POLITECNICO
MILANO 1863

Data Intelligence Application

Online Pricing

Bonfanti Luca, Crippa Dominic, Krasniqi Filip, Pina Alessandro

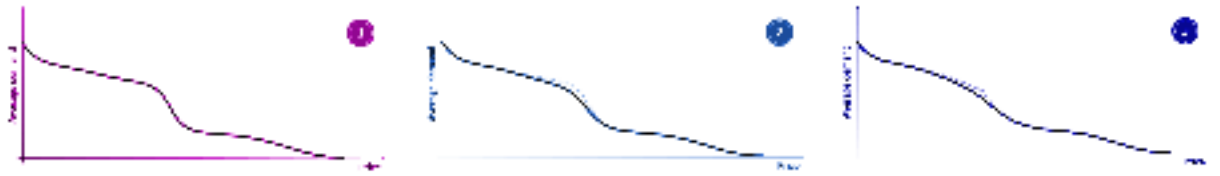
December 16th, 2019

Abstract

Online pricing is a problem subject to many factors that can affect its complexity.
Such a problem consists in choosing, given a demand curve, the selling price to maximize profit given by the sale of a certain product.

In a context in which the demand curve is unknown, such a function can be learned by an algorithm that is able to compute it for defined values of price.
The complexity of the problem depends on how this function varies. Usually, the demand curve is not only a function of the price, but also of the time (**non stationary environments**).

A common formalization of this problem has a distinction between **abrupt changes** and **smooth changes**.
The first one describes how the function varies between big intervals of time named *phases* (e.g. seasons), identified by unknown breakpoints.
The second one describes how the function varies inside a phase.



Example of smooth change in a monotonically decreasing demand curve

Together with this aspect (that literally introduces a demand curve as a function depending on two variables), we also need to consider the presence of contexts.

A **context** identifies how different classes of users are combined together into **subcontexts**, each of which is described by a demand curve that follows the previously described characteristics.
Being aware of this issue, an algorithm that selects the best price to maximize the profit should do the same by also considering the possible context alternatives.

Profit maximization would be a problem of selecting not only the best price, but also the best context, i.e., how to group the classes of users.

The **goal** of this report is to show how we formalized the problem and applied MAB algorithms to find the best setting, while showing results in terms of regret in two different scenarios: with and without context generation.

Environment

Description of the product (question 1)

Smartbands (aka fitness bands or fitness trackers) are a cheap solution for people who cannot afford a smartwatch or do not want to wear it in particular conditions.

They are also a common product on the internet, and we considered them to be affected by both the time and user information factors.

These products are commonly used to track sport activities, heart rate, sleeping activity, etc.

We are working on the following **assumptions**:

- **Monopoly**.
- Maximization of the **revenue** equals maximization of the **profit**.
  The revenue is computed by multiplying the demand *D(p)* by the price *p*.
  This assumption holds for example in case the difference *D(p)* - *C(p)* is constant.
- **Users** are distributed with a weighted uniform distribution that varies in time.

Users: definition, disaggregate curve, aggregate curve (question 2)

We identified **three classes of users**, briefly explained in the following table.

Class	Place	Age	Occupation	Hobby	D(p)
User 1 U_1	**Lombardia**	**25-30**	**Business**	**Movies, TV series, tennis**	$D_1(p) = 12 * e^{-0.002p}$
User 2 U_2	**Milano**	**15-25**	**Student**	**Reading, group sports**	$D_2(p) = 50 * e^{-0.009p}$
User 3 U_3	**Barzanò**	**20-40**	**Factory Worker**	**Playing soccer, movies**	$D_3(p) = 16 * e^{-0.003p}$

The table contains features that justify the function $D_u(p)$.
The function $D_u(p)$ visible in the table represents the demand curve **in winter**.

First, let's consider the meaning of such a function.

We formalized the demand curve of a certain class of users u as:

$$D_u(p) = D_u * e^{-\alpha_u p},\ being\ \alpha_u\ the\ slope,\ D_u\ the\ maximum\ demand,\ D_u(p)\ monotonically\ decreasing\ function\ of\ p$$

Given these disaggregated curves, the **aggregated curve** is computed by summing, for each value of price, the corresponding demand (e.g. Deriving a Market Demand Curve).

In addition, we define a **probability distribution** for each user.
This probability distribution varies over time to simulate the fact that the user interests change inside a phase, and it is relevant when we want to consider the context generation.
We consider this to be the main factor that characterizes the smooth changes.
In details, fixed a phase, we compute the probability of each user u as the ratio $P_u(t) = \frac{N_u(t)}{\sum_u^U N_u(t)}$,

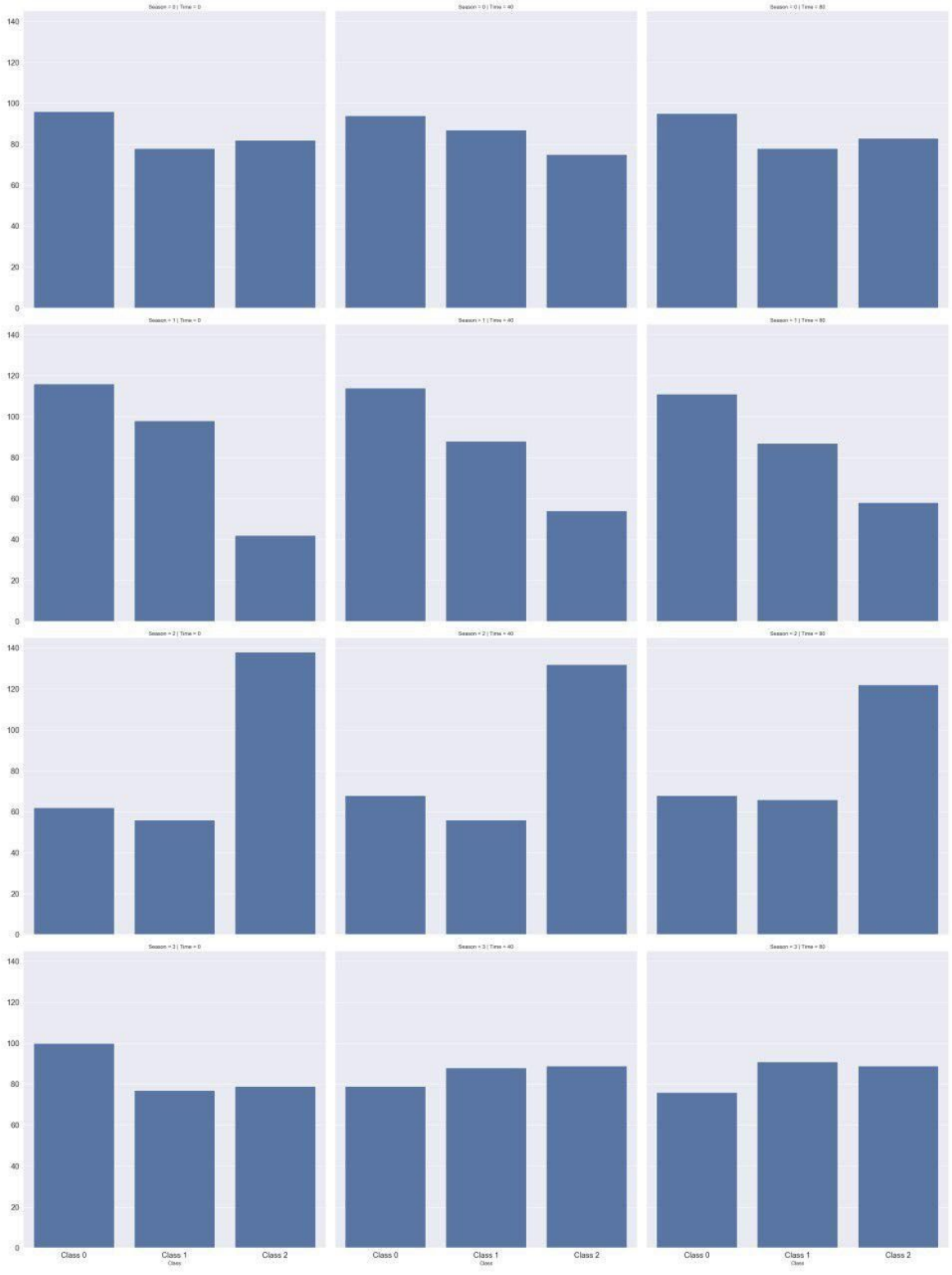
being $N_u(t)$ the number of users of class u, and U the set of distinct classes of users.
Different seasons are associated to different values of α_u, D_u for every user, as well as to different probability distributions (as shown in the probability distribution plot).

Explanation of the parameters D_u, α_u

- D_u: the higher this parameter, the higher the maximum value for the demand, obtainable for low value of price being that we want $D(p)$ to monotonically decrease wrt price p
- α_u: the higher this parameter, the more we are representing users willing to pay low prices

Eg: Characterization of $N_u(t)$ for winter season

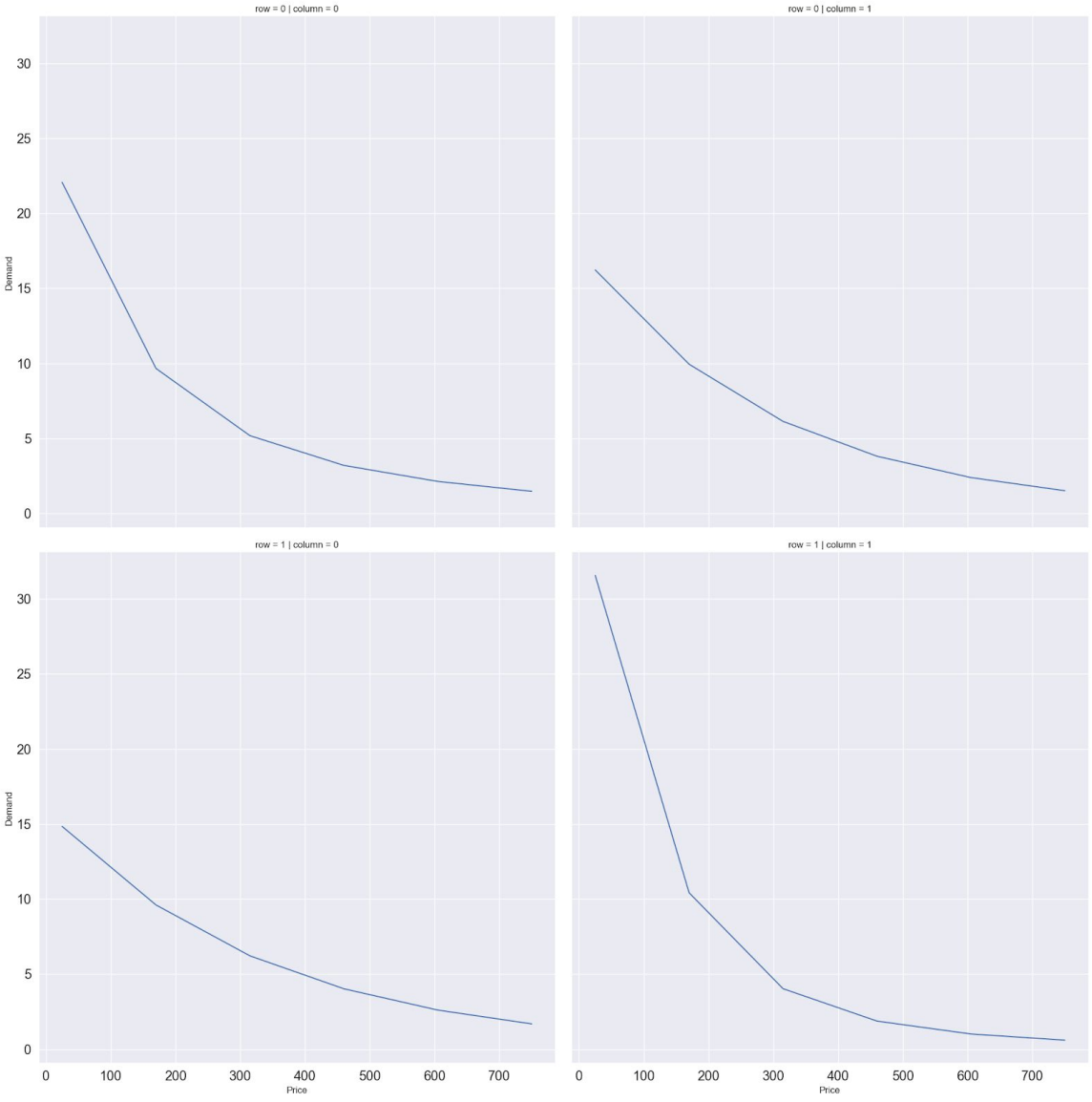
- $N_1(t) = 360 - t\ modS$
- $N_2(t) = 720 - 2 * t\ modS$
- $N_3(t) = 1080 - 3 * t\ modS$, $being\ t\ [0,\ T]\ the\ day,\ T = 365\ the\ horizon,\ S\ the\ season\ length$

