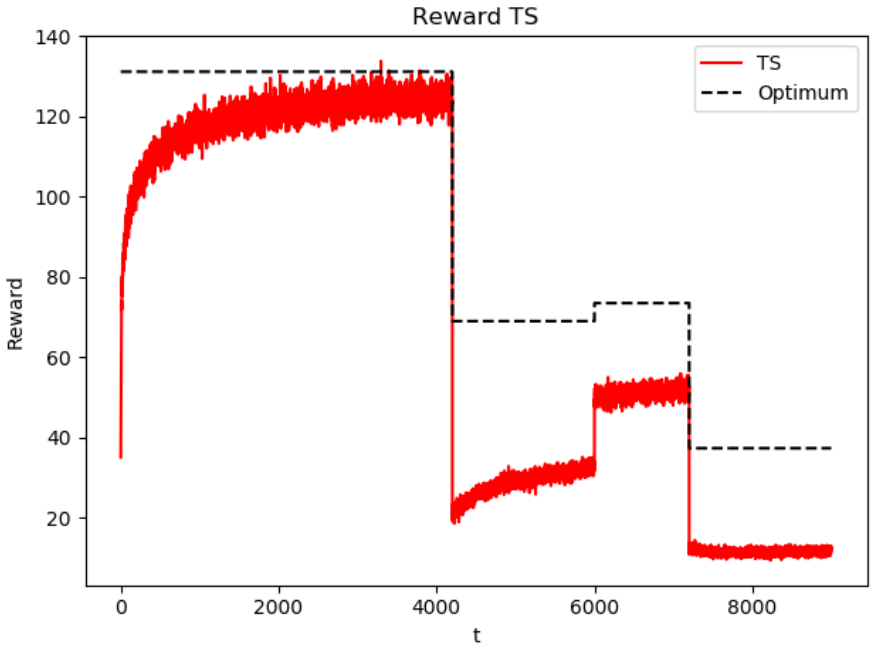

Figure 5.3: Thompson Sampling reward

The picture above describes the trend of the regret of the Thompson Sampling. At the beginning the regret decreases day by day due to the fact that the algorithm is learning well, but as soon as the environment changes, the regret strongly increases since, as already explained, the algorithm does not perform well in a non-stationary environment.

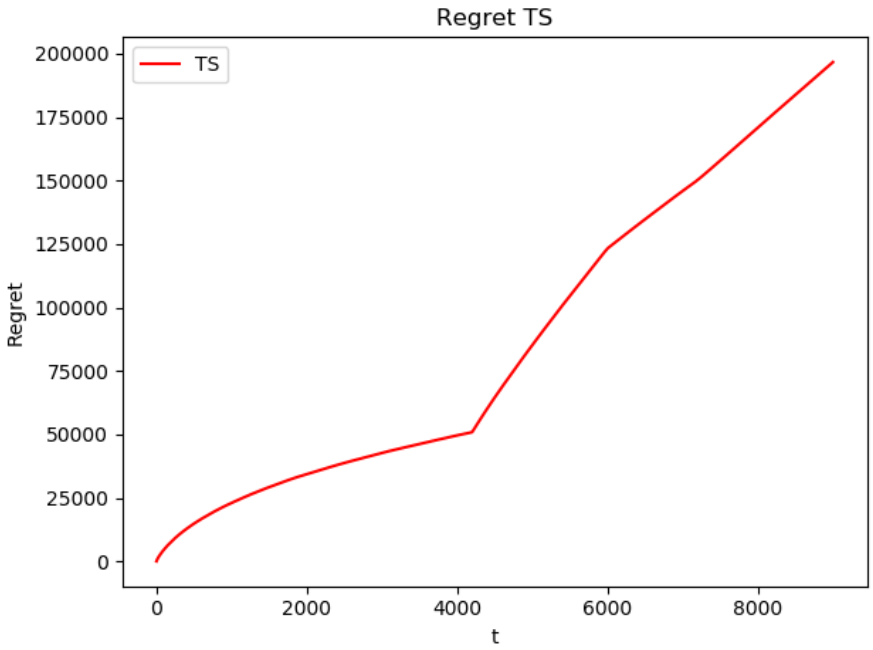

Figure 5.4: Thompson Sampling regret

Comparison

Here we report the comparison between the reward and the regret of the UCB1 and Thompson Sampling algorithms. As we can see, although both algorithm doesn't perform well in the non-stationary environment, the Thompson Sampling one operates better than the UCB1 one. Indeed, the regret of the first grows slowly w.r.t. the second one, which also reaches a final higher value. This is probably caused by the environment we have assumed; having few samples per day is enough for the Thompson Sampling algorithm for better learning the optimum, but this is not true for the UCB1 algorithm, that needs more.

