Politecnico di Milano
A.Y. 2019-2020

POLITECNICO
MILANO 1863

Project 2: Social Influence and Pricing

Data Intelligence Applications

Diego Piccinotti, Federica Ilaria Cesti, Lorenzo Casalini,
Matteo Carretta, Umberto Pietroni

TABLE OF CONTENTS

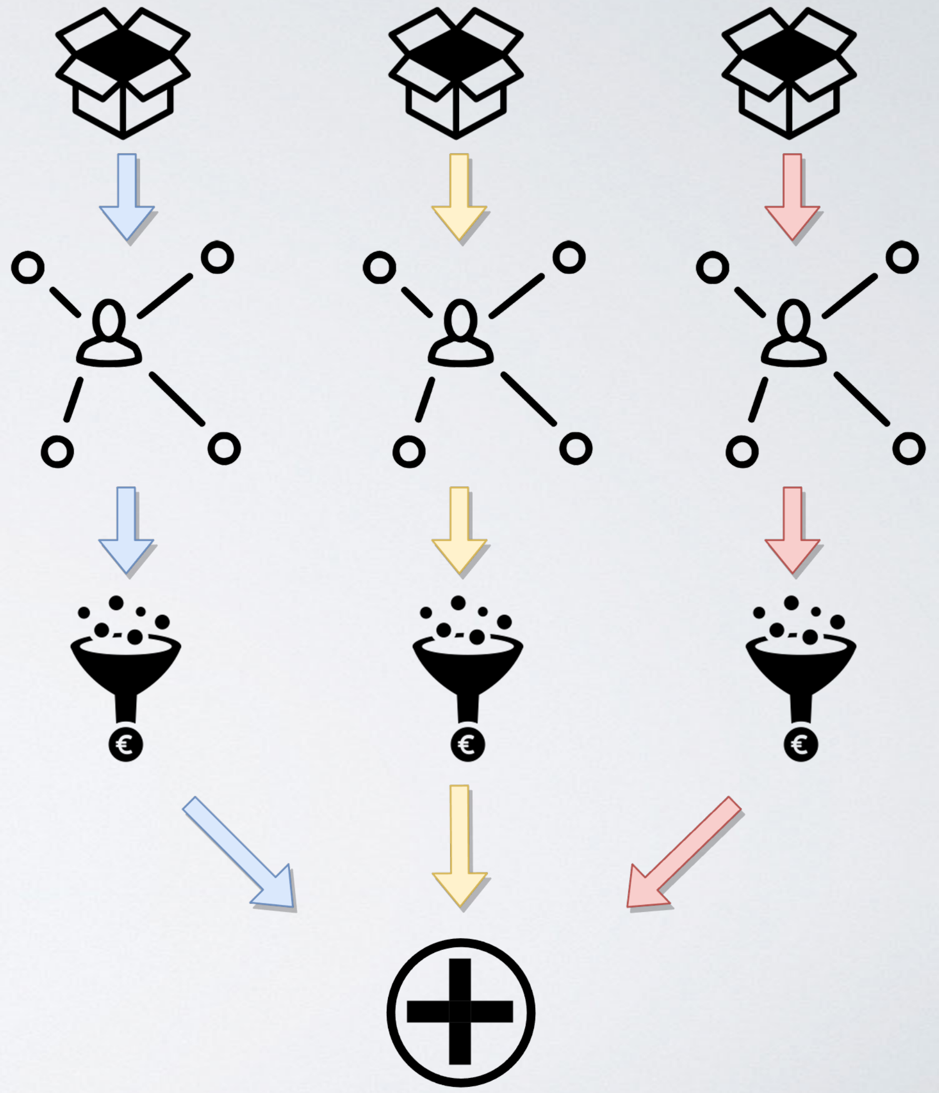
INTRODUCTION

The goal is to model a scenario in which a seller is pricing some products and spends a given budget on **social networks** to persuade more and more users (nodes) to buy the products, thus artificially increasing the demand.

The seller needs to learn both some information on social networks and conversion rate curves. In particular, the seller has to price **three different products**, each with an infinite number of units, in a given time horizon T, on three social networks composed of thousands of nodes.

Each product is associated to a social network. The activation probabilities of the edges of the social networks are linear functions in the values of a least three features, potentially different for each social network.

There are three seasonal phases, such that the transition from a phase to another is abrupt. For each social network, and for each phase, there is a **conversion rate curve**, for a total of nine curves, which return the probability that a generic node of the social network buys the product it is associated with.



SOCIAL INFLUENCE

Social Influence Environment

- Three datasets with thousands of nodes taken from SNAP datasets collection:
    - **Google Plus:** this dataset consists of 'circles' from Google+
    - **Wikipedia:** vote history data where the community decides who to promote to administrator
    - **email-Eu-core:** generated using email data from a large European research institution
- We assumed that each edge connecting two nodes is characterised by 5 different **features**. Our assumption is that each feature corresponds to a type of interaction between users. In a social network these could corresponds to number of likes, post shares, comments, tags and private messages. The features were assigned randomly to each edge in the dataset with values going from 0 to 100.
- Each social network has 5 different feature **weights** (that sum up to 1) which are used to compute the edge activation probabilities:

$$p^e = \frac{\sum_i^F w_i \ * \ x_i^e}{max_f}$$

$where$ w are the weights, x is the feature vector for edge e and max_f is the maximum value of a feature, used to normalize the probability

SOCIAL INFLUENCE

Influence Computation
Input = (**seeds**, edge activation probabilities matrix, # mc simulation, # max steps)
Output = (influence reward)

- We adopted the **Independent Cascade** model to simulate the influence spreading on the social graph:
  - Seed nodes are initial k active nodes
  - If node a becomes active, it can activate a neighbour b with probability $p_{a,b}$
  - If a node activates at time t, then at time t+1 it becomes inactive
  - A single episode lasts until the maximum number of steps is reached or there is no new active node

- **Monte Carlo Sampling**
  - Simulates n episodes where n is the number of Monte Carlo simulations z_i is the number of episodes in which node i activates
  - It returns the activation probability of each node: $p_i = \frac{z_i}{n}$
- **Influence reward**
  - Expected number of activated nodes is computed as the sum of the activation probabilities of each node.

$$influence = \sum_{i}^{N} p_i$$

SINGLE INFLUENCE MAXIMISATION

Influence Maximisation: maximize the social influence in a single social network given a certain budget.

Input = (**budget**, edge activation probabilities matrix, # mc simulation, # max steps)
Output = (**best seeds**, max influence)

- First we designed the **Exact Solution algorithm** which enumerates all possible combinations of seeds on a given budget and computes the influence of each of them. The advantage of this approach is that it returns the best solution since it explores all possible combinations of seeds. However, the computation complexity of this algorithm is unfeasible as the number of possible combinations gets extremely high with just hundreds of nodes.

- To overcome the computation unfeasibility, we designed a **Greedy algorithm** that selects incrementally the seeds by choosing, for every node that is not a seed, the one for which the marginal increase if maximised, until the number of seeds reaches the budget.

- For a faster execution we designed a **parallel** version of the Greedy algorithm where we create a separate thread each time we compute the marginal increase of a node.

SINGLE INFLUENCE MAXIMISATION

Experimental Results

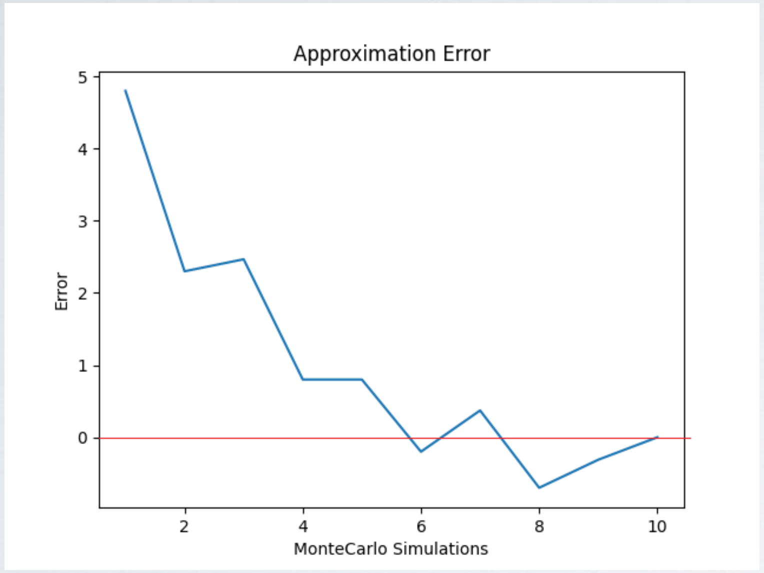
Due to the high computational cost required by the Influence Maximisation algorithms, we resized each dataset to contain at most 1000 nodes in our experiments.