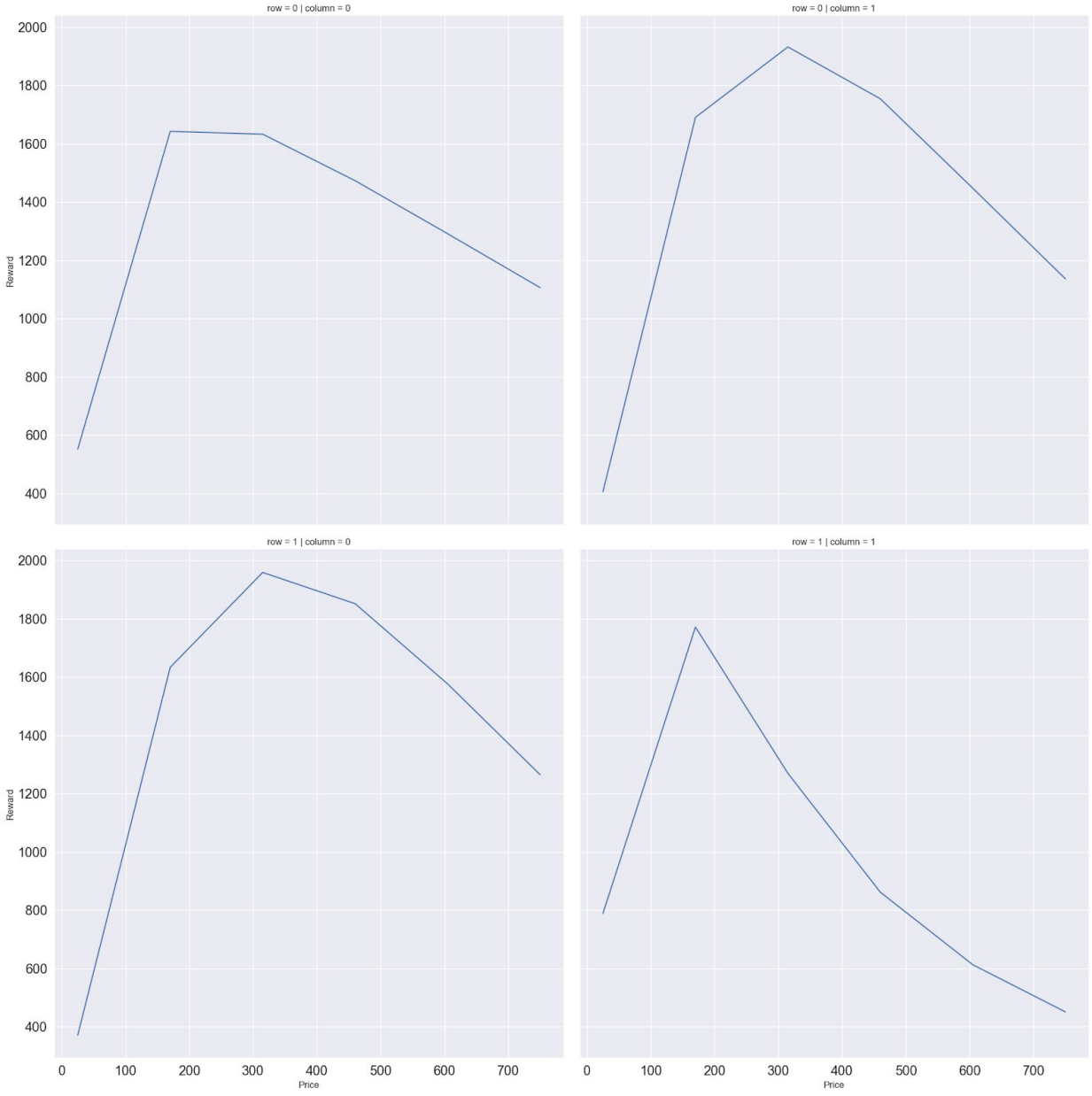


Resulting probability distribution of users.
Rows represent seasons, columns the days of the considered season (we chose 0, 40, 80, representing beginning, middle and end).
Each plot represents the countplot of the classes of users

Aggregated curve $D(p,\ t)$

Given the previously defined demand curves, in case we don't exploit the knowledge we have on the users (i.e., no context generation), here are our aggregate demand curves for first day of every season:





Demands and corresponding rewards for every season.

Among arms {0, 1, 2, 3, 4, 5}, the best arms for each season happen to be, respectively

{1, 2, 2, 1},

Now that the we have clarified the meaning of the functions parameters, we justify these values as follows:

- A **class 1 user**
    - is willing to pay relatively high prices, so the demand starts from lower value (15) and it doesn't decrease much, due to lower α than the other two cases;
    - has more purchasing power, so the number of users changes less inside a phase;
- A **class 2 user**
    - is willing to pay low prices (because they are students)
- A **class 3 user**
    - is willing to buy the smartband for lower prices than user 1, but higher than 2
- During **winter**
    - All the three classes of user have uniform distribution during this season, they want to buy the product to make Christmas gifts.
- During **spring**
    - Users of class 3 are initially not so interested in the product, but as the season advances their will to start practising a new sport increases, therefore their interest in the product rises.
    - Users of class 1 and 2 are sporty people so they continue to buy the product since the weather starts to push them to practice sports. Their willingness decreases a little during the season.
- During **summer**
    - Users of class 3 start to buy the item, having seen the benefits of the product. The beginning of this season is characterized by a high amount of users of this class buying the product
    - Users of class 1 and 2 remain quite conservative during this season.
- During **autumn**
    - Users of class 2 and 3 are very similar during this season, they increase their willingness a little as we get close to the Black Friday period.
    - Users of class 1 are quite strange, they want to buy the product at the beginning of the season, while their interest decreases towards the end.

Adding noise

The learning algorithm must not only find the best price s.t. the reward (profit) is maximized.
It must do so while also learning the demand curve(s) and adapting to changes.

This is done by discretizing the price domain (the **arms**) and by learning the corresponding values of the demand.
At each timestep, a batch of samples is given to the learner, that will use them to find out the value of the demand for that arm.
The **samples** are subject to a Gaussian Noise that follow a distribution $N(0, ^2)$.
We consider three values of to show the potential of the algorithm, i.e., $\sigma \ \{0.1, \ 2, \ 3\}$.

Definition of horizon, computation of discrete values of price (question 3)

We define a time horizon T of approximately a year (T = 364), and we compute the number of discrete values of price (i.e., the number of arms). This value is computed as follows:

```
n_arms = math.ceil(math.pow(T * math.log(T, 10), 0.25))
```

That is, $num_{arms} = T * \sqrt[4]{log_{10} T}$

K-testing, UCB1, TS, SW-UCB1, SW-TS on the aggregated curve (question 4)

Introduction

We will briefly introduce each algorithm and we will explain the results obtained.

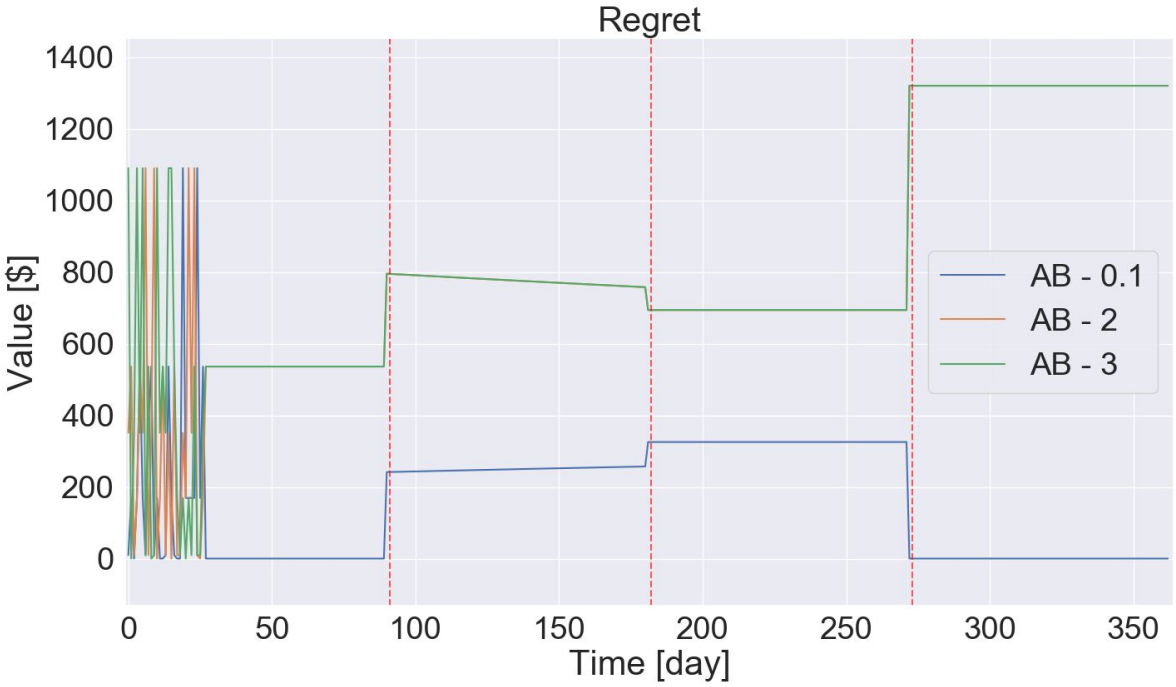
For each learner we will insert three plots containing three curves.
The **plots** represent the evolution over time of (in order): *regret*, *rewards*, *cumulative regret*.
The **curves** represent the results when increasing the noise ( ).