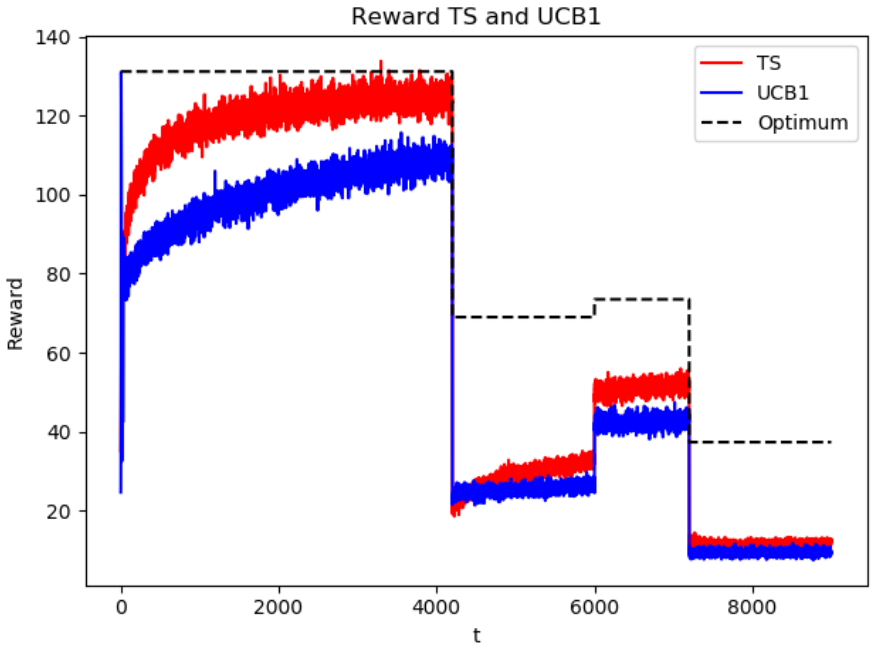

Figure 5.5: UCB1 and Thompson Sampling rewards

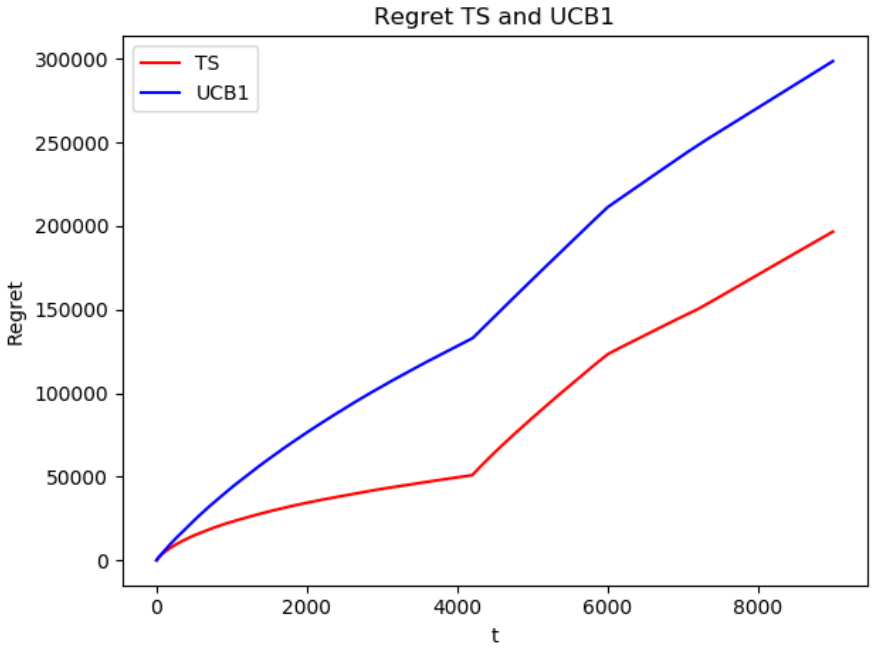

Figure 5.6: UCB1 and Thompson Sampling regrets

5.3.3 Sliding window UCB1

In this section we report how the performance of the Sliding window UCB1 varies as the length of the sliding window changes in terms of cumulative regret.
The length of the sliding window is computed as:

$$SW = K * \sqrt{T}$$

with T = Time horizon: 9000

and K = Proportionality coefficients: $\{0.1, 0.2, 0.5, 1, 10, 20, 50\}$

The variation of the regrets depending on the sliding window size is reported in the following picture. As can be seen, the algorithm doesn't perform better than the classical UCB1 with any of the tested window lengths. In the best case the regret is very similar to the UCB1 one (if window size equals to 9 or 4743). This happens because the classical UCB1 did not reach the optimum even in the first phase of the environment (due to the too few number of samples per day) and having introduced a sliding window further reduces the number of samples the algorithm needs to properly learn.

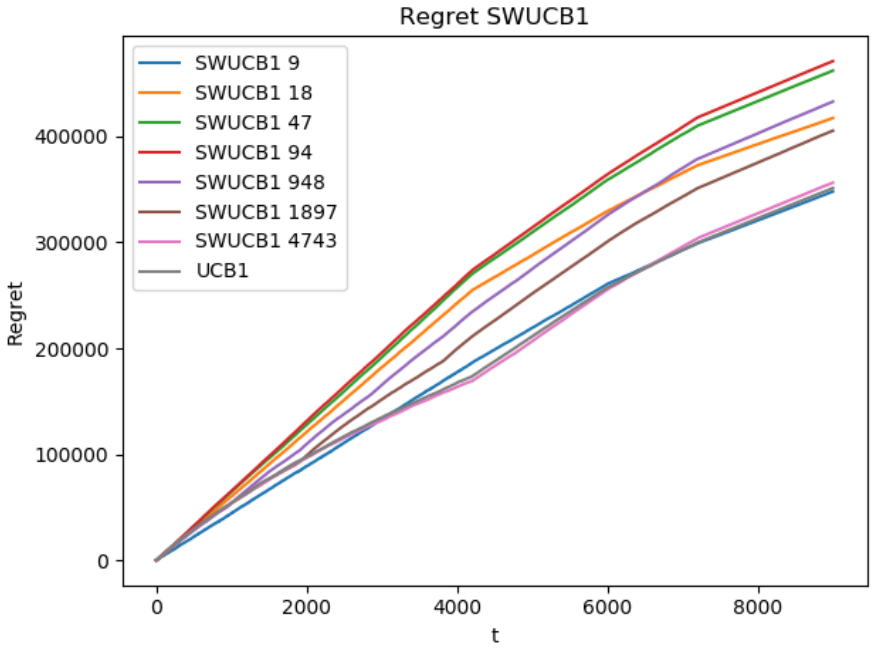

Figure 5.7: Sliding window UCB1 regrets with windows length

We decided to select a sliding window with length equals to 4743 since the regret is very close to the UCB1 one. We decided to discard the window with length equals to 9, since it is smaller than the number of candidates we set.

Here we report the trend of the reward and the regret of the Sliding window UCB1 with window size of 4743 samples. As can be seen from the following picture in the first the algorithm does not reach the optimum, similarly for what we already said for the classical UCB1. Despite of the introduction of the sliding window that should handle the problems of the non-stationary environment, the algorithm keeps to have a bad performance due to low number of samples that it needs for learning properly.

Figure 5.8: Sliding window UCB1 reward

The following picture shows the regret of the Sliding window UCB1. As can be seen, there is an increase of the regret after $t = 4743$ because the old samples are discarded from the sliding window and, since the upper bound associated to the candidate which old samples are discarded increases, the algorithm would choose it, even if it is not convenient.

Figure 5.9: Sliding window UCB1 reward

5.3.4 Sliding window Thompson Sampling

Here we report how the performance of the Sliding window Thompson Sampling varies as the length of the sliding window changes in terms of cumulative regret. The length of the window is computed as before (*5.3.3*).