Given the relatively low number of nodes for social network, we performed our experiments with 5 budget nodes and 2 max steps per simulation after observing that the algorithm reached a high value of expected influence.

For each social network, we performed an experiment where we increased the number of Monte Carlo simulations in order to observe the behaviour of the Greedy Influence Maximisation algorithm. In particular we analysed the approximation error, which is the difference of the expected influence obtained with k simulations with respect to the one with the highest number of Monte Carlo simulation.

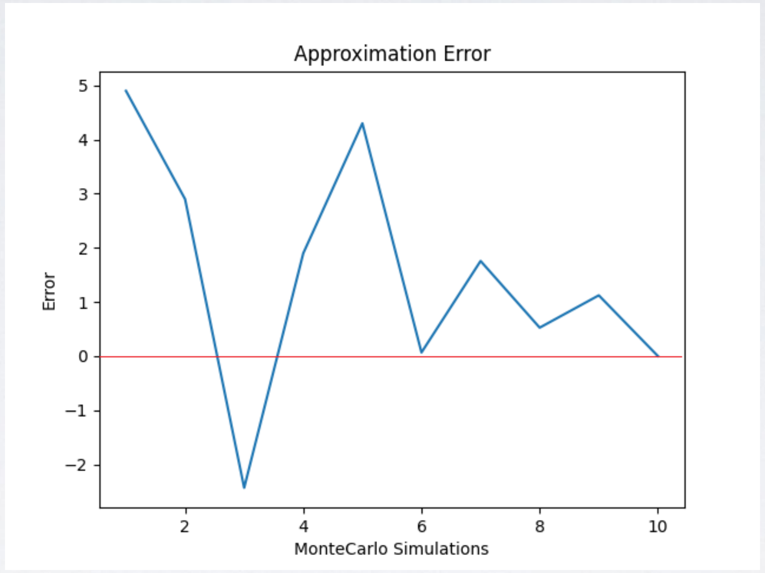
SINGLE INFLUENCE MAXIMISATION

Gplus apx. error



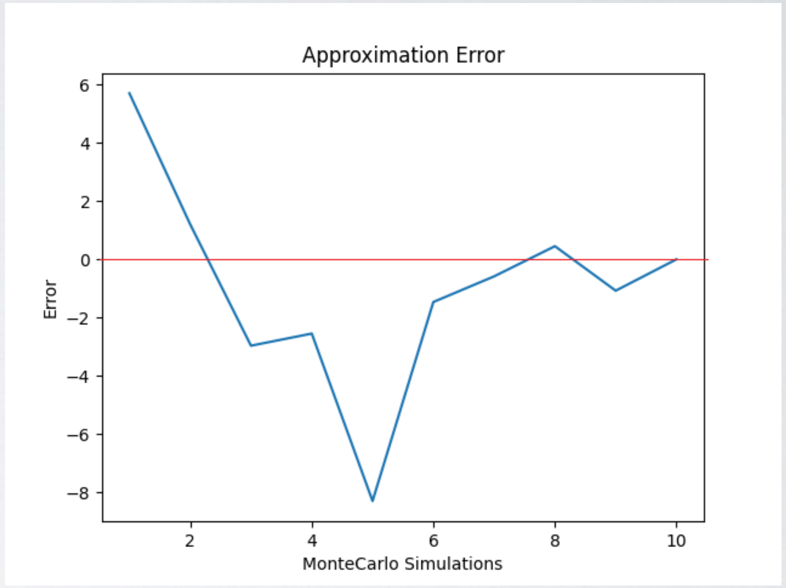
Influence with mc 10: 928,20

Wikipedia aprx. error



Influence with mc 10: 381,10

Email aprx. error



Influence with mc 10: 848,30

For the same computational problems, we couldn't experiment with a very high number of monte carlo simulations. However, even if the number of Monte Carlo simulation is low, it is possible to observe the desired perfomances where the influence varies a lot with less simulations but it tends to converge when the number of simulations increases. This means that the value is converging to the optimal solution returned by the algorithm when the number of simulation is very high

JOINT INFLUENCE MAXIMISATION

Joint Influence Maximisation: maximize the total social influence in the three social networks given a shared and unique budget.
Input = (**total budget**, edge activation probabilities matrix, # mc simulation, # max steps)
Output = (**distributed budget**, seeds, influence)

- Distributing the budget between the three social networks is a constrained integer optimization problem. The function we have to maximize is the sum of the three social influences, and the constraint is that the sum of the respective budget for each social network has to be equal to the total budget given as input. The objective function we want to maximize is the sum between the three single social network influences each multiplied by its price weight:

$$Joint\ influence = \sum_{i=1}^{3} social\_influence[i] * price\_weight[i]$$

- Price weight is the price at which the corresponding product is sold on the given social network.
- We have two constraints on the three budget of the Social Networks: the sum of the three elements of the **distributed budget** array at the end of the algorithm must be equal to the **total budget**, the input of our algorithm, and each Social must have a distributed budget:

$$\sum_{i=1}^{3} distributed\_budget[i] = total\ budget$$

$$distributed\_budget[i] \geq 1, 1 \leq i \leq 3$$

- We tried different approaches to solve this problem. We looked and tried to implement a solution using linear optimization libraries like SciPy or Pulp, but there are no libraries to do integer constrained optimization that accept as objective function a custom function like the Joint Influence we declared formerly.

JOINT INFLUENCE MAXIMISATION
Algorithm intuition

- We designed then a **greedy algorithm** to perform this constrained maximization. Every Social Network starts with the same budget, we increment the budget of the one who's going to get the biggest increase in terms of Social Influence at each time, until the sum of every single budget is equal to the total budget.

Key elements

- **Initialization:** each Social Network starts with budget 1. We calculate each SN influence with budget 1 and budget 2 since at the first step we need to compute the maximum increase that each SN has with budget = 2 with respect to budget = 1.
- **Repeat** (until sum of budget = total budget):

  1) **Choose what SN should have a budget increase:**

     for each SN, we compute the difference between $social\_influence[budget+1] - social\_influence[budget]$ to choose which one gets the biggest increase in influence when its budget is increased.

  2) **Increment that budget**
  3) **If sum of budget $\neq$ total budget:**

     for the SN whose budget has been increased at point 2) we need to compute $social\_influence[budget+1]$, that we're going to use again in point 1) at the following iteration of the algorithm

     Else:

     the total budget as been reached, the optimal allocation of the total budget is returned

- ***When computing the incremental social influence of a SN we can resume the algorithm from the budget it stopped at by just providing the list of best seeds it has computed so far.***
- *Hence, the **maximum number of iterations** is equal to **total budget***

JOINT INFLUENCE MAXIMISATION
Example

- We present a toy example with total budget = 6
- Each Social Influence result has this format:

{budget: [seed, cumulated influence]}

With budget going from 1 to the optimal budget + 1 of each SN

- First, we compute the Social Influence of each Social Network for budget = 1 and budget = 2. As we said before, each Social Network has a minimum budget of 1.