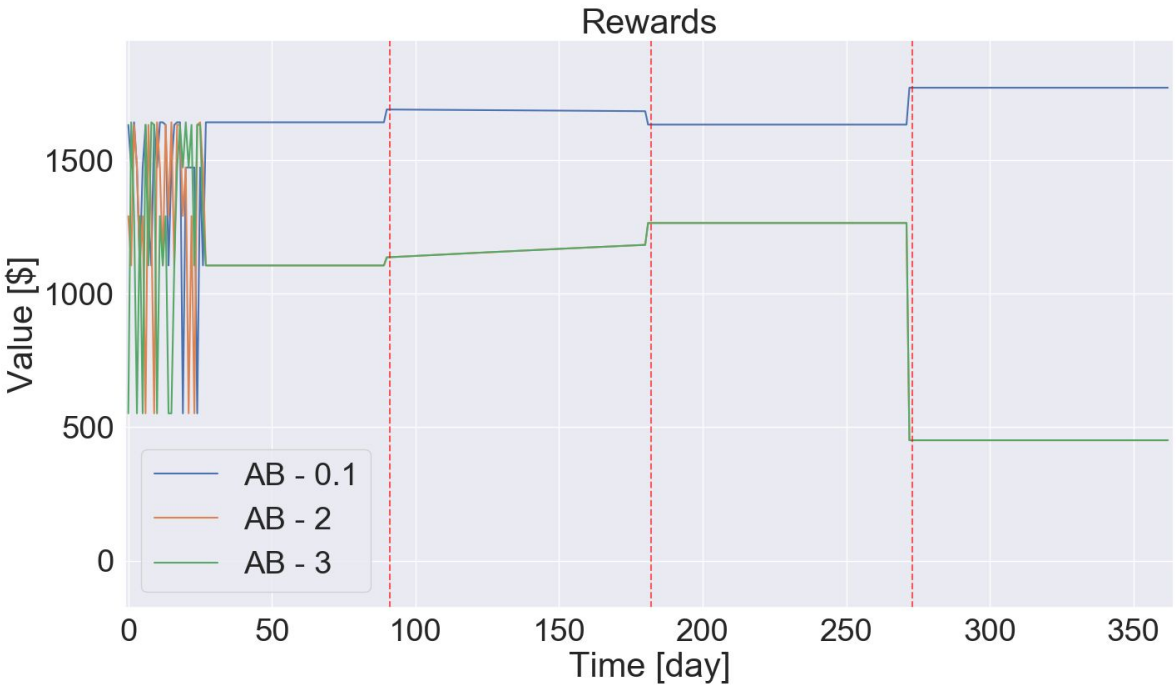
K-testing

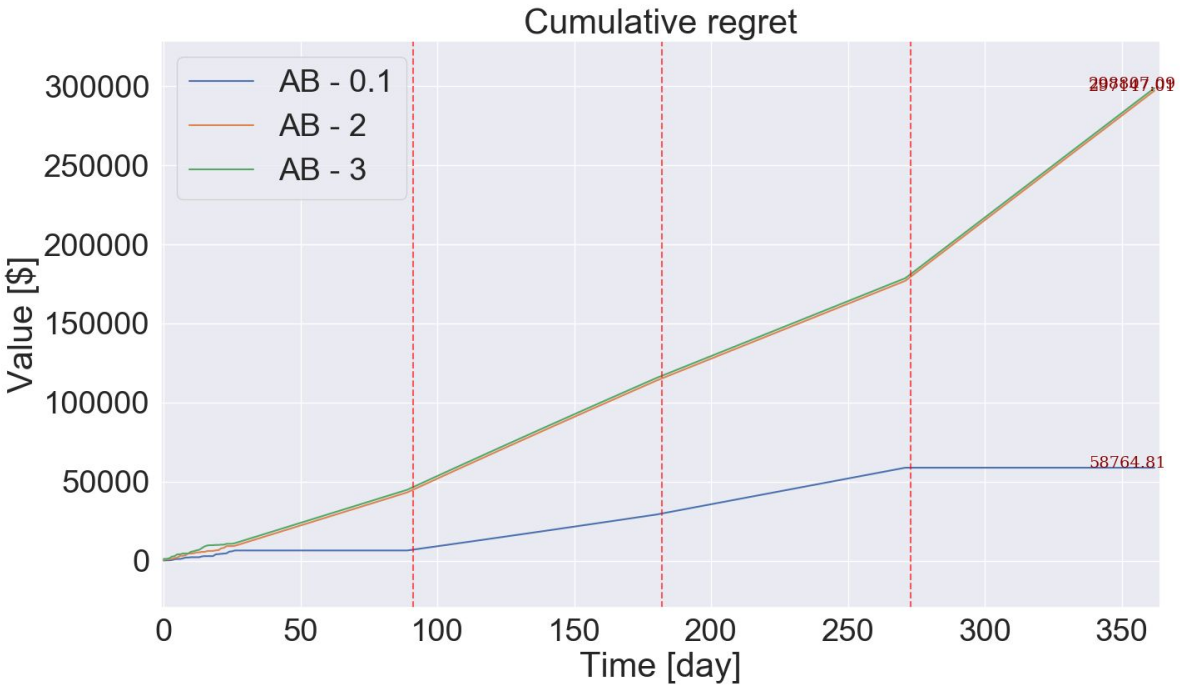
This statistical approach consists of fixing a priori E, corresponding to the timestep t when the algorithm stops exploring and starts exploiting.

The exploration phase consists in understanding which strategy is the best one to apply during the exploitation phase, and it is done by drawing values from each option and by building a hypothesis test.

This implementation can be done in many ways. We applied **Sequential AB testing**.


Regret
1400
1200
1000
800
600
400
200
0
Value [$]
0
50
100
150
200
250
300
350
Time [day]
AB - 0.1
AB - 2
AB - 3


Rewards
1500
1000
500
0
Value [$]
0
50
100
150
200
250
300
350
Time [day]
AB - 0.1
AB - 2
AB - 3



Conclusion

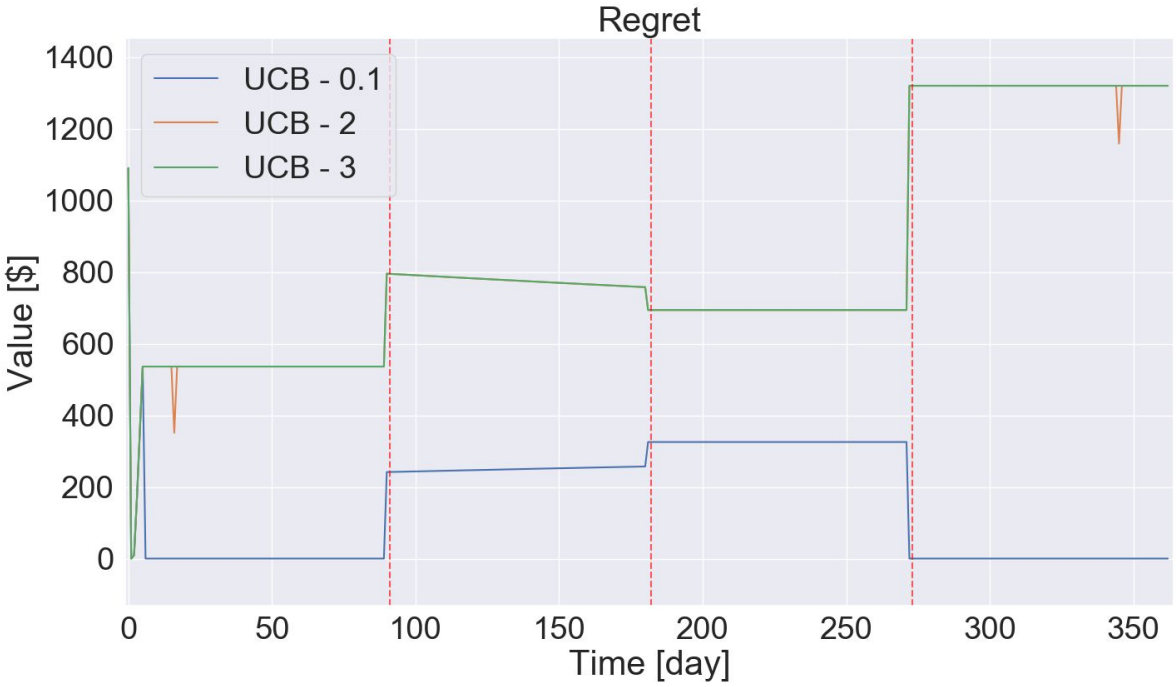
Sequential AB testing, T = 365, E = 28.
The learner, after a random exploration, learns the optimal solution only for $\sigma = 0.1, 2$ for the first season.
The abrupt change is critical as **the algorithm does not adapt**.

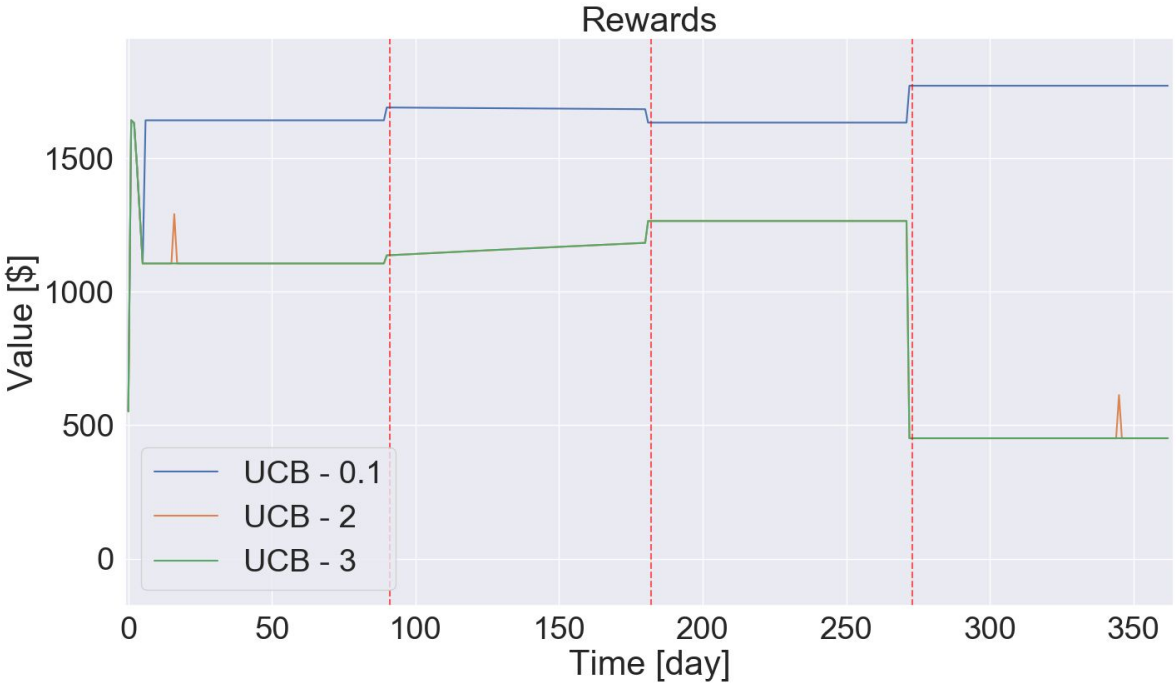
Upper Confidence Bound UCB1

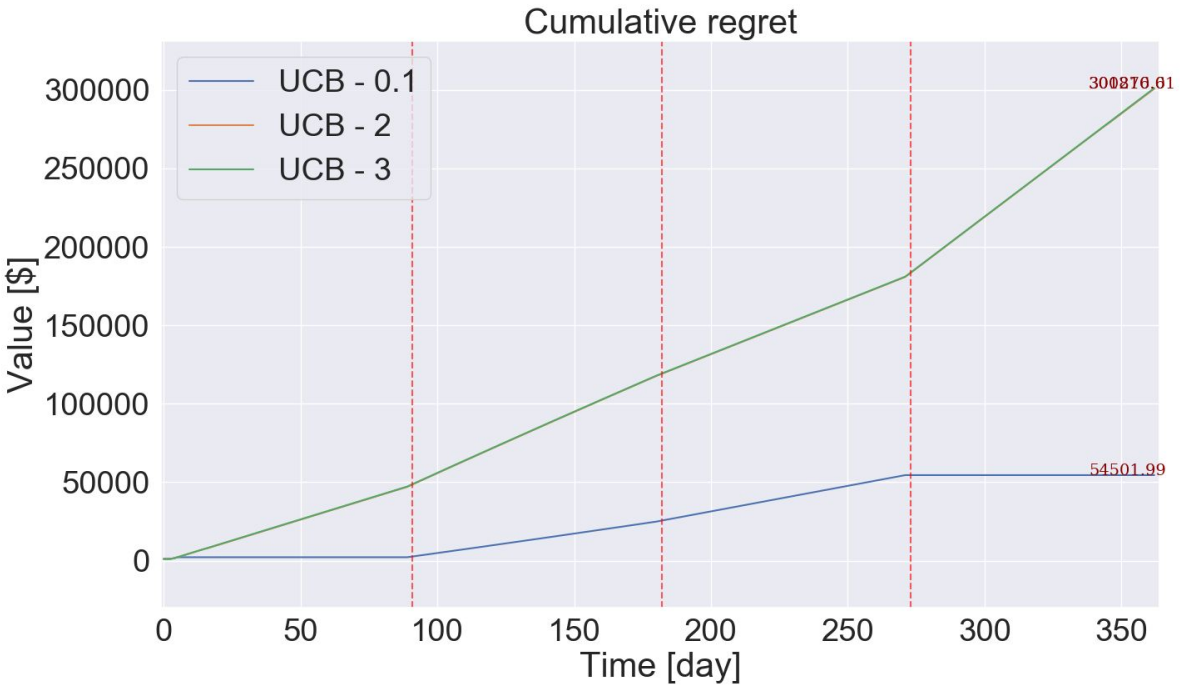
UCB1 is a MAB algorithm that maximizes the reward by selecting the more promising arm.