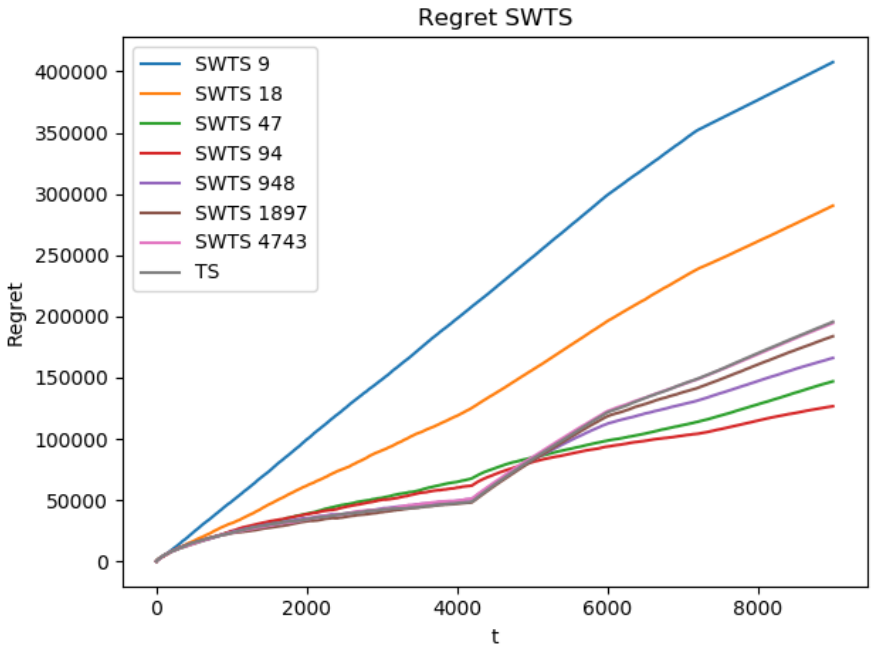

Figure 5.10: Sliding window Thompson Sampling regrets with windows length

As can be seen, the regret decreases faster than the classical Thompson Sampling algorithm if the sliding window has length equals to 94, therefore we decide to select this value for our analysis.

Here we report the reward and the regret of the Sliding window Thompson Sampling with window size of 94 samples. As can be seen from the following picture, the algorithm almost learn the optimum in the first phase, but, differently from its classical version, as soon the environment changes, the Sliding window Thompson Sampling is able to learn the changed optimum discarding (with the sliding window) the older samples.

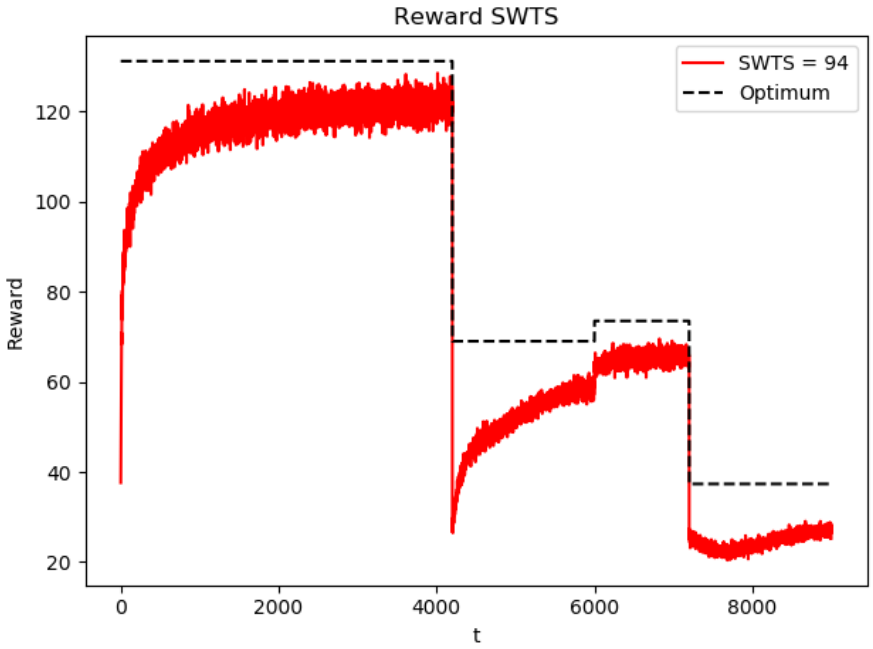

Figure 5.11: Sliding window Thompson Sampling reward

Th regret of the Sliding window Thompson Sampling as soon as the environment changes increases a little, but it decreases day by day thanks to the discarding of older samples.

Figure 5.12: Sliding window Thompson Sampling regret

Comparison

Here we report the comparison between the reward and the regret of the Sliding window UCB1, with window size = 4743 and the Sliding window Thompson Sampling with window size = 94. As can be seen from the pictures below the Sliding window Thompson Sampling perform strictly better than the Sliding window UCB1, which performance, as already said, are very similar to the classical UCB1 version. This behaviour can also be seen from the graphs of the regrets: the final value of the regret the Sliding window UCB1 is much greater than the Sliding window Thompson sampling one. As already said, this is probably caused by the pessimistic number of samples per day we set that is fine for the second algorithm but not for the first one.