- E-mail: {1: [0, 281.0], 2: [348, 576.0]}

- GPlus: {1: [653, 890.5], 2: [275, 916.0]}

- Wikipedia: {1: [8, 345.0], 2: [3, 371.0]}

- 576 - 281 = 295, 916 – 890,5 = 25,5, 371 – 345 = 26 → we increment the budget of E-mail from 1 to 2, and we compute the Social Influence of email for budget 3 resuming the step influence calculation at budget 2 with seeds [0, 348]
- E-mail: {1: [0, 281.0], 2: [348, 576.0], 3: [686, 700.0]}

JOINT INFLUENCE MAXIMISATION
Example

- We present a toy example with total budget = 6
- Each Social Influence result has this format:

  {budget: [seed, cumulated influence]}

  With budget going from 1 to the optimal budget + 1 of each SN

- First, we compute the Social Influence of each Social Network for budget = 1 and budget = 2. As we said before, each Social Network has a minimum budget of 1.

- E-mail: {1: [0, 281.0], 2: [348, 576.0], 3: [686, 700.0]}

- GPlus: {1: [653, 890.5], 2: [275, 916.0]}

- Wikipedia: {1: [8, 345.0], 2: [3, 371.0]}

- 700 - 576 = 223, 916 – 890,5 = 25,5, 371 – 345 = 26 → we increment the budget of E-mail again from 2 to 3, and we compute the Social Influence of email for budget 4 resuming the step influence calculation at budget 3 with seeds [0, 348, 686]
- E-mail: {1: [0, 281.0], 2: [348, 576.0], 3: [686, 700.0], 4: [107, 712.5]}

JOINT INFLUENCE MAXIMISATION
Example

- **We present a toy example with total budget = 6**
- Each Social Influence result has this format:

  {budget: [seed, cumulated influence]}

  With budget going from 1 to the optimal budget + 1 of each SN

- First, we compute the Social Influence of each Social Network for budget = 1 and budget = 2. As we said before, each Social Network has a minimum budget of 1.

- **E-mail:** {1: [0, 281.0], 2: [348, 576.0], 3: [686, 700.0], 4: [107, 712.5]}

- **GPlus:** {1: [653, 890.5], 2: [275, 916.0]}

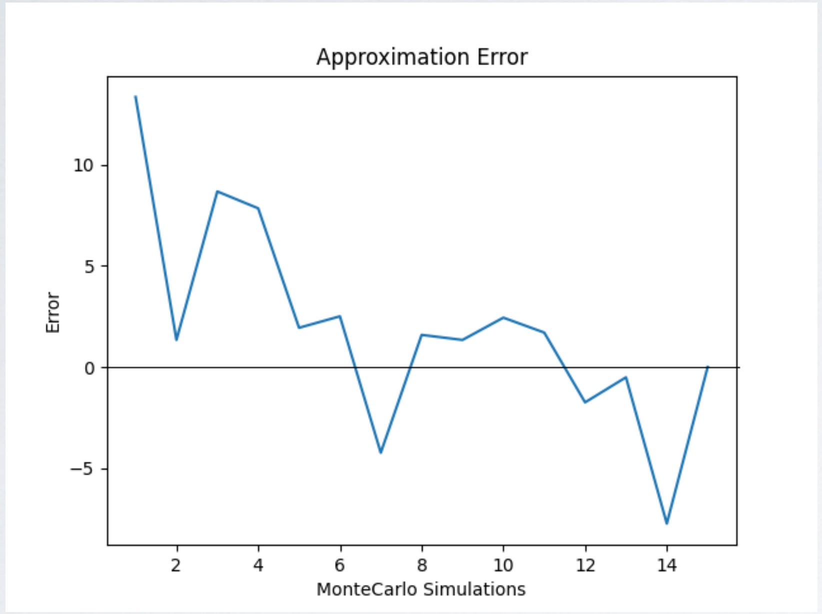
- **Wikipedia:** {1: [8, 345.0], 2: [3, 371.0]}

- 712,5 - 700 = 12,5, 916 – 890,5 = 25,5, 371 – 345 = 26 → we increment the budget of Wikipedia this time from 1 to 2, but since the sum of the single budgets is equal to **total budget**, the termination condition is satisfied and the algorithm terminates returning the optimal solution.

- The **optimal distributed budget** vector is [3, 1, 2] and the total cumulated influence is 700 + 890,5 + 371 = 1961,5

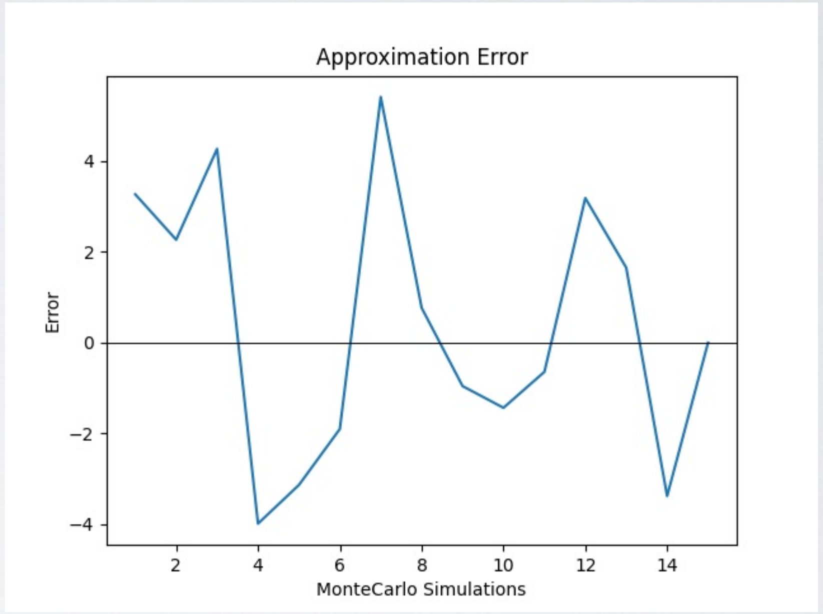
JOINT INFLUENCE MAXIMISATION
Experimental results

Budget 10, MC 15



Optimal budget: [5, 2, 3] Optimal joint influence: 2148.0

Budget 20, MC 15



Optimal budget: [9, 7, 4] Optimal joint influence: 2183.0

We can observe that if we double the budget, the variance of the approximation error increases significantly, thus converging gradually and limiting the fluctuations around the mean value by raising the number of MC simulation, even if it was not computationally feasible to test it with our means.

UNKNOWN ACTIVATION PROBABILITIES

We implemented the LinUCB bandit algorithm to estimate the edge activation probability matrices, considering as reward a sample from a Bernoulli distribution with probability equal to the edge activation probability.

LinUCB pulls at each round the arm with highest upper bounds, checks the activation of the arm, and then updates the estimated distributions.

COMBINATORIAL INFLUENCE MAXIMISATION

After tuning the c parameter for each social network, we implemented an algorithm that iteratively learns their probability matrices and performs influence maximization at the same time.

1. For each social we make a round of LinUCB, pulling an arm and updating the estimated probability matrix.

2. For every round of the bandit algorithm we apply the combinatorial optimization by running the joint influence maximization, to find the best budget allocation over the three social networks, using the matrices estimated with LinUCB.