It does that by estimating not only the expected reward for that arm, but also an upper bound UB.
At time t, it selects **the arm having the highest upper bound**.
At each iteration the bound of the selected arm drops.

$$UB(arm) = expected(arm) + \sqrt{\frac{2logt}{n}},\ arm = argmax_{arm}(UB).$$


Regret
UCB - 0.1
UCB - 2
UCB - 3
Value [$]
Time [day]
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300
350


Rewards
UCB - 0.1
UCB - 2
UCB - 3
Value [$]
Time [day]
0
500
1000
1500
0
50
100
150
200
250
300
350



Conclusion

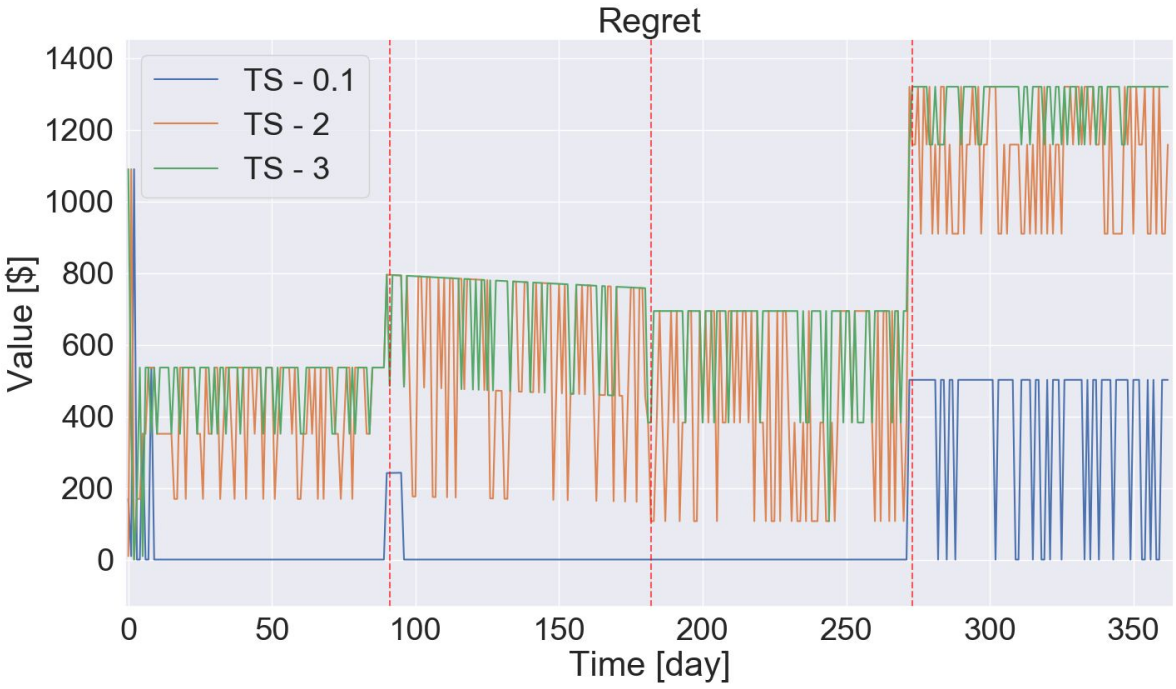
UCB1, T = 365.
The exploration lasts the time required to the algorithm to reach convergence to the optimal arm.
Such a solution is found only for the first two cases of noise and it is better than K-testing.
The abrupt change is critical as **the algorithm does not adapt**.

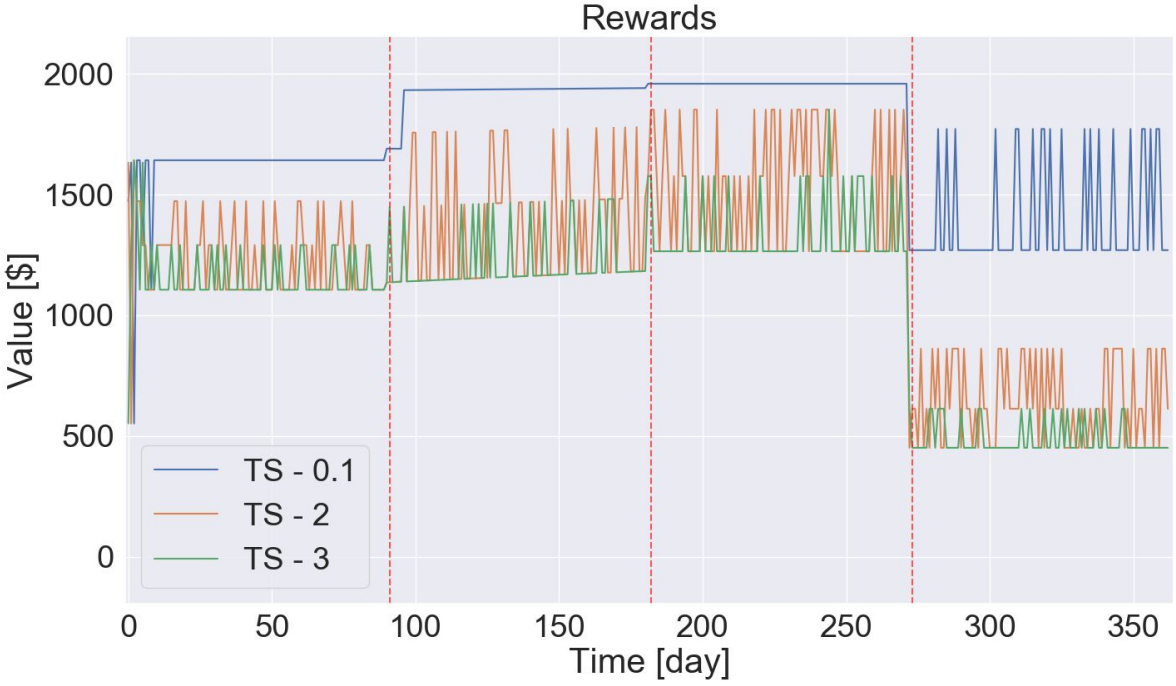
Thompson-Sampling TS

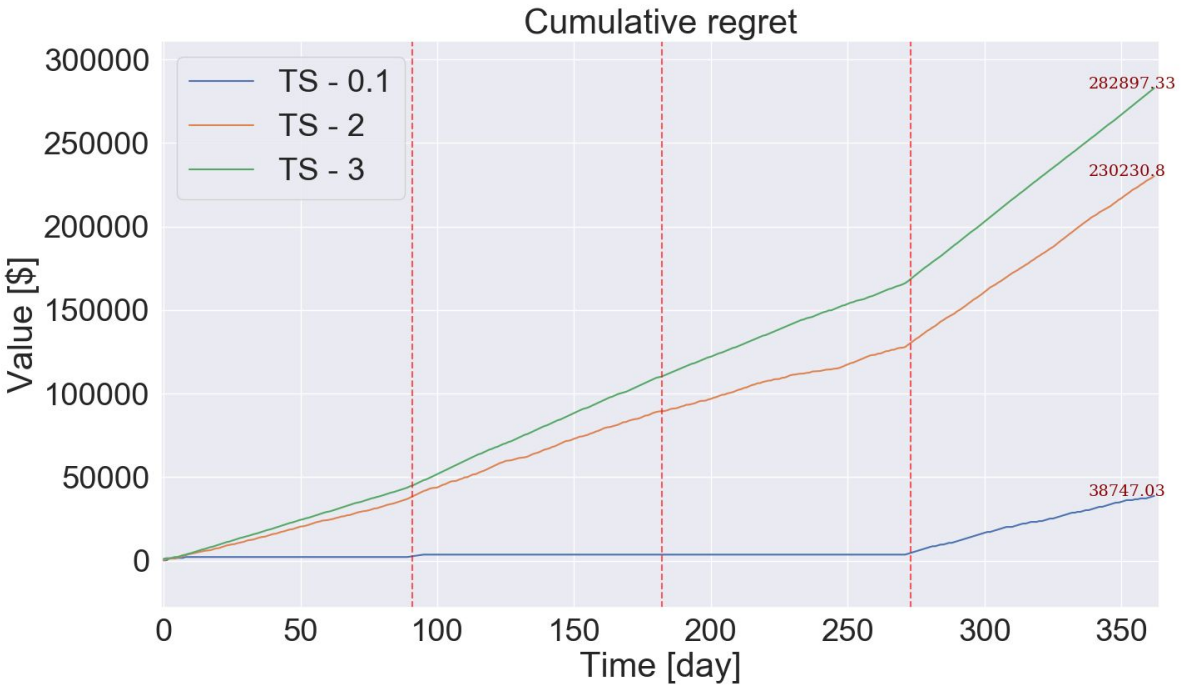
TS is a **MAB algorithm** that estimates the distribution of the unknown variable for each arm.

It maximizes the reward by drawing a sample from each arm and by selecting the arm that provides a higher reward.
Once the arm has been pulled, it updates the selected arm distribution according to its parameters.

It requires an assumption on the distribution of the unknown variable; in our case, we assume a **gaussian distribution** for each arm, so we estimate (μ, σ).


Regret
TS - 0.1
TS - 2
TS - 3
Value [$]
Time [day]
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300
350


Rewards
TS - 0.1
TS - 2
TS - 3
Value [$]
Time [day]
0
500
1000
1500
2000
0
50
100
150
200
250
300
350



Conclusion

TS, T = 365.
The algorithm is able to adapt in the first three seasons, even with high values of noise.
However, the changes relative to the last season are not learned.

Adding the sliding window (SW)

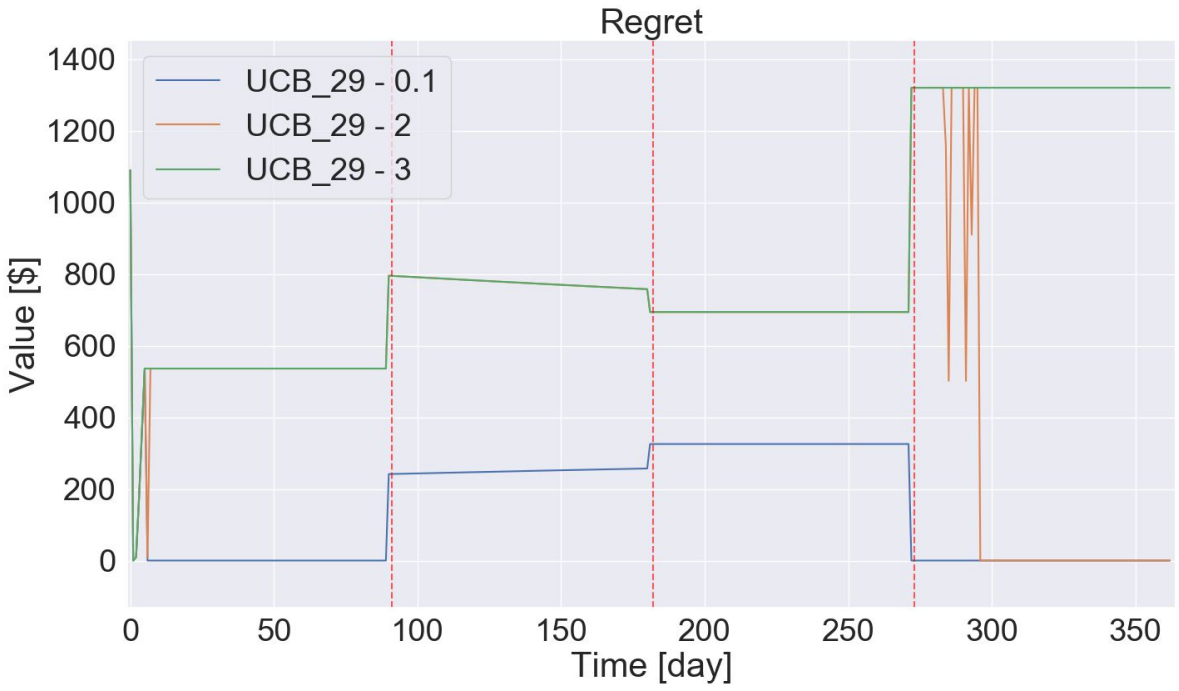
None of the algorithms proposed so far were able to adapt perfectly to abrupt changes.