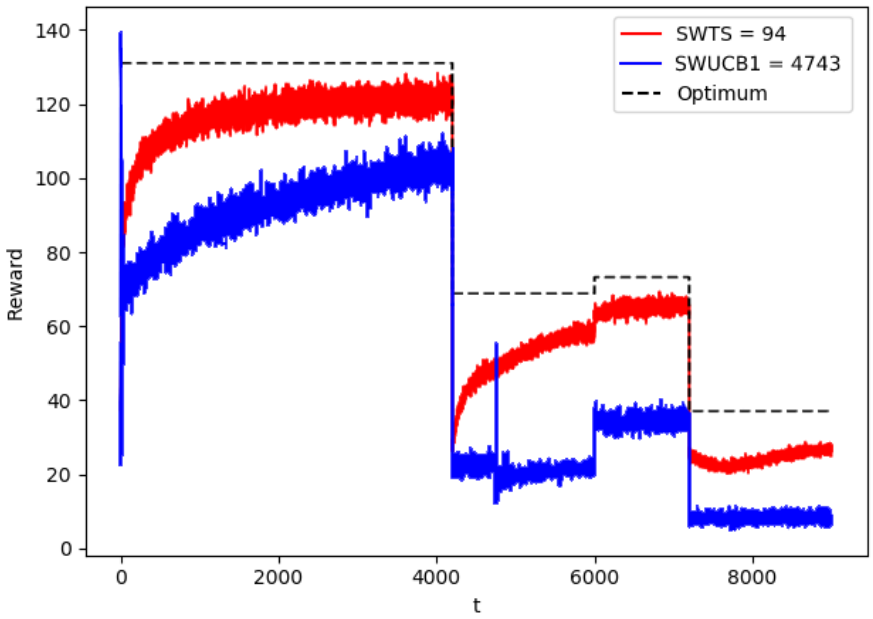

Figure 5.13: SW UCB1 and SW Thompson Sampling rewards

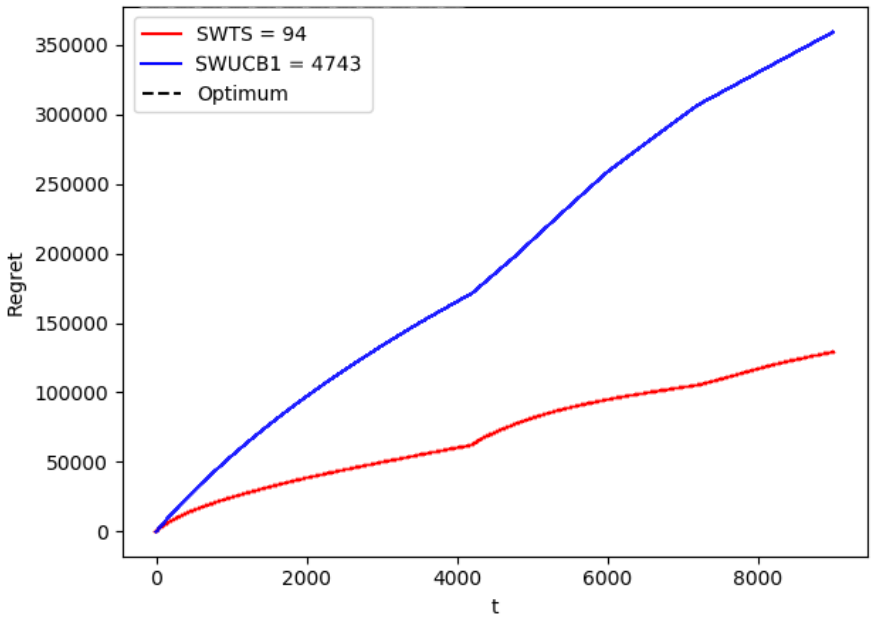

Figure 5.14: SW UCB1 and SW Thompson Sampling regrets

5.3.5 Normal vs Sliding window algorithms

In this section we compare the performance of the classical UCB1 and Thompson Sampling algorithms w.r.t. their sliding window versions in terms of rewards and regrets.

UCB1

As we can see from the following pictures the performance of both algorithms are very bad; moreover, the performance of the Sliding window UCB1 is worse than the classical version of the UCB1. This is probably because of the few samples per day that we set in our environment that are not enough, for both the algorithms, to learn properly the optimum. Moreover, the Sliding window UCB1, that usually performs better in a non-stationary environment, since the old samples are discarded is worse than the classical UCB1 and so it has an higher final regret.

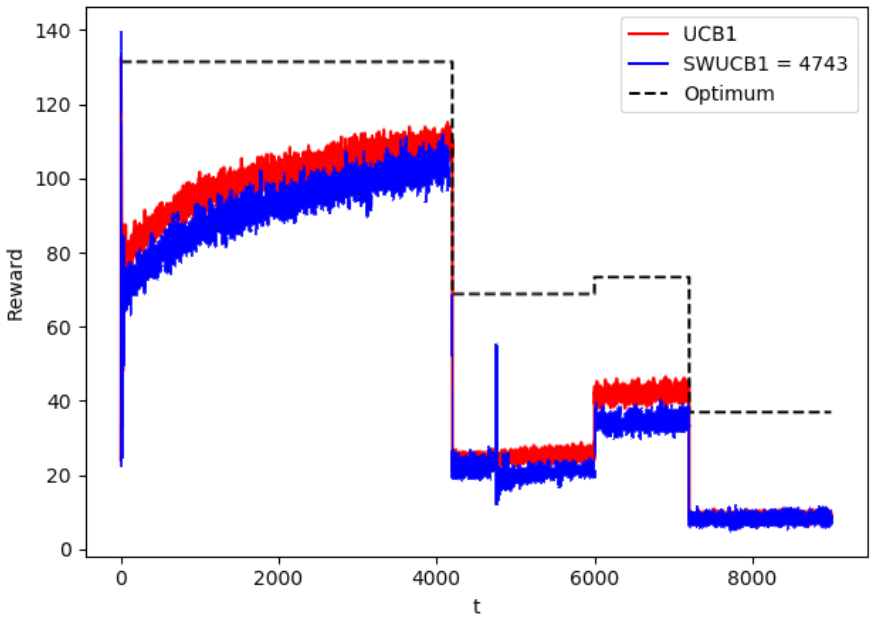

Figure 5.15: UCB1 and SWUCB1 rewards (Sliding window = 4743)

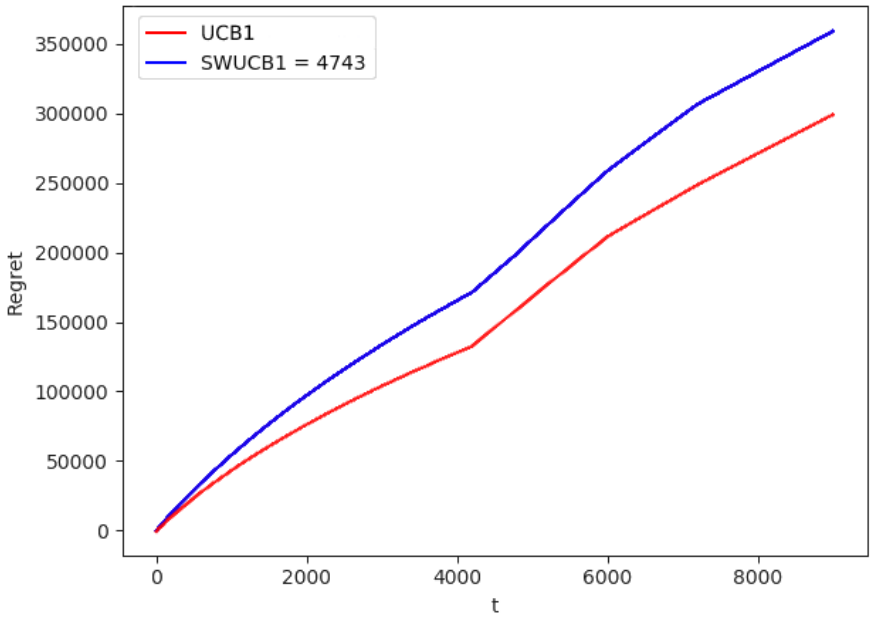

Figure 5.16: UCB1 and SWUCB1 regrets (Sliding window = 4743)

Thompson Sampling

As we can see from the following pictures, initially, in the first phase of the environment, having fixed the best sliding window length for the Sliding window Thompson Sampling (94 samples), the reward of the classical Thompson Sampling is a little bigger and its regret

decreases faster. This is probably caused by the number of samples used for learning the optimum: because of the window, the Sliding window Thompson Sampling has not enough samples for learning faster than the classical version of the algorithm. However, after the environment changes, the Sliding window Thompson Sampling perform much better that Thompson Sampling due to the fact that the older samples are discarded. Indeed, as we can see from the graphs, the reward of the first one in the following phases is bigger than the second one and its regret decreases much more quickly.