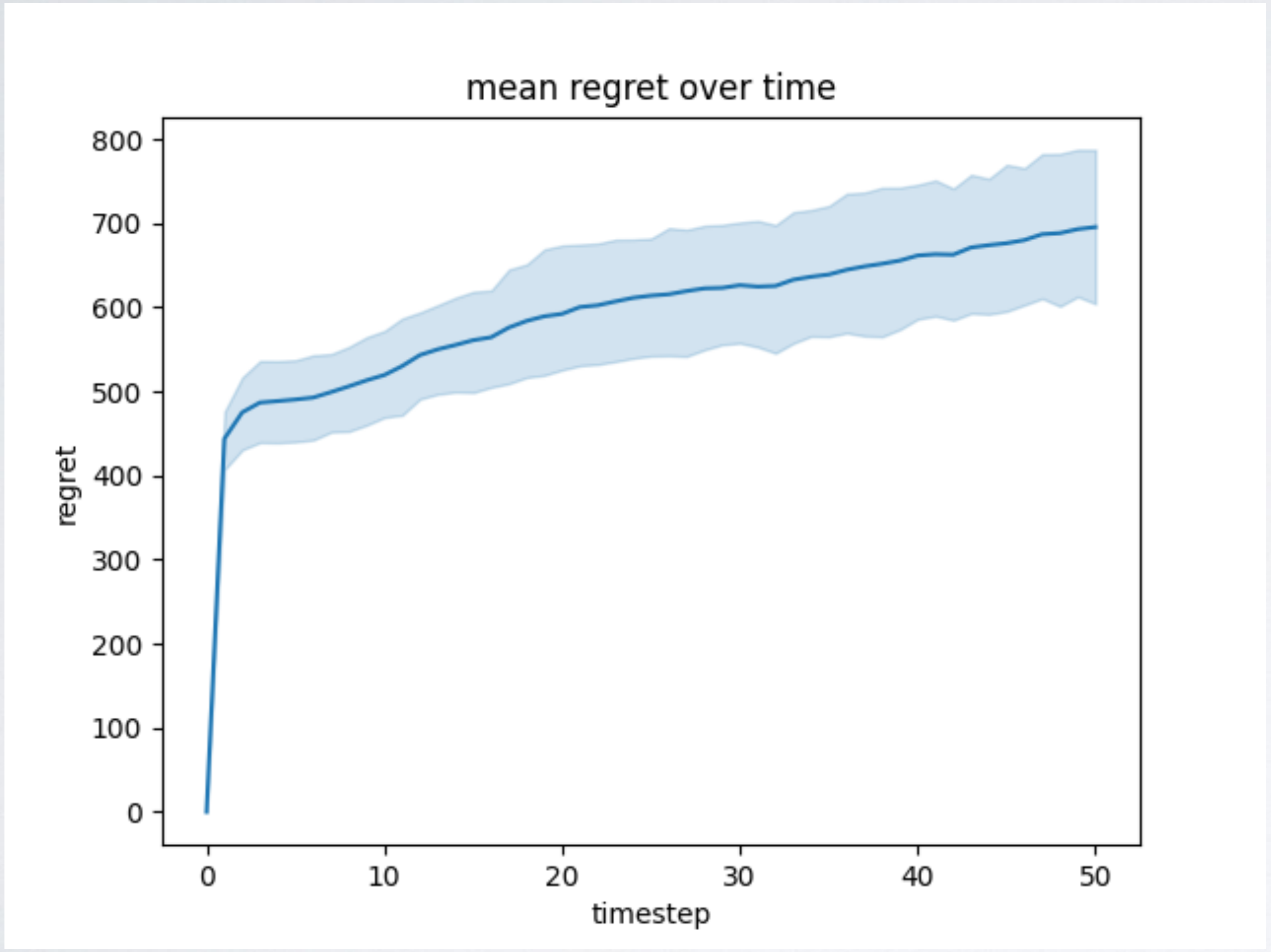
3. We compute the regret [see next slide].

LINUCB REGRET

1. For each social we run a sampling of the edge activations using the seeds provided by the influence maximization step, then we sum the number of activated nodes in each social.

2. We consider as regret the difference between the average number of activated nodes in the three socials when the activation probabilities are known and the value obtained at step 1.

RESULTS

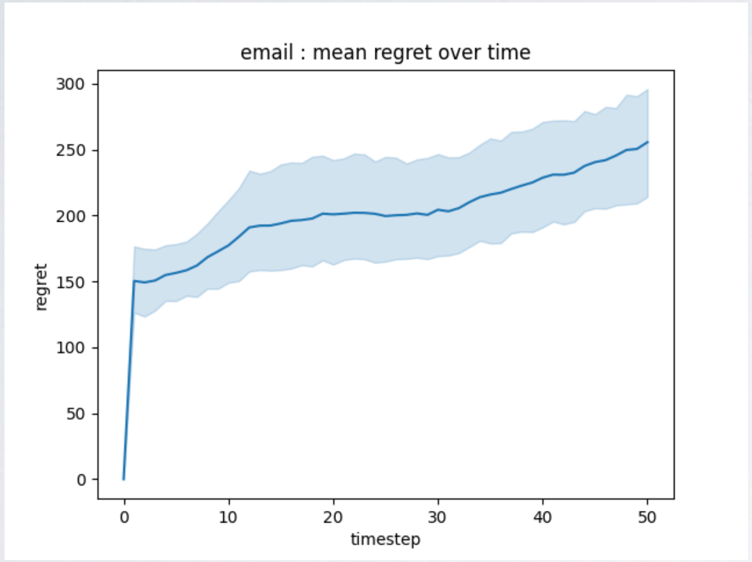


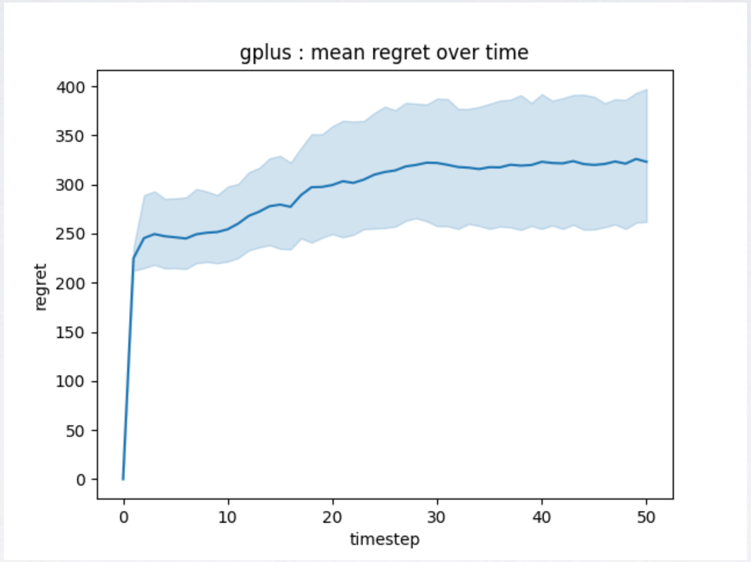
For this scenario, 10 experiments were performed with the following parameters:

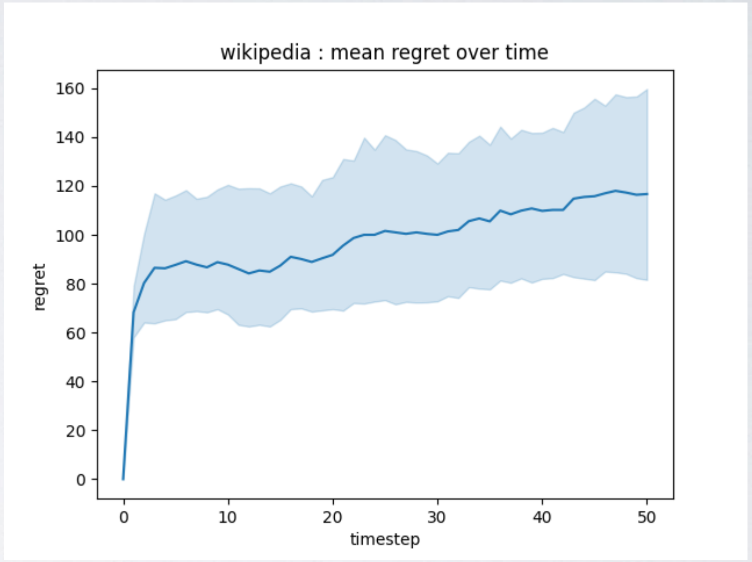
- 300 nodes
- Total budget to allocate: 5
- 2 propagation cascade steps
- Time horizon: 50

The results are plotted as the mean cumulative sum of regret for the single social networks, product pairs.

LINUCB PERFORMANCE PER SOCIAL NETWORK







As expected, the bandit algorithm provides sub-linear regret. The noisiness in the plot is due to the sampling of the rewards for both the clairvoyant and learner algorithms, with the latter outperforming the former at times because of the stochasticity of such sampling.