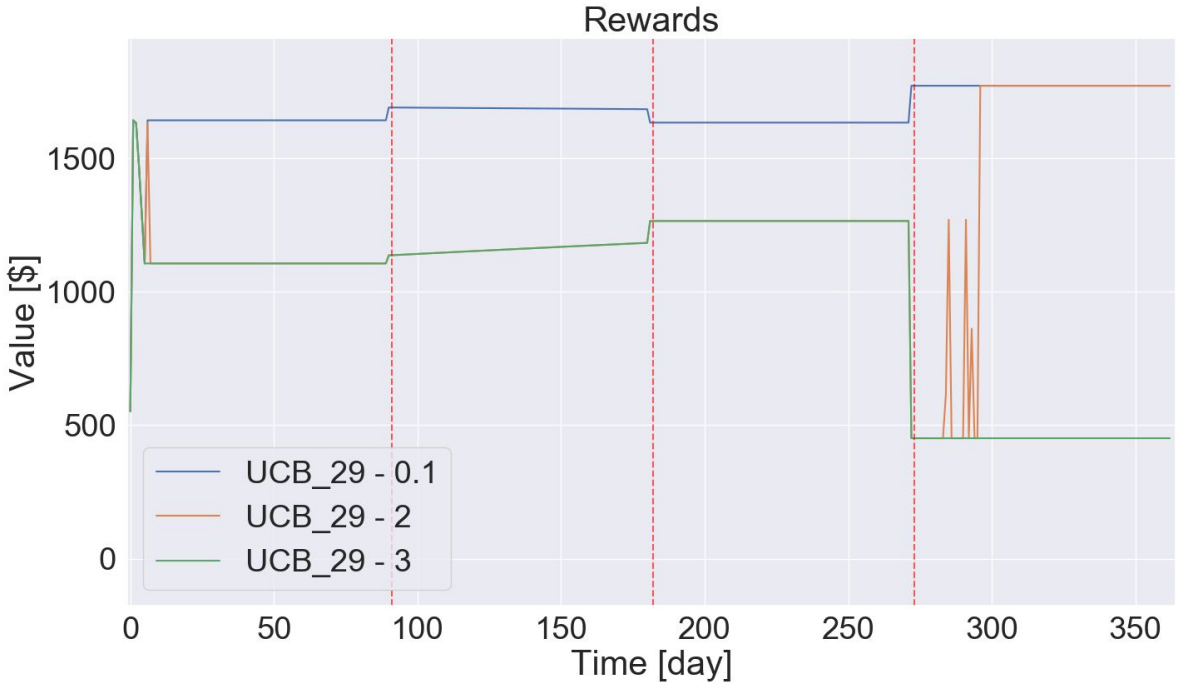
The problem behind both UCB1 and TS is that they both don't expect abrupt changes, so they tend to trust more and more their predictions.
In case of UCB1 this happens by monotonically reducing the bound with the increase of t, while TS with Gaussian arms reduces variance when increasing the number of samples.

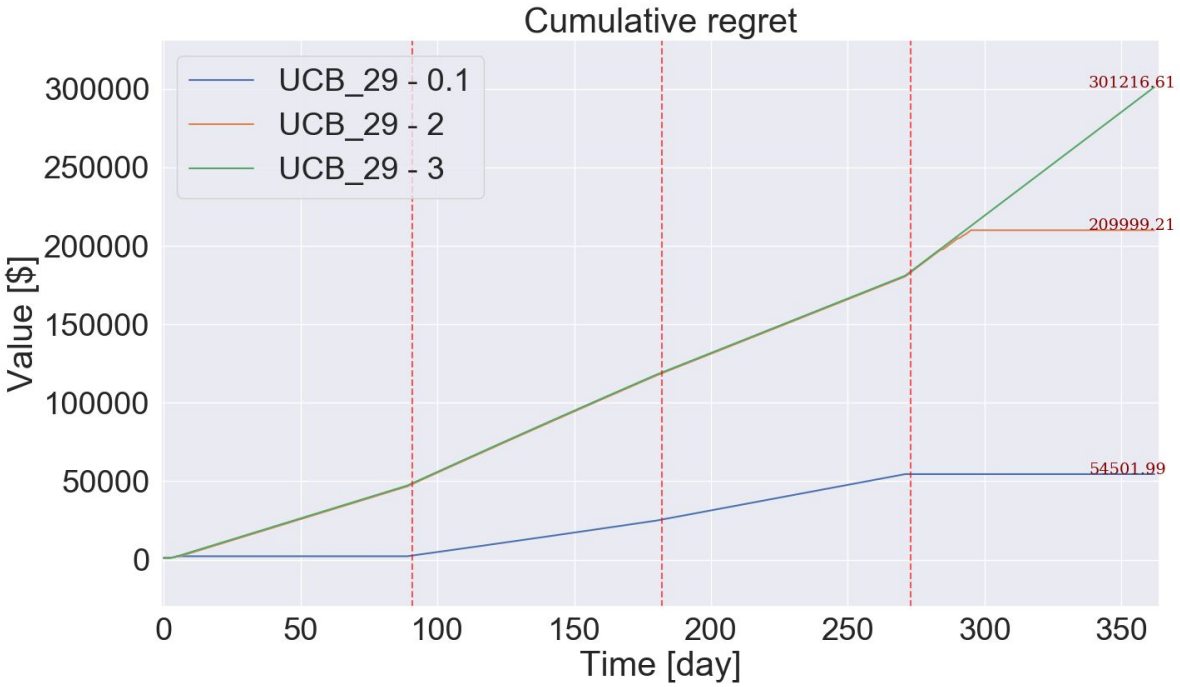
This means that these approaches are not well suited to handle **non stationary environments**.

To avoid this limitation, we adopt a **sliding window approach**, therefore in both algorithms we consider only the most recent samples.
The number of samples considered corresponds to the size of the window $W = \sqrt{T} = 29$.

Sliding window UCB SW-UCB







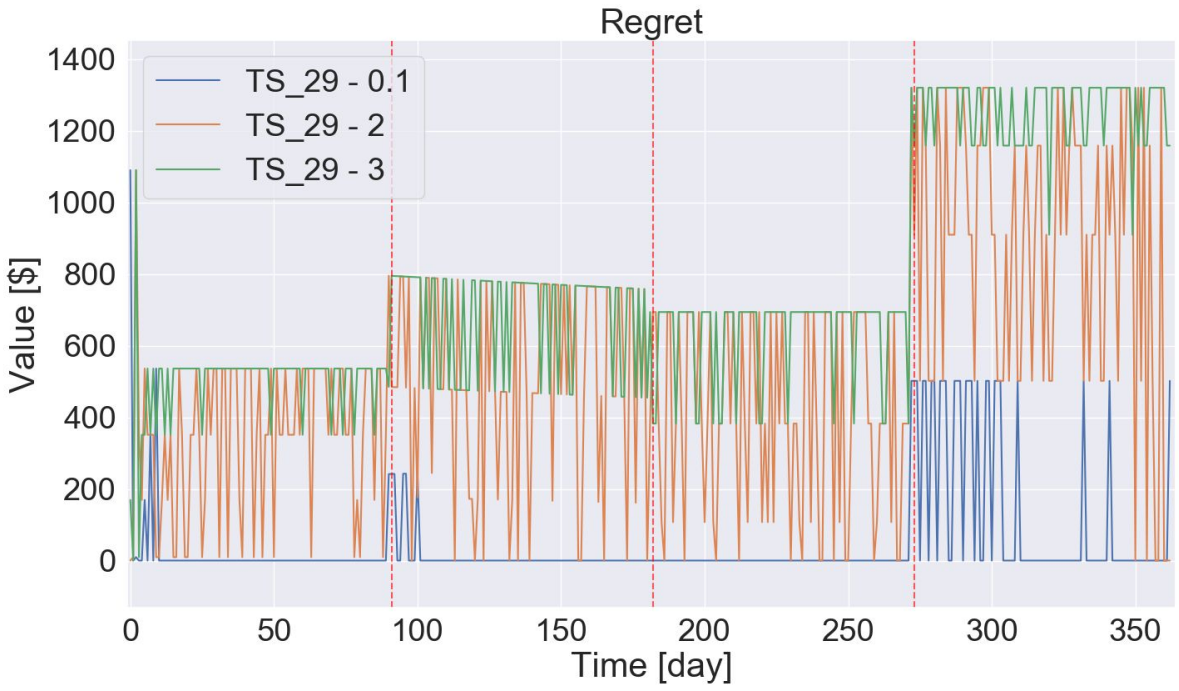
Conclusion

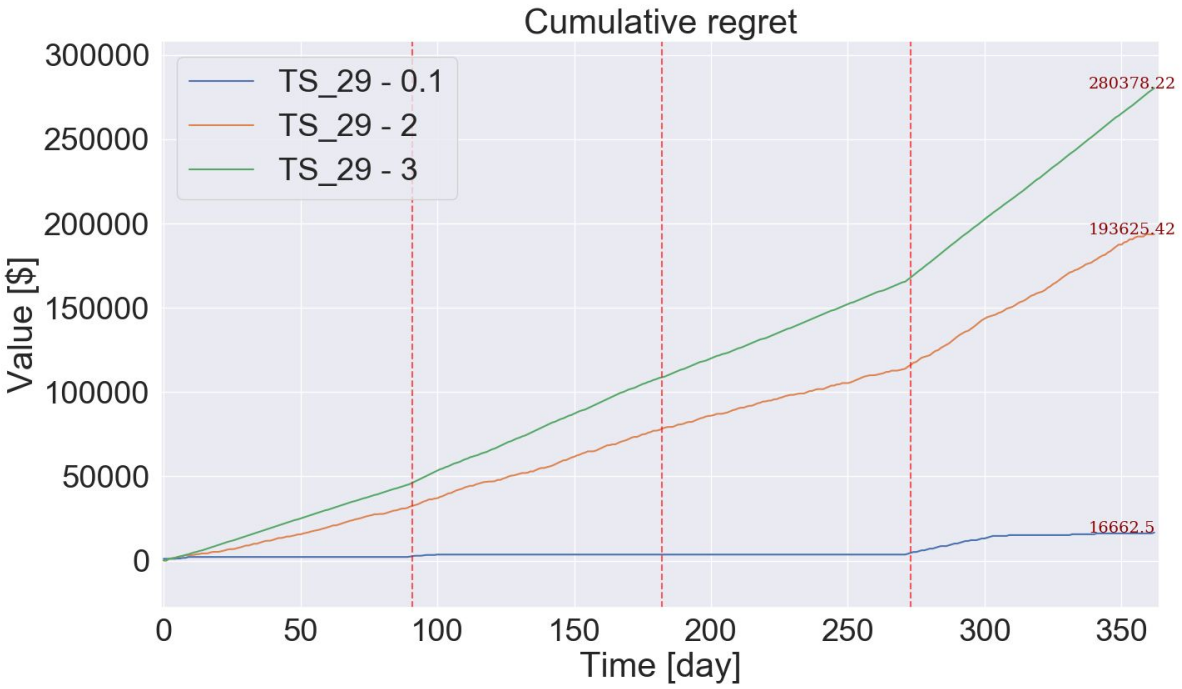
SW-UCB1, T = 365.

The algorithm is able to find the best solution in the first season regardless of the noise but it is not able to adapt to abrupt changes, i.e., **UCB1 continuously takes the best arm for the first season.**

This is due to the fact that the values related to the best arm for season 1 is updated, while the one of season 2 is not, i.e., best arm for season 1 is better in season 2 than best arm for season 2 in season 1.

Sliding window Thompson-Sampling SW-TS





Conclusion

SW-TS, T = 365.

For low noise, **SW-TS is the only one that is able to find the best solution in every season**.

UCB1, TS, SW-UCB1, SW-TS with context generation (question 5)

Context generation

Instead of always considering the aggregate curve (i.e., same demand curve for all the users), we consider the case in which we have different curves depending on the context.

The **context selection** consists of finding the correct way to "group" the users.

Indeed, the demand curve representing users of class 1 and class 2 is different from the one that represents, separately, those of class 1 or class 2.

Three users correspond to a total of five contexts, each of which is the union of subcontexts. Each subcontext is associated with a function.