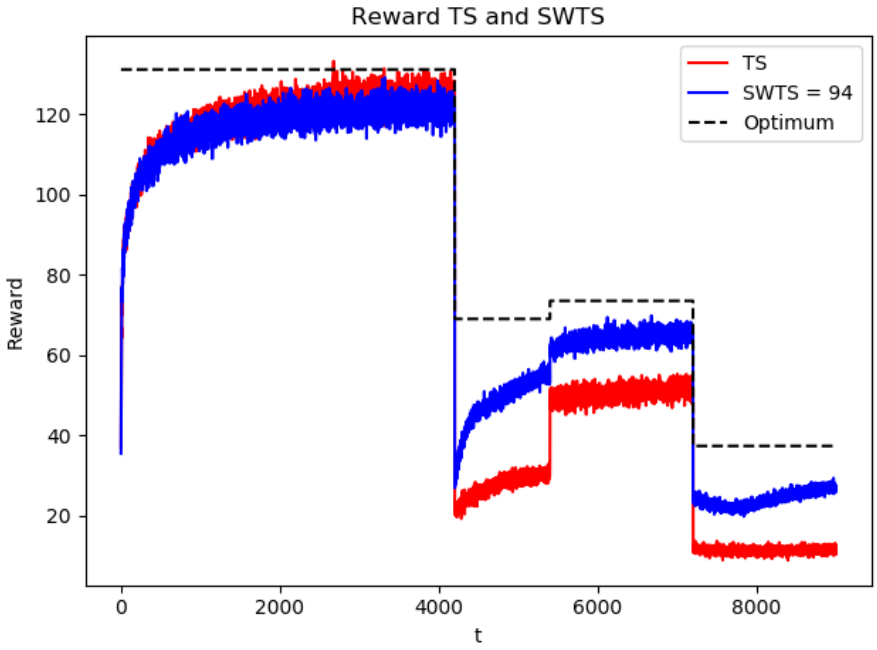

Figure 5.17: TS and SWTS rewards (Sliding window = 94)

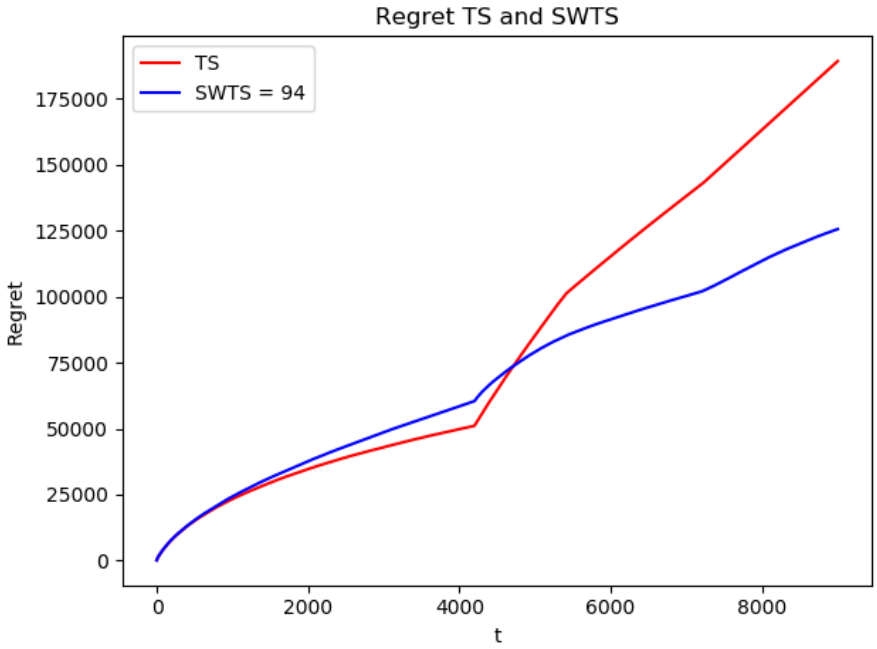

Figure 5.18: TS and SWTS regrets (Sliding window = 94)

Chapter 6

Disaggregation ad context identification

In this chapter we analyse the scenario in which there exist an algorithm that identifies the context; the goal of this analysis is to check whether at some point is better to disaggregate the aggregate curves and continue the experiment with this situation. The check is performed the first day of every week.

Setup:

- *Number of experiments*: 10000
- *Samples per day*: 20

6.1 Implementation

The learner used by the run-time disaggregation algorithm is in reality composed by 7 learners all of the same type (UCB1, Thompson Sampling...):

- *agg_learner*: it learns the aggregated demand curve;
- *cl0_learner*: it learns class 1 demand curve;
- *cl1_learner*: it learns class 2 demand curve;
- *cl2_learner*: it learns class 3 demand curve;
- *cl01_learner*: it learns the aggregated curve of class 1 and class 2;
- *cl02_learner*: it learns the aggregated curve of class 1 and class 3;
- *cl12_learner*: it learns the aggregated curve of class 2 and class 3;

In addition, the algorithm has some *Active Learners* that are a subset of the previously described learners that are used in that specific moment by the algorithm. At the beginning of the experiment the only active learner is the one that learns the aggregated demand curve and when the algorithm decides to disaggregate, this one will be

removed by the mentioned subset while others will be inserted (depending on the chosen disaggregation).

Important: in the active learners set there are *ALWAYS* learners that learn different classes and which union is the total classes ensemble. For example, *cl02_learner* and *cl01_learner* will not be together since they both learn class 0.

At the beginning of each week the algorithm checks if it is convenient to disaggregate or not, with a method called *check_aggregation* that takes in input the remaining possible aggregations combinations. I.e. if the active learners set contains only *agg_learner*, then the possible aggregations combinations will be: {*[cl01, cl2], [cl02, cl1], [cl12, cl0], [cl0, cl1, cl2]*}. For each of these possible aggregations 5 samples (that we know to which classes belong) are taken. After that, the expected reward of the learner associated to the class sample is computed and, finally, the average of the rewards of the aggregation is calculated. At the end the aggregation with the highest average reward is chosen. If in the active learners set there are the single class learners, these ones are kept until the end of the experiment, since no more disaggregations are possible.