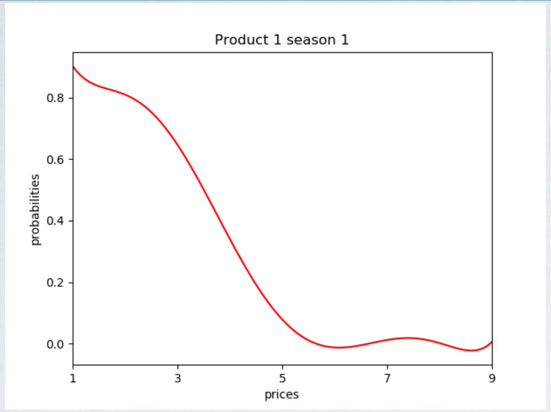
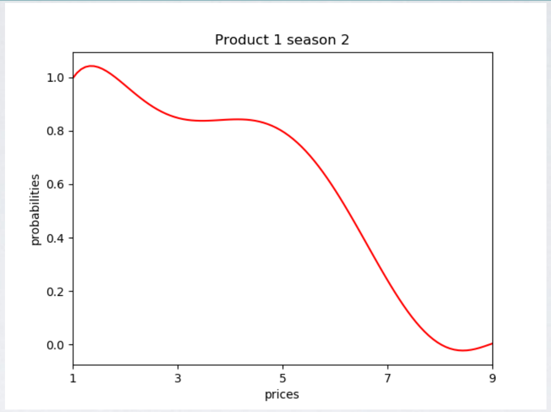
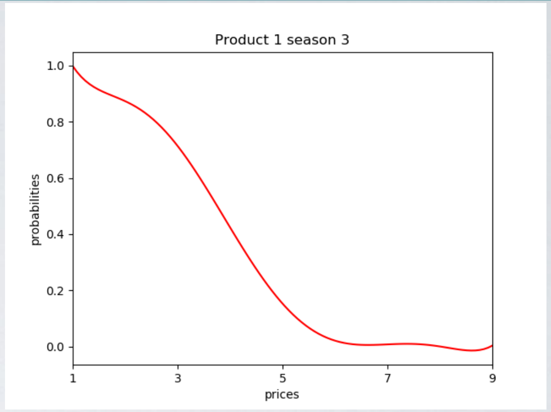
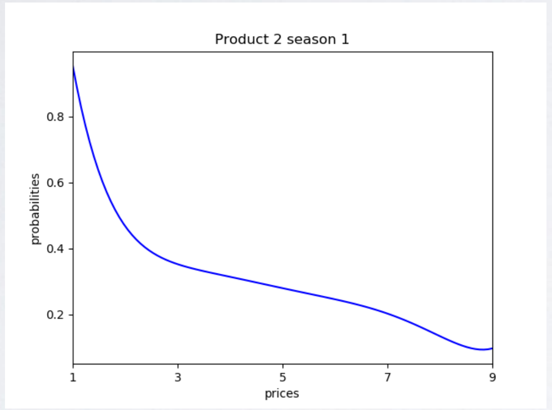
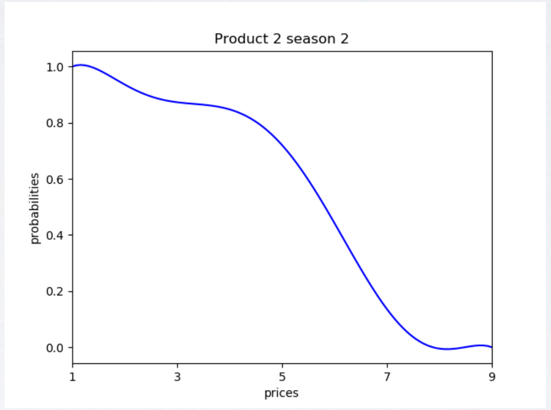
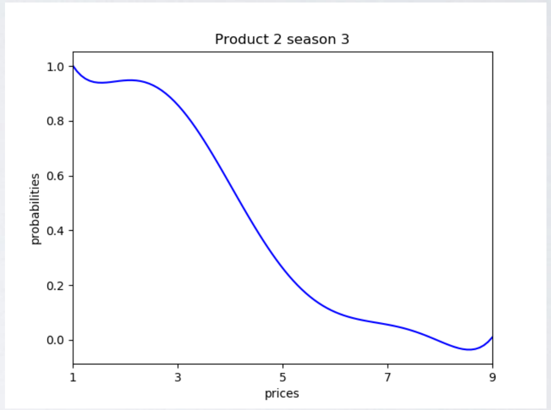
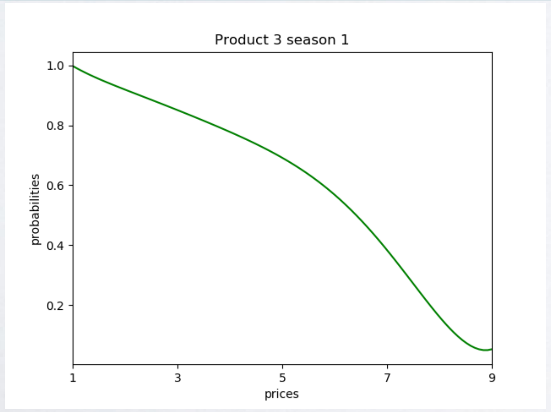
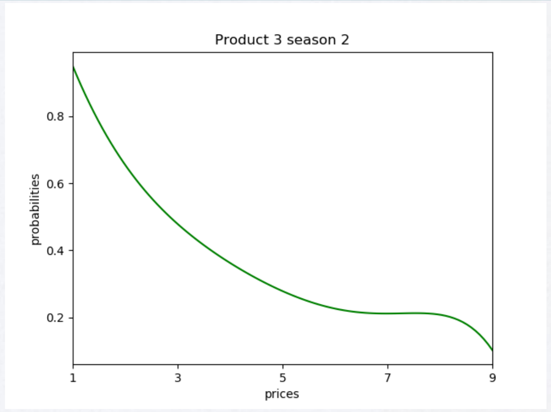
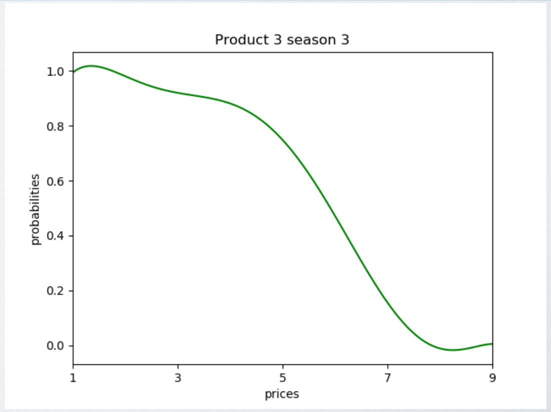
PRICING

We imagined selling three distinct products, one for each of the social network that we considered in the social influence maximisation phase.

We considered cheap products that follow three abrupt main seasonal changes, all of them sold with a price between 1 and 9 euros, with gaps of 2 euros. We considered seasonal phases as groups of four months in the year, with the first phase being from January to April, the second from May to August and so on.

The products we imagined are:

- **Product 1**: sunscreen, which sells best in the second quarter of the year (warmer months).
- **Product 2**: iced coffee ice cream, which sells best in the second quarter of the year, but still holds part of its value in September and October
- **Product 3**: pile scarf, which it is mainly used during winter time.

	January-April	May-August	September-December
Sunscreen			
Ice Cream			
Scarf			

CONVERSION RATE CURVES

In the previous slide we plotted the conversion rate curves for each product during their seasonal phases. On the x axis we placed the prices in increasing order, so the curves follow the same pattern: they decrease monotonically from 1 to 0 with different concavities because we assumed that a user would buy a product that costs less rather than the same product but with higher price.