Here's a summary:

Context	Description
C_0	Aggregate curve, users $\{\{U_1, U_2, U_3\}\}$
C_1	Two subcontexts. $\{\{U_1, U_2\}, \{U_3\}\}$
C_2	Two subcontexts. $\{\{U_1, U_3\}, \{U_2\}\}$
C_3	Two subcontexts. $\{\{U_2, U_3\}, \{U_1\}\}$
C_4	Three subcontexts (disaggregated curves). $\{\{U_1\}, \{U_2\}, \{U_3\}\}$

Algorithm for context generation

Context generation	Context selection

• *Initialize curves for the context*
• *Define* $\tau \equiv$ *period of selection.* $C_{max} = C_0$
• *Initialize a Context Learner for each context*
• *Train for the time horizon* T *on* C_{max}
• *At every period* τ : $C_{max} = contextSelection()$
	• *For each context* C_i
	  ○ *For each subcontext* $S_{i,j}$
	  ■ *Compute* $\bar{\mu}_{i,j}$, $\bar{p}_{i,j}$
	• $C_{max} = argmax_i(\sum_j \bar{\mu}_{i,j} , * \bar{p}_{i,j})$

- A context learner is a set of learners, each of which corresponds to a subcontext
- $\mu^{-}_{i,j}$, $p^{-}_{i,j}$ are lower bounds
    - $\mu^{-}_{i,j} = \mu_{i,j} - t_{N-1,\ 1-\alpha/2} * \frac{s}{\sqrt{N}}$, i.e., Chernoff bound
    - $p^{-}_{i,j} = p_{i,j} - \sqrt{\frac{log(1/\alpha) * 2}{N}}$

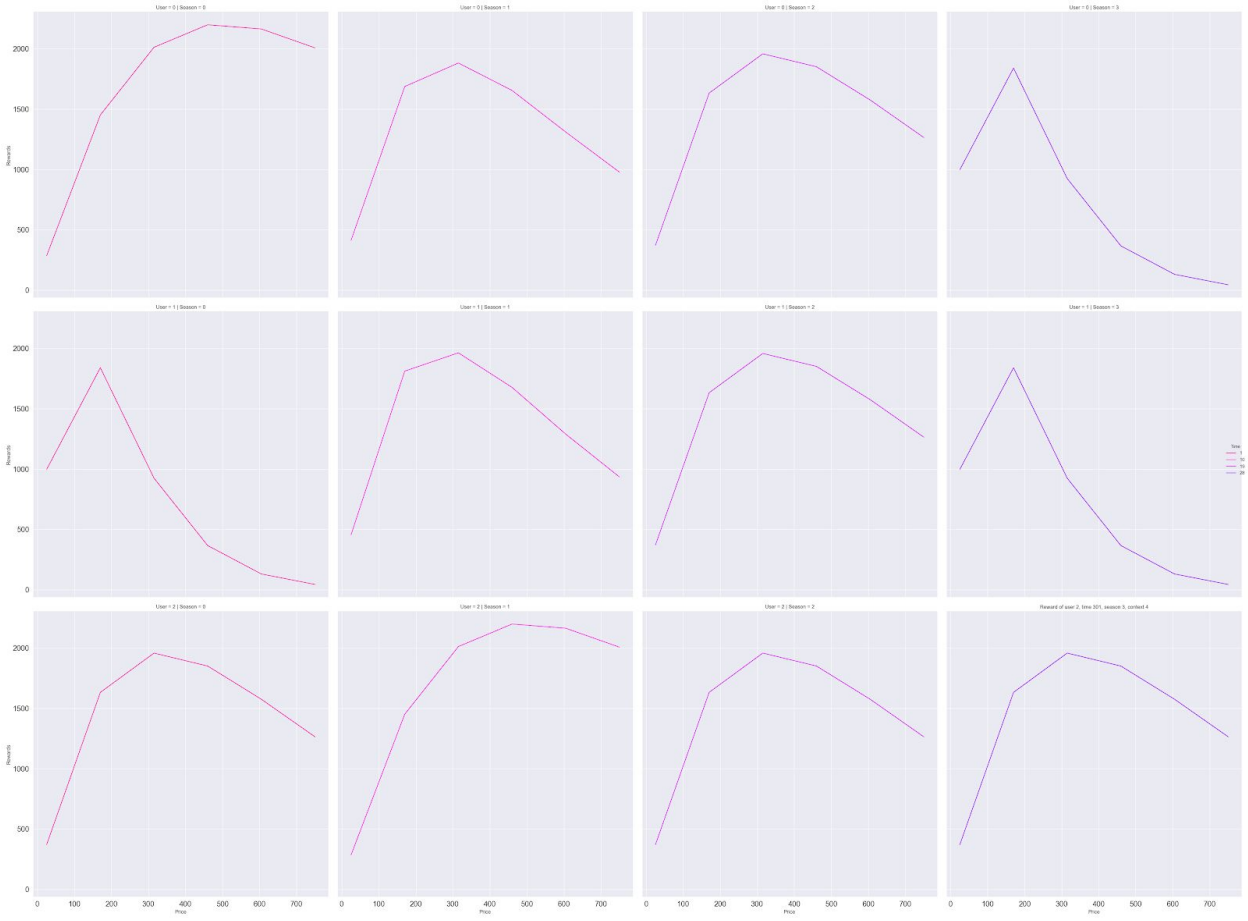
Notes and guesses

Please note that it is not always intuitive to understand which context is the best one to be selected.

This choice depends on the distribution of the users at timestamp t and on the values of the single curves. We can notice, though, that the first and the third seasons are represented by a distribution that varies over time, uniform among the classes (please refer to the previous plot of the distribution).

Together with the distribution, as mentioned before, we need to see the distribution of the disaggregate curves in the different seasons to make a priori consideration on the desired result.

Here we show the disaggregate reward curves (i.e., each value of demand has already been multiplied by the value of the corresponding arm) in the different seasons for each class of user.

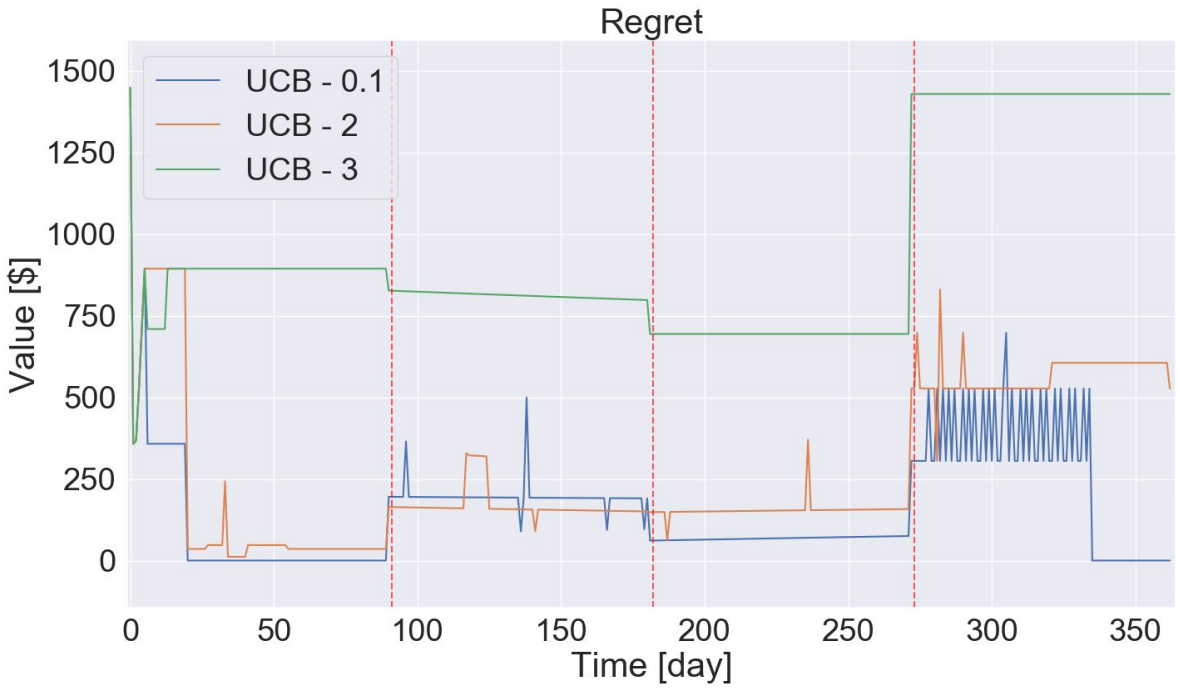


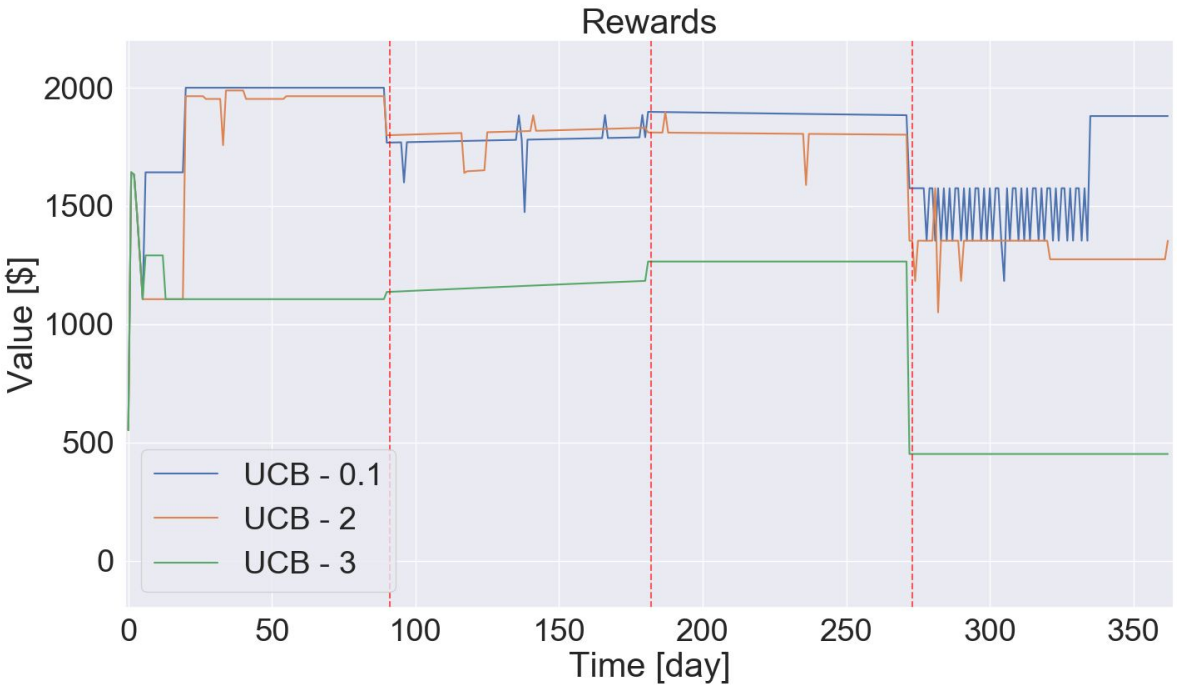
By analyzing these curves, we can see that:

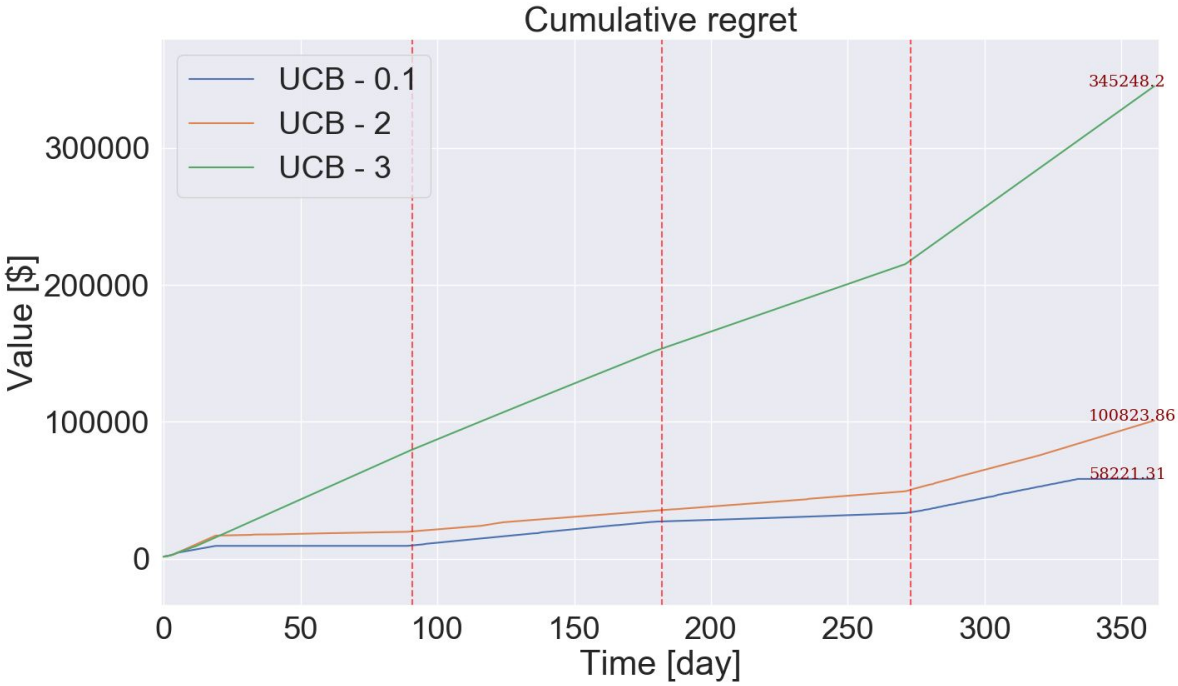
- **First season** is characterized by curves having different best arms.
  This should lead the model to choose the **disaggregated** context
- **Third season** is characterized by curves associated with the same best arm.
  This should lead the model to choose the **aggregated** context.