Example: suppose to have the following as possible aggregations: {[cl01, cl2], [cl0, cl1, cl2]}. From the first aggregation we choose that the sample, according to the classes probabilities, comes from class 1. Since in this possible aggregation the learner for class1 is cl01, we compute the reward of it. We repeat this operation for all the 5 samples and then we compute the average reward. We perform this operation also for the other coalition, that, in our example, gave us a greater average reward. Given this the algorithm chooses this last aggregation as the best one.

6.2 UCB1

In this section we analyze the behaviour of the disaggregation algorithm with UCB1 as learner. It found out that, in average, is convenient to separate the aggregated curve after 1 week from the starting point. It also discovered that is convenient to perform a complete disaggregation in the 64% of the cases and a partial disaggregation followed by a complete one in the remaining 36% of the cases (this is probably because that the models are not trained enough for performing immediately a complete disaggregation). The algorithm always suggest to disaggregate.

Here we report the reward and the regret of the UCB1 algorithm when the disaggregation of the aggregated demand curve is performed after 1 week. As can be seen by the following pictures, the reward, in the first phase of the environment, is higher than the classical UCB1 without disaggregation *(See the picture 5.1)* and this happens because now the optimum is bigger. This also explain why the regret doesn't decrease too much in the same phase. As soon as the environment changes its regret increases, since, as we already explained in the previous chapter, the UCB1 algorithm doesn't perform well.

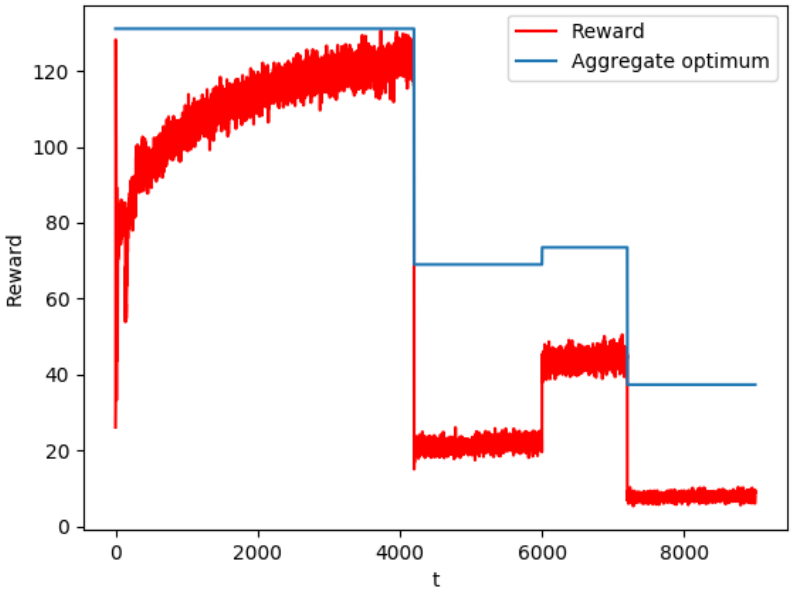

Figure 6.1: UCB1 reward

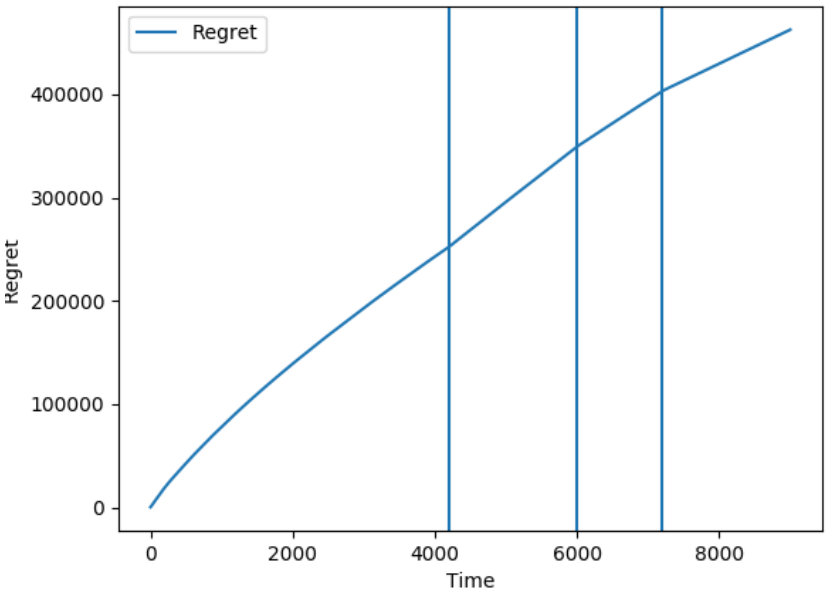

Figure 6.2: UCB1 regret

6.3 Thompson Sampling

Among all the experiment we performed having as learner the Thompson Sampling, the algorithm found out that is convenient to perform a complete disaggregation of the aggre-

gated demand curve in the 32% of the cases, while is convenient to have firstly a partial disaggregation and then a complete disaggregation in the 68% of the cases (this is due to the fact that the number of samples for performing immediately a complete disaggregation are not enough and the models are not consequently trained enough). In both the cases the algorithm perform this operation after 1 or 2 weeks. There are no cases in which is convenient to keep the aggregated curve, without performing any kind of disaggregation.

Here we report the average trend of the reward and the regret of the Thompson Sampling algorithm with the disaggregation performed after 1 or 2 weeks from the starting point. As can be seen from the pictures above the algorithm gain more performing the disaggregation of the demand curve (the 'new' optimum is higher) in the first phase, but it still have some problems when the environment changes. The regret decreases day by day at the beginning with the new optimum, but it start to increase as soon as the environment changes.