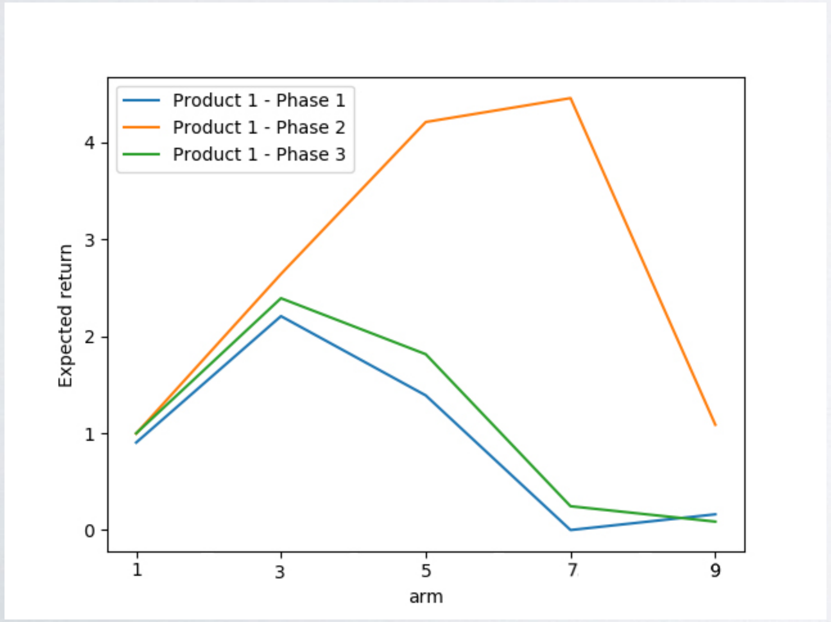
The different concavities between the phases are due to the fact that a product may be considered more valuable during some phase and, therefore, users may be willing to pay more for it.

OPTIMAL ARMS FOR PHASES

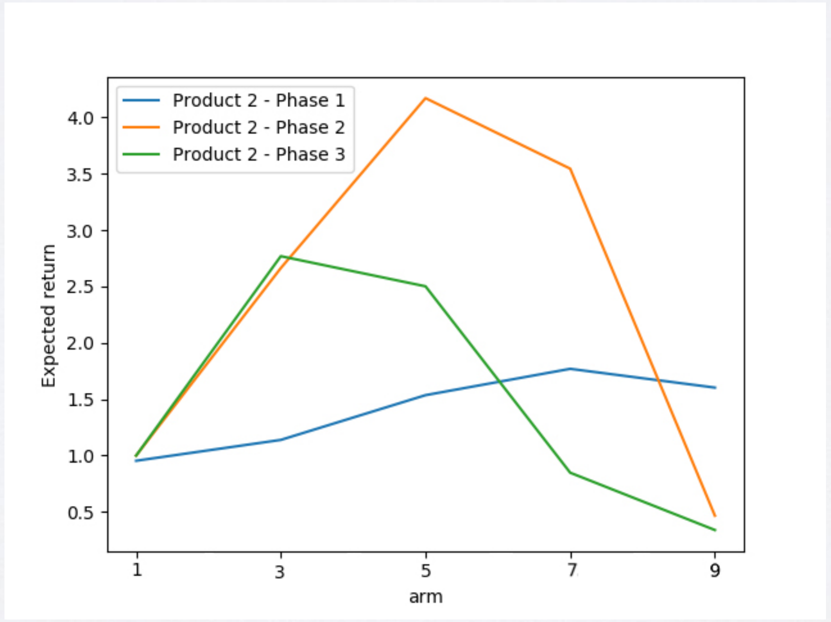
In the following graphs is it possible to see how the optimal price changes depending on the product and the season:

- The sunscreen (product 1) is mainly sold during summer time
- Also the ice cream (product 2) has more value during phase 2
- The scarf's value (product 3) rises during colder months.

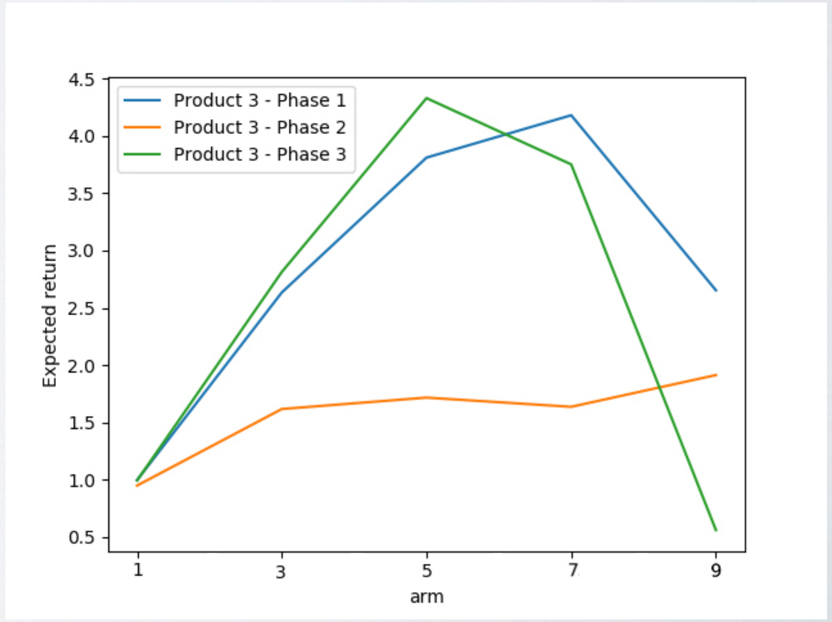
Sunscreen



Ice Cream



Scarf



STATIONARY SCENARIO - I

First we considered a stationary environment case, in which we disregarded any variations caused by the change of season. We used for each product the curve corresponding to their first phase throughout the whole simulation.

At each time step we performed the following:

1. Weighted influence maximisation, weighing each social's influence by its current best performing price, estimated by the bandit learner.
2. For each social network:
   A. For each user apply the bandit pricing sequence (see next slide).
   B. Compute the regret for the current time step and add it to the cumulative regret.

STATIONARY SCENARIO - II

For each user we applied the bandit algorithm loop:

1. Apply the bandit algorithm selection policy to choose the price to offer to the user.
2. Collect the binary reward as buy/not buy, sampling from a Bernoulli distribution with probability equal to the conversion rate for the selected price at the current phase.
3. Update the learner.
4. Collect the reward for the clairvoyant algorithm, sampling from a Bernoulli distribution with probability equal to the conversion rate for the optimal price at the current phase.

STATIONARY SCENARIO - III

The regret for the timestep is computed as:

$$R_T = \sum_{i=1}^{n^*} \theta^* \cdot p^* - \sum_{j=1}^{m} \theta_j \cdot p_j$$

With:

- $p^* = argmax\{cr_a \cdot p_a\}$, price of the arm with the best performance.
- p_j, price (arm) selected to be offered to the j-th user.
- θ^*, sample from a Bernoulli with probability equal to the conversion rate associated to the optimal arm.
- θ_j, sample from a Bernoulli with probability equal to the conversion rate associated to the arm selected for j-th user.
- n^* optimal number of users for the current timestep, computed by the social influence algorithm.
- m number of users computed by sampling the activations with the seeds provided by the joint influence maximiser.

PRICING LEARNERS - I

In order to have a performance comparison we applied both UCB1 and Thompson Sampling, with the second one outperforming the first one on all the considered (social network, product) pairs.

Both learners showed sub-linear cumulative regret performance, as expected.

In order to take into account the prices associated with the arms, few changes have been done on how the two algorithm pull arms and update their distributions. These variations are explained in the following slides.