Rewards and **regrets** are computed considering the rewards of the best context, therefore we expect to converge to the optimal solution (if that happens) slower than before.

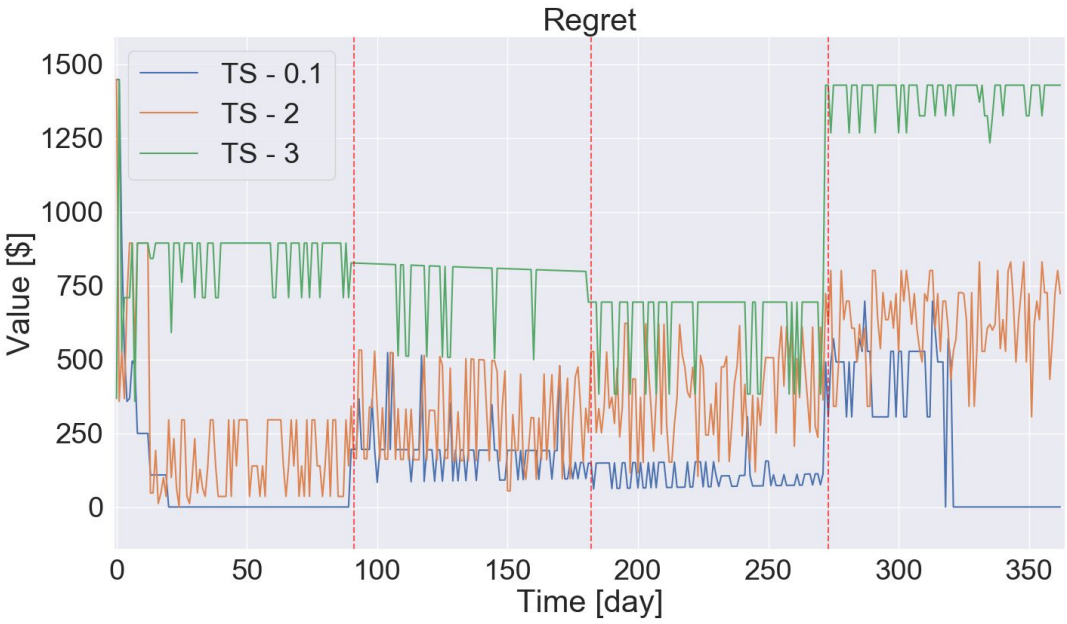
UCB


Regret
UCB - 0.1
UCB - 2
UCB - 3
Value [$]
1500
1250
1000
750
500
250
0
0
50
100
150
200
250
300
350
Time [day]

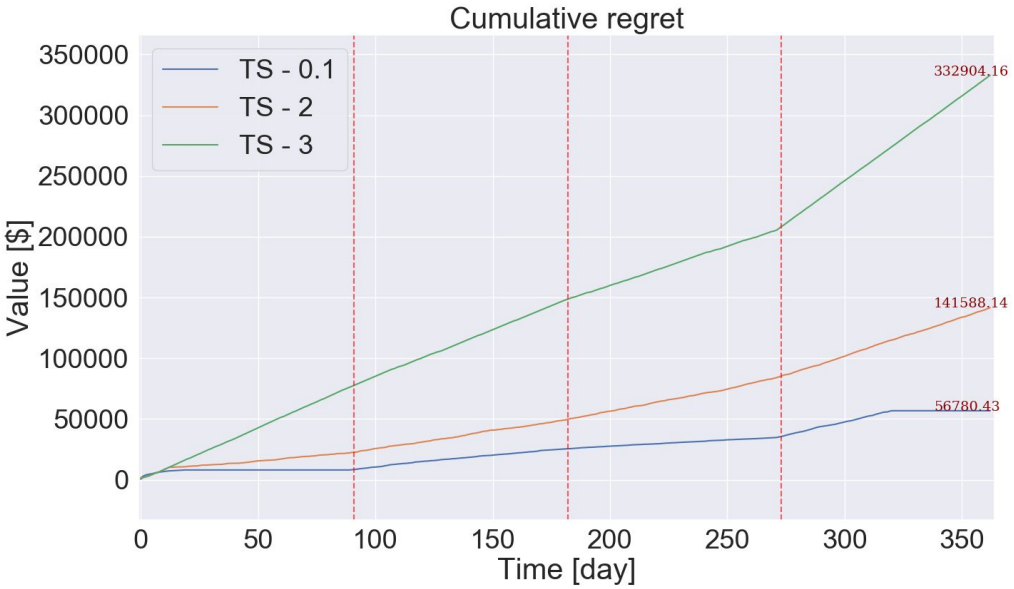

Rewards
2000
1500
1000
500
0
Value [$]
UCB - 0.1
UCB - 2
UCB - 3
0
50
100
150
200
250
300
350
Time [day]


Cumulative regret
UCB - 0.1
UCB - 2
UCB - 3
345248.2
100823.86
58221.31
300000
200000
100000
0
Value [$]
0
50
100
150
200
250
300
350
Time [day]

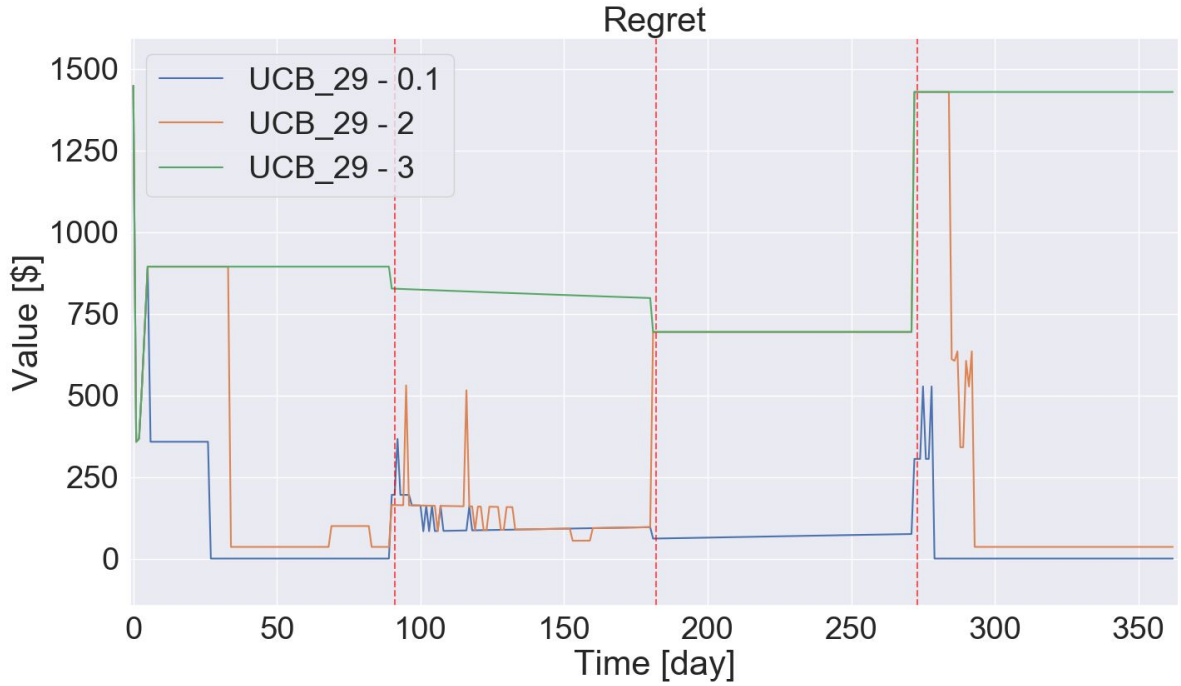
TS

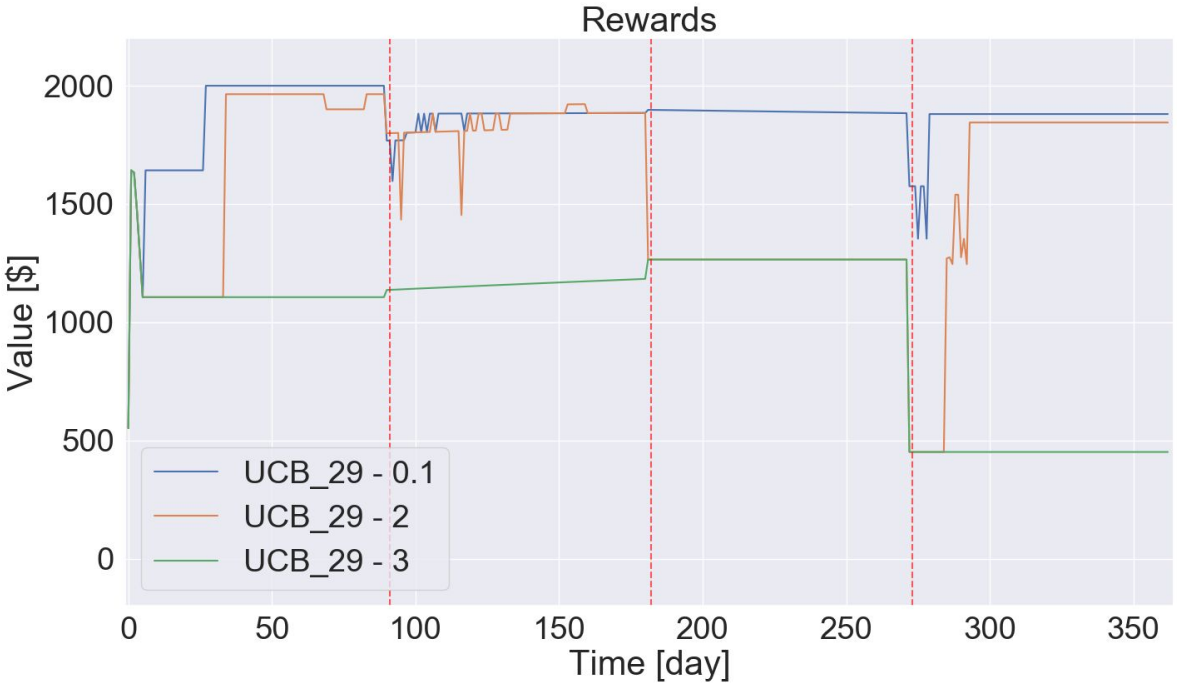

Regret
TS - 0.1
TS - 2
TS - 3
Value [$]
Time [day]
0
250
500
750
1000
1250
1500
0
50
100
150
200
250
300
350

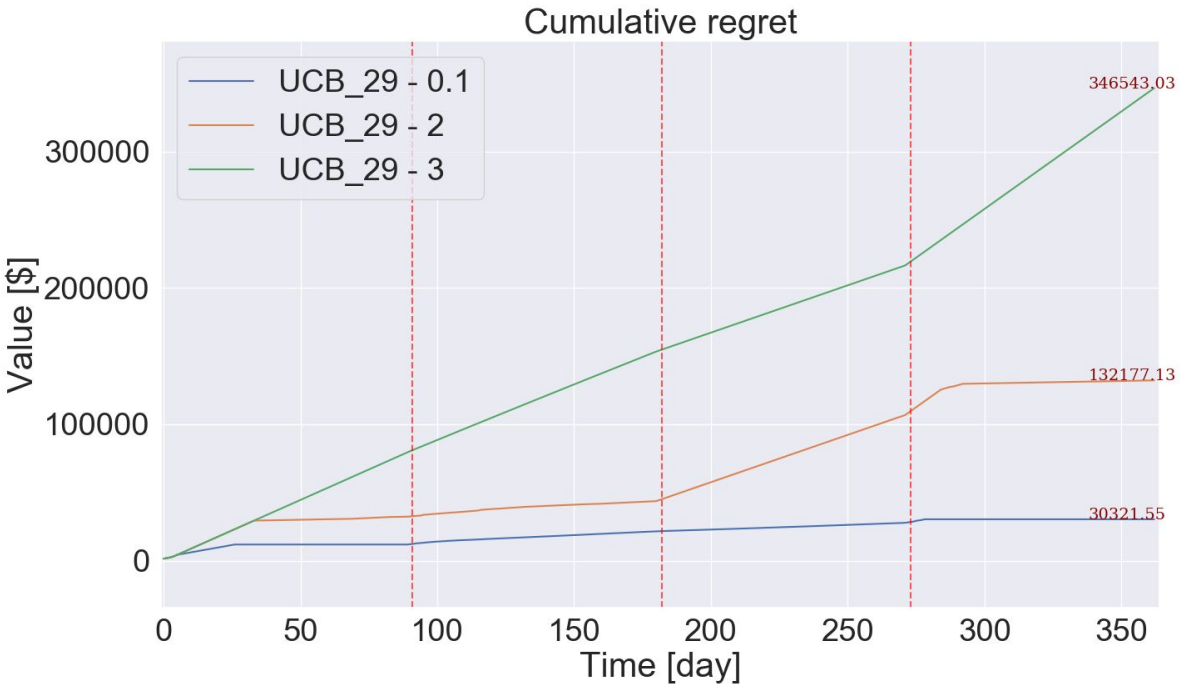

Rewards
TS - 0.1
TS - 2
TS - 3
Value [$]
Time [day]
0
500
1000
1500
2000
0
50
100
150
200
250
300
350