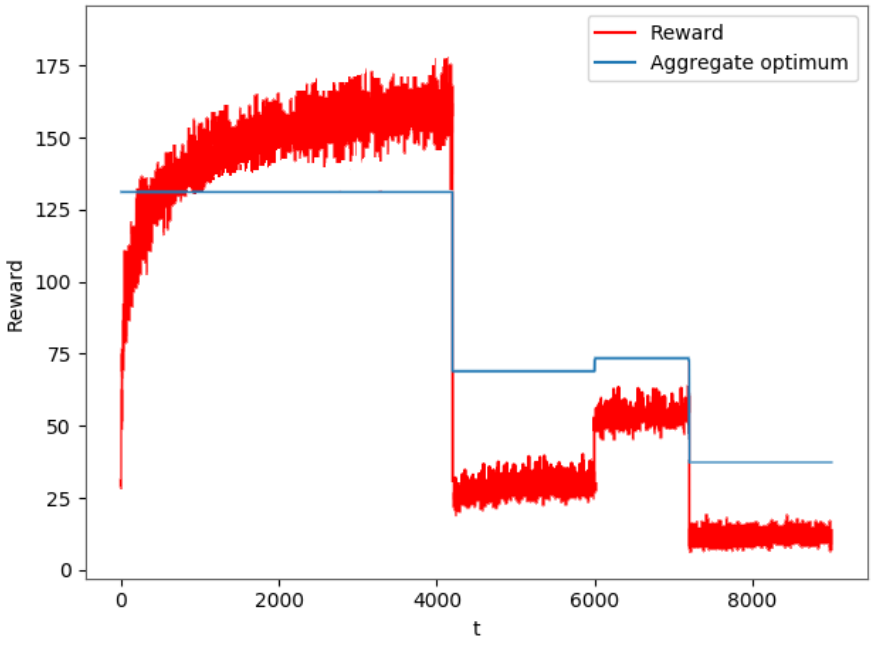

Figure 6.3: Thompson Sampling reward

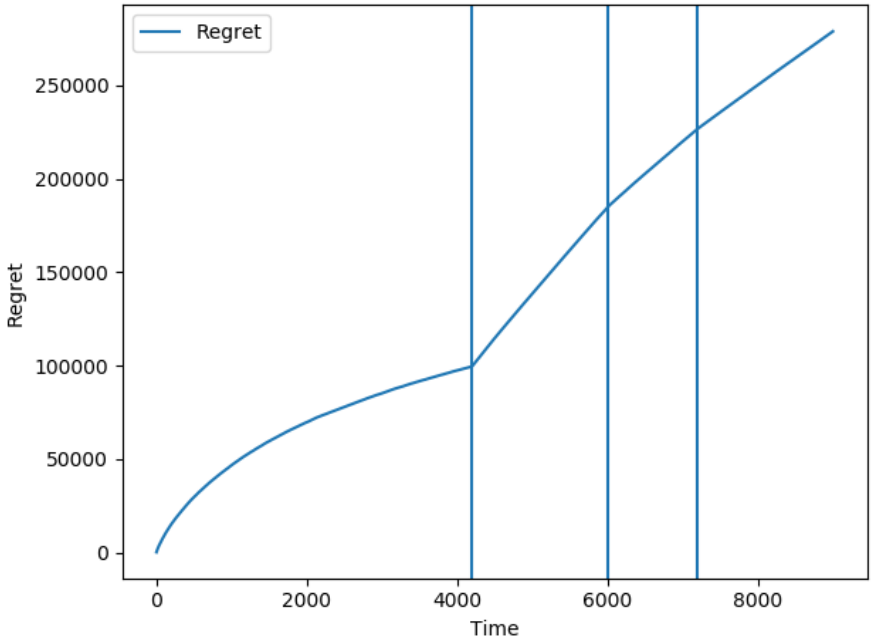

Figure 6.4: Thompson Sampling regret

6.4 Sliding window UCB1

In this section we discuss the behaviour of the disaggregation algorithm with the Sliding window UCB1 with a window of length equals to 4743. Similarly to the previous cases the algorithm suggest to disaggregate the demand curve after 1 weeks. In the 21% of the case the algorithm found out that is convenient to perform a partial decomposition and the a complete one, while, in the remaining 79% of the cases, is better to perform directly a complete disaggregation. Also with the Sliding window UCB1 there are no cases in which the disaggregation algorithm suggest to keep the aggregated demand curve until the end.

Here we report the reward and the regret of the Sliding window UCB1 (window length = 4743). As can be seen from the following pictures, the algorithm learns better than the algorithm without disaggregation *(See the picture 5.8)*, but is still far, as can be seen by the trend of the regret, from the 'new' optimum. Unfortunately, the introduction of the sliding window that should have improved the performance in the non-stationary environment, does not increase the performance, due to the too few number of samples.