THOMPSON SAMPLING

In Thompson Sampling the selection of the arm to be pulled was modified to take into account that the regret is computed with respect to the price and conversion rate, instead of the simple conversion rate:

For Thompson sampling: $i = argmax_i\{\widetilde{\theta}_i * p_i,\ i = 1, 2, 3, 4, 5\}$

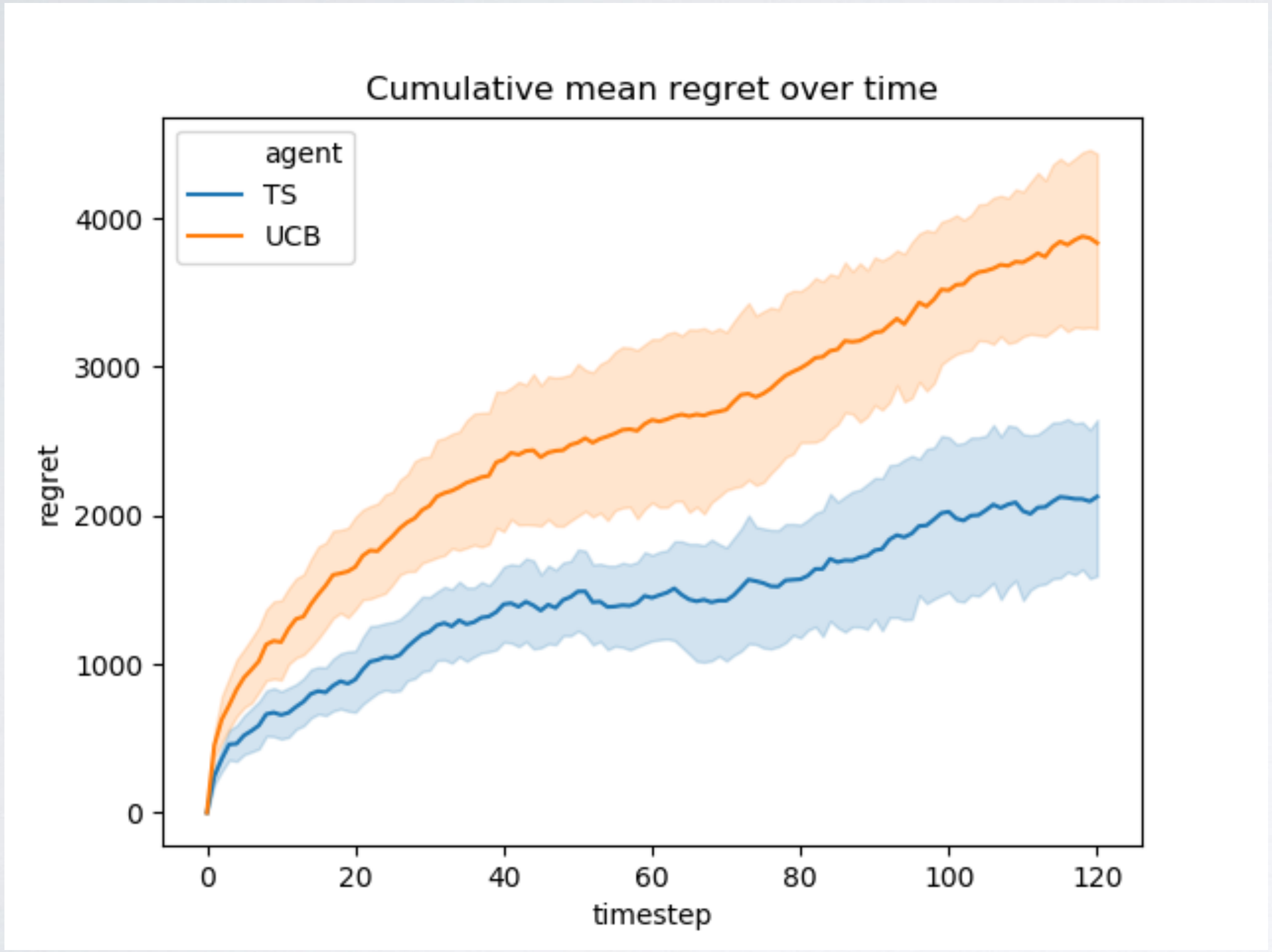
This allows to keep using the standard Beta distributions for each arm and consider the Bernoulli reward (buy/not buy) for updating the arms.

UCB

Since UCB doesn't assume any prior for the reward distributions, we decided to use the price as reward when the item is sold, zero otherwise.

The selection policy for UCB already takes into account arbitrarily distributed rewards, so we didn't need to modify it unlike the TS algorithm.

STATIONARY SCENARIO RESULTS

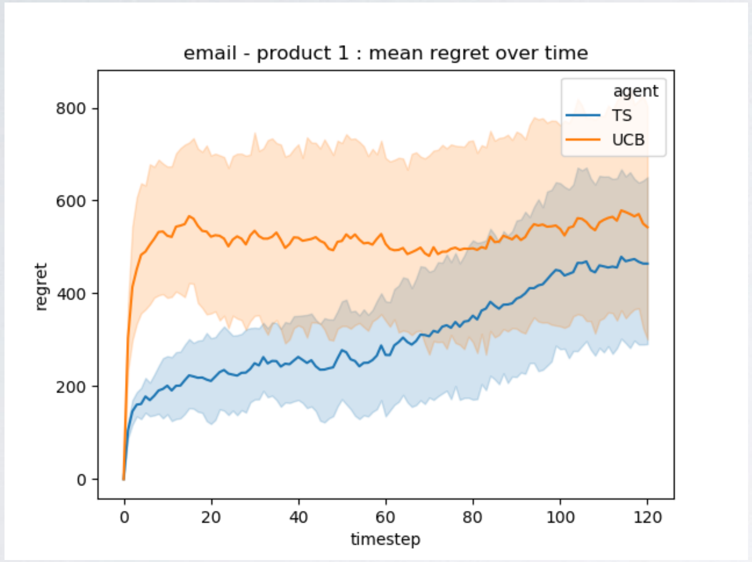


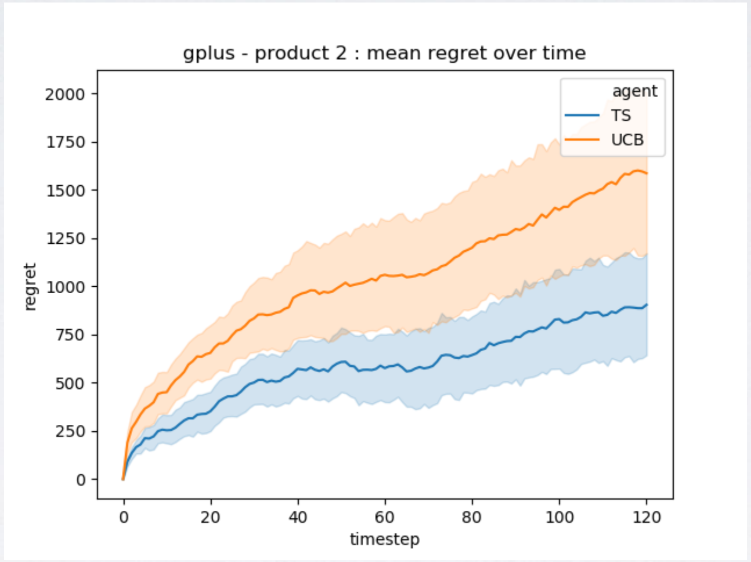
For this scenario, 50 experiments were performed with the following parameters:

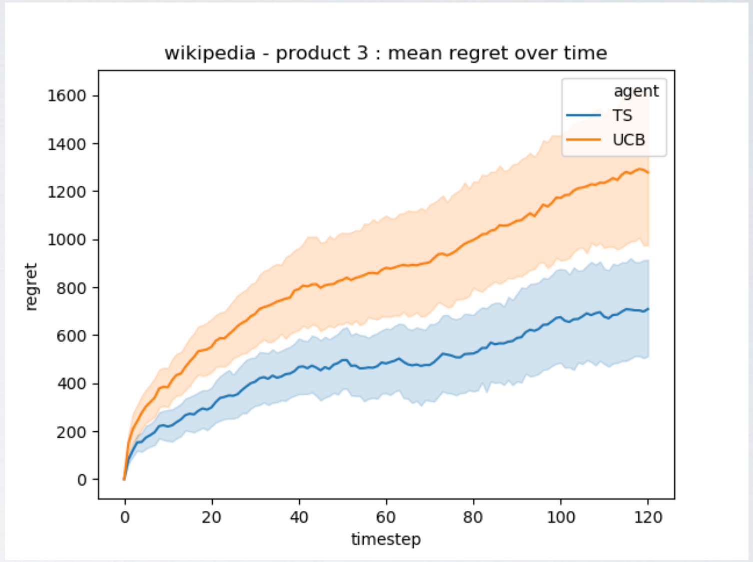
- 500 nodes
- Total budget to allocate: 5
- 3 propagation cascade steps
- Time horizon: 120 days

The results are plotted as the mean cumulative sum of regret for the single social networks, product pairs.