Cumulative regret
TS - 0.1
TS - 2
TS - 3
332904.16
141588.14
56780.43
Value [$]
Time [day]
0
50000
100000
150000
200000
250000
300000
350000
0
50
100
150
200
250
300
350

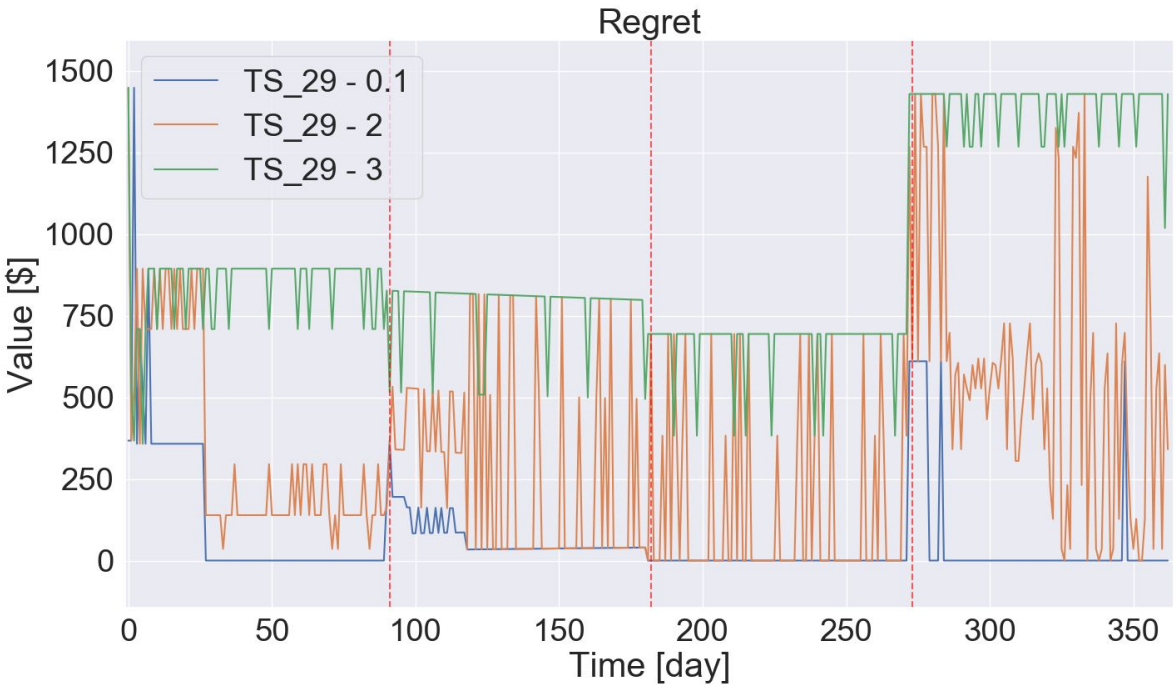
SW-UCB

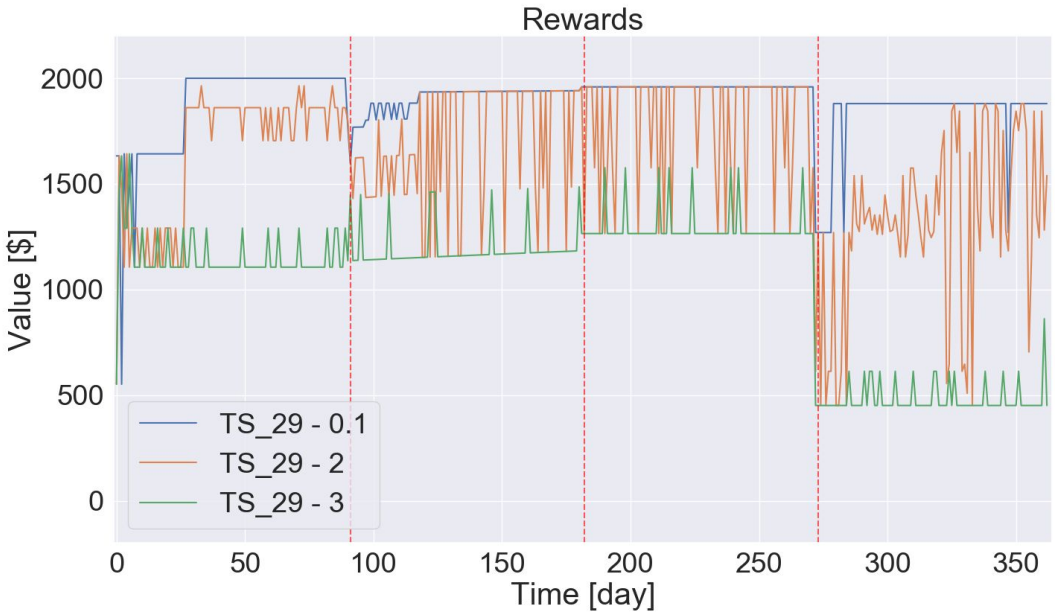

Regret
UCB_29 - 0.1
UCB_29 - 2
UCB_29 - 3
Value [$]
Time [day]
0
250
500
750
1000
1250
1500
0
50
100
150
200
250
300
350

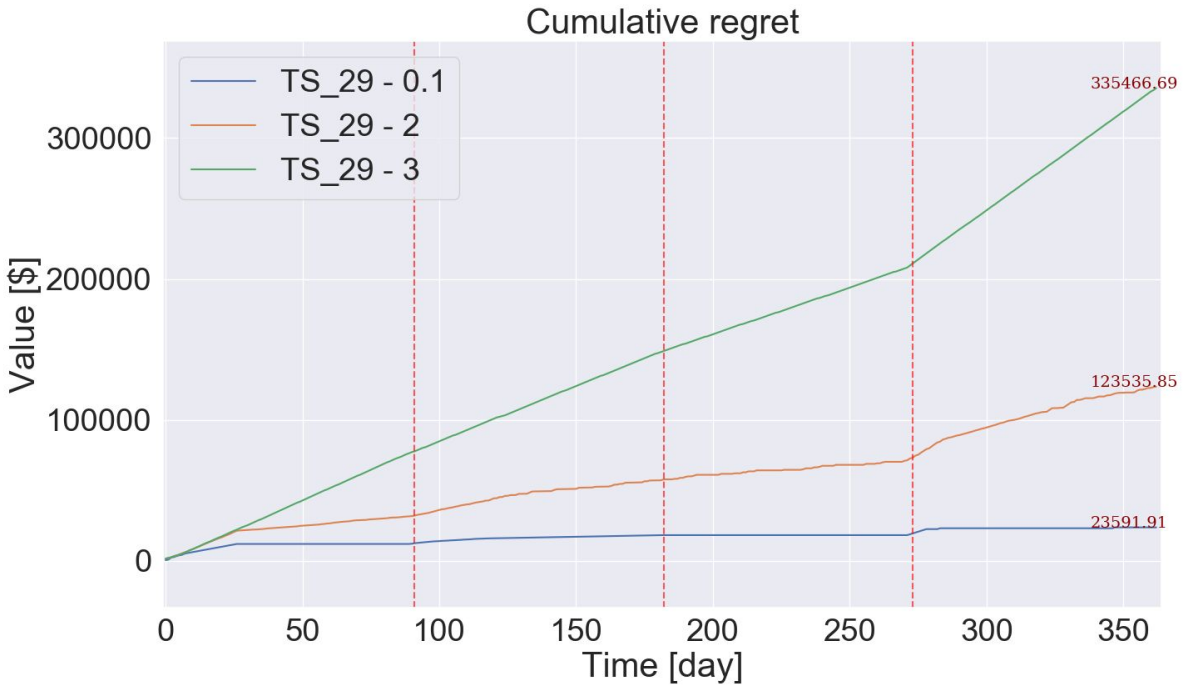

Rewards
2000
1500
1000
500
0
Value [$]
UCB_29 - 0.1
UCB_29 - 2
UCB_29 - 3
0
50
100
150
200
250
300
350
Time [day]


Cumulative regret
UCB_29 - 0.1
UCB_29 - 2
UCB_29 - 3
346543.03
132177.13
30321.55
300000
200000
100000
0
Value [$]
0
50
100
150
200
250
300
350
Time [day]

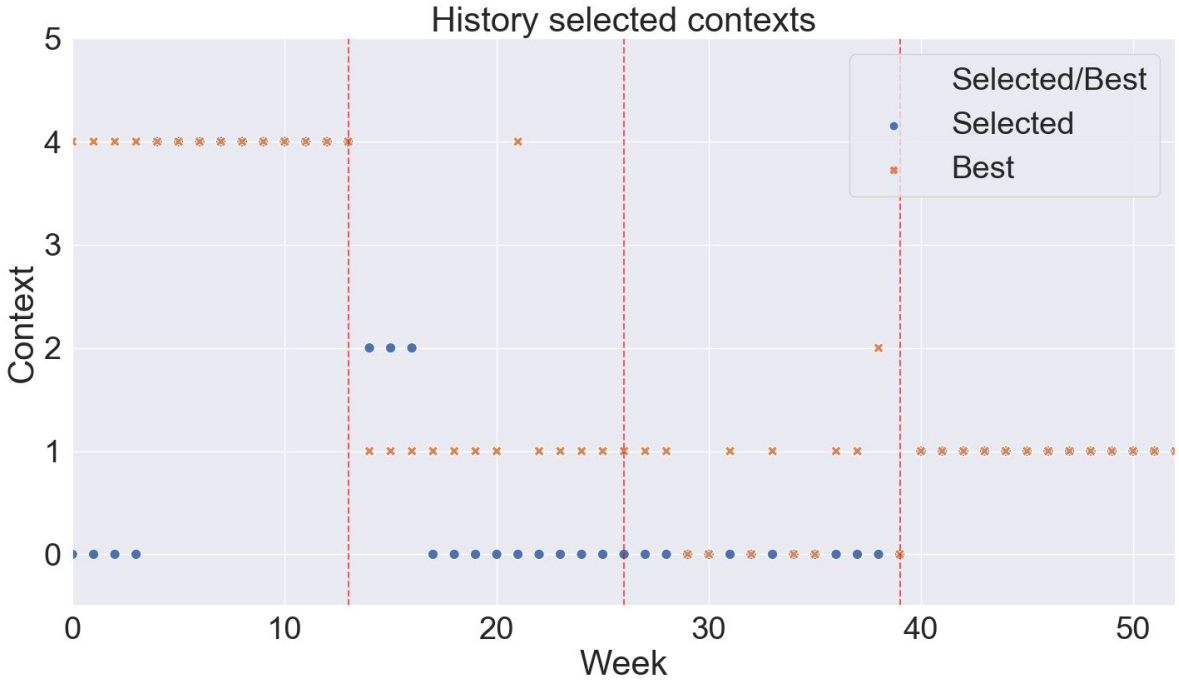
SW-TS


Regret
TS_29 - 0.1
TS_29 - 2
TS_29 - 3
Value [$]
Time [day]
0
250
500
750
1000
1250
1500
0
50
100
150
200
250
300
350


Rewards
TS_29 - 0.1
TS_29 - 2
TS_29 - 3
Value [$]
Time [day]
0
500
1000
1500
2000
0
50
100
150
200
250
300
350



Example of selected context vs best context in SW-TS for sigma = 0.1



Conclusion

In the last plot we observe the capability of the SW-TS algorithm to select the best context in every phase.

The first and last seasons are almost perfect, while those in the middle are a bit off.
Anyway, the model is capable of producing low regret, as seen in the cumulative regret plot.

SW-TS outperforms every other algorithm as expected.

Increasing the noise with sigma = 3 is problematic to each algorithm.

This issue could be solved by associating a regressor (e.g., a Gaussian process), even though this problem may be created by the high value of sigma that causes a bias too high for the estimates.