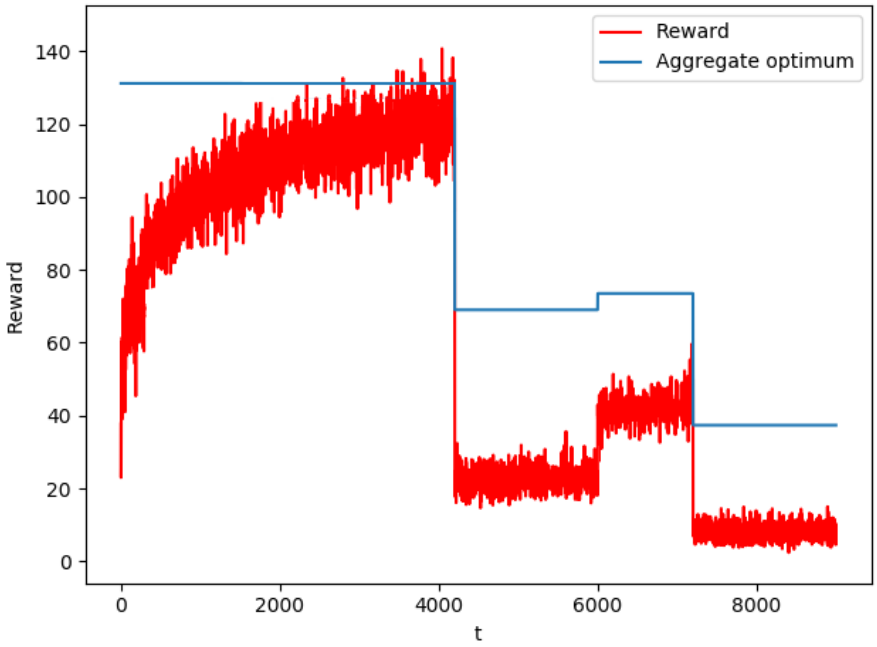

Figure 6.5: Sliding window UCB1 reward

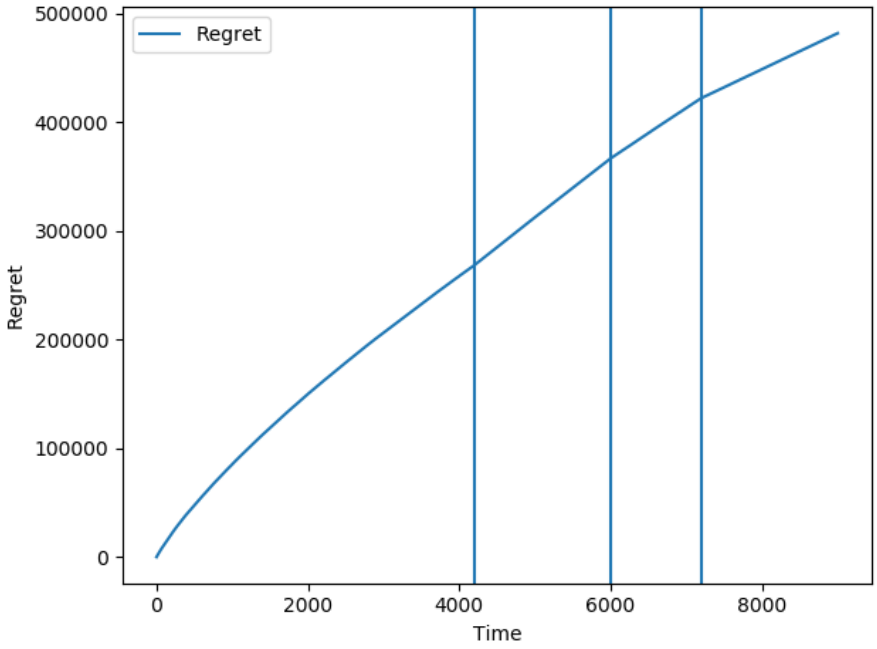

Figure 6.6: Sliding window UCB1 regret

6.5 Sliding window Thompson Sampling

In this section we discuss the behaviour of the disaggregation algorithm with the Sliding window Thompson Sampling with a window of length equals to 94. Similarly to the

previous cases the algorithm suggest to disaggregate the demand curve after 1 or 2 weeks. In the 76% of the case it assert that is convenient to perform a partial decomposition and the a complete one, while, in the remaining 24% of the cases, to perform directly a complete disaggregation. Also here there are no cases in which the disaggregation algorithm suggest to keep the aggregated demand curve.

The following pictures report the trend of the reward and the regret of the Sliding window Thompson Sampling algorithm (window=9) when the disaggregation is performed. Also here, in the first phase of the environment, the algorithm is able to gain more than the version without disaggregation, but, thanks to the sliding window, now is able to learn better also when the environment changes and that's why the final regret is lower than the regret of the disaggregation algorithm with the Thompson Sampling.

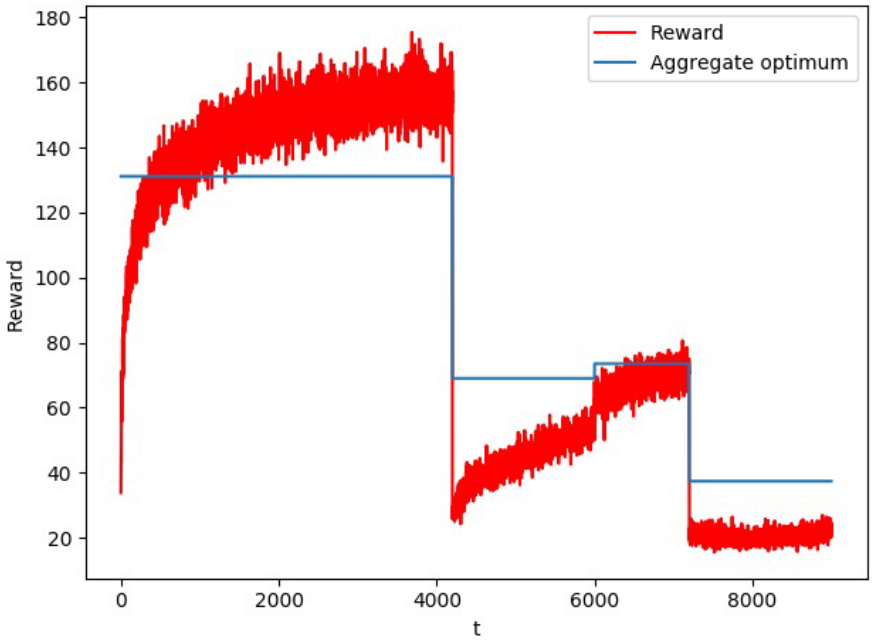

Figure 6.7: Sliding window Thompson Sampling reward

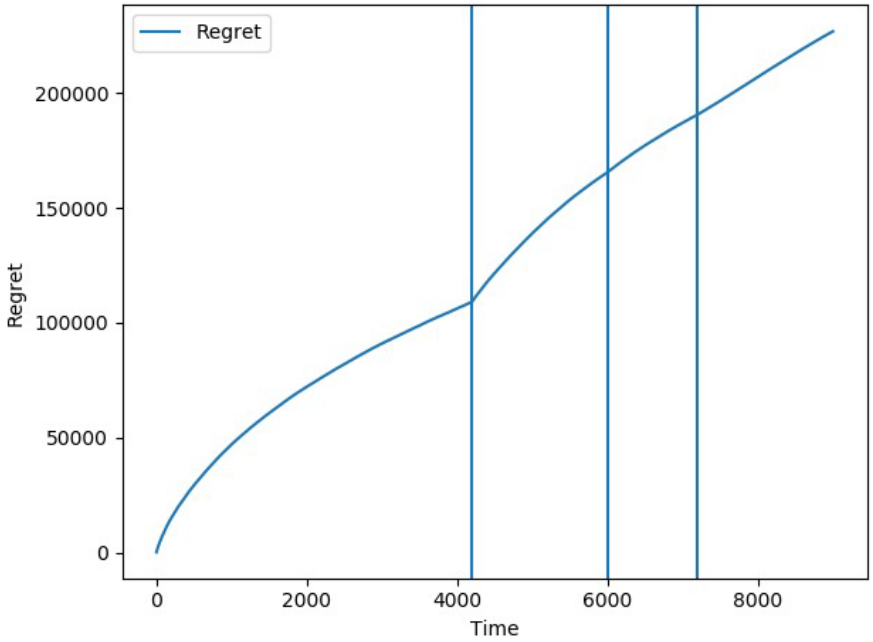

Figure 6.8: Sliding window Thompson Sampling regret

Chapter 7

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.
- *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 2.3 are results of the matplotlib modules adopted.