SOCIAL NETWORK, PRODUCT PAIR PERFORMANCES







Even if both the employed algorithms provide a sub-linear regret, the result of the experiments show that Thompson Sampling is able to completely outperform UCB-1 in this setting. The noisiness in the plot is due to the sampling of the rewards for both the clairvoyant and learner algorithms, with the latter outperforming the former at times because of the stochasticity of such sampling.

NON-STATIONARY SCENARIO - I

The second pricing phase extends what done in the stationary environment to a non-stationary scenario, where the product is evaluated over three seasons such that changes from one season the other are abrupt. Therefore, three different curves for each product were used and the duration of each phase corresponds to $1/3$ of the whole time horizon.

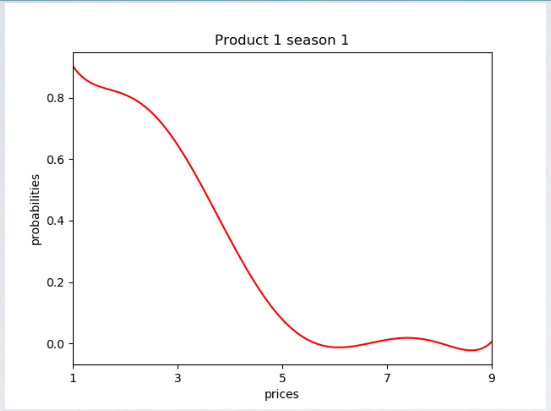
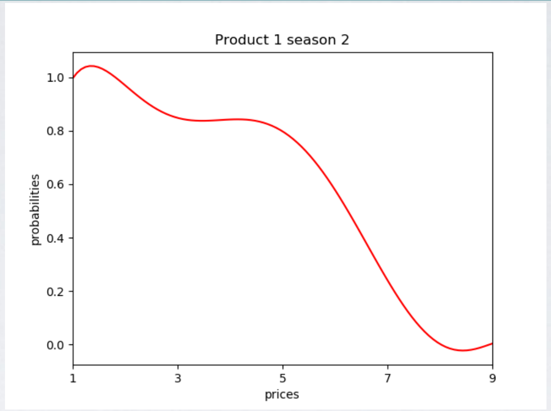
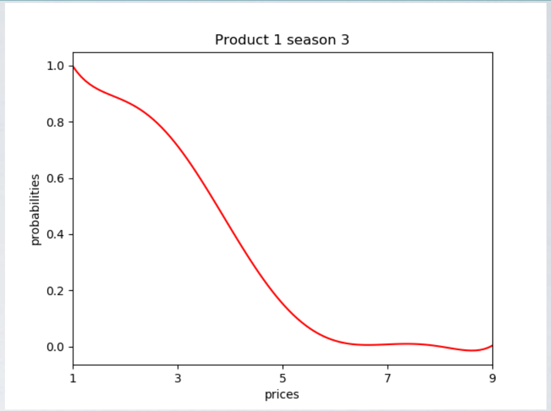
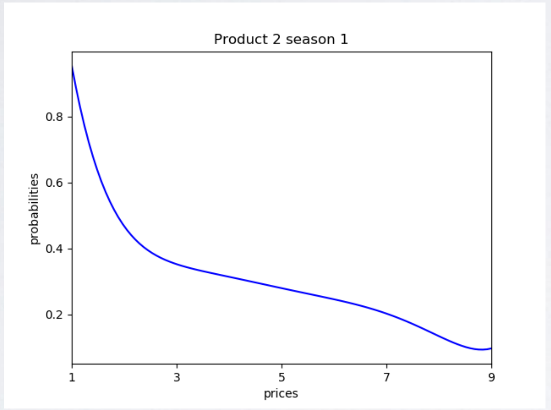
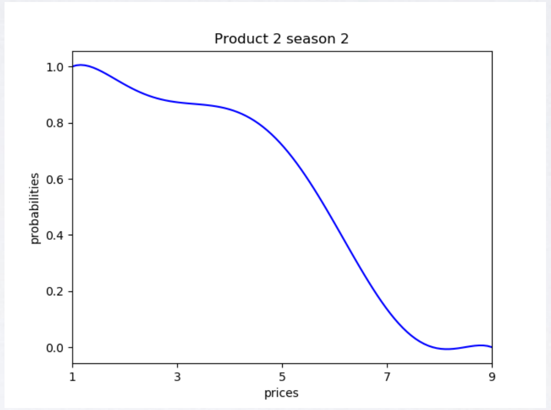
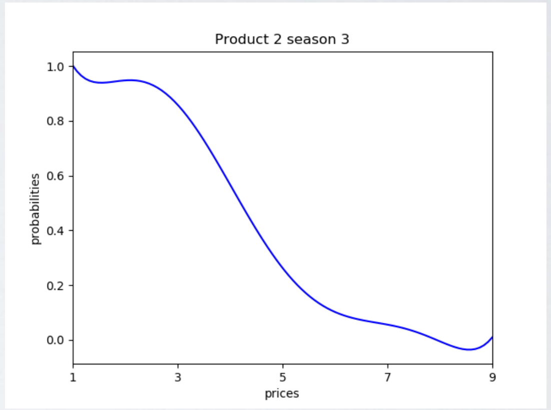
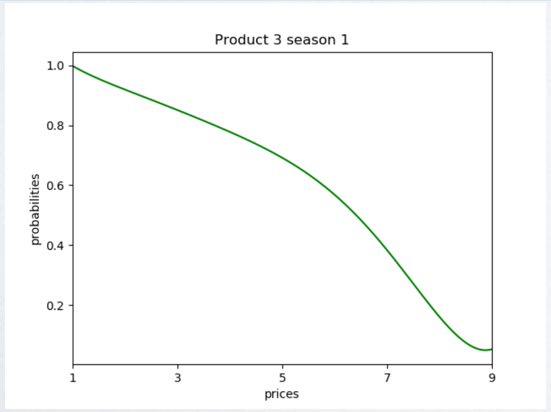
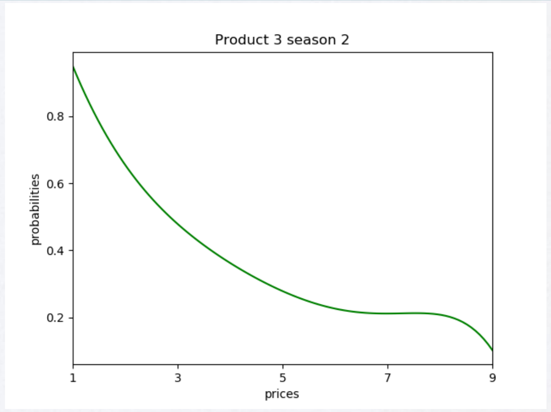
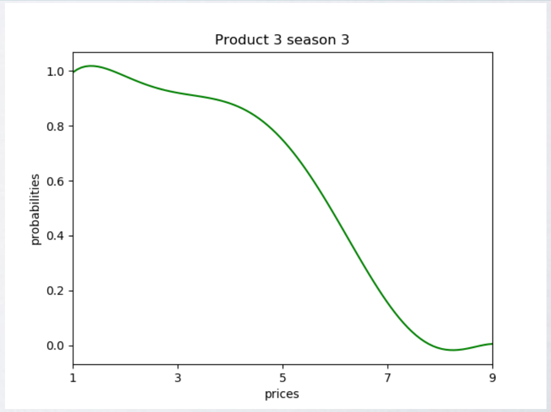
At each time step we performed the following:

1. Weighted influence maximisation, weighing each social's influence by the current best performing price, estimated by the bandit learner.
2. For each social network:
   A. For each user apply the pricing bandit loop (see next slide).
   B. Compute the regret for the current time step and add it to the cumulative regret.

NON-STATIONARY SCENARIO - II

For each user we applied the bandit algorithm loop:

1. Apply the bandit algorithm selection policy to choose the price to offer to the user.
2. Collect the binary reward as buy/not buy, sampling from a Bernoulli distribution with probability equal to the conversion rate for the selected price at the current phase.
3. Update the learner.
4. Collect the reward for the clairvoyant algorithm, sampling from a Bernoulli distribution with probability equal to the conversion rate for the optimal price at the current phase.

	January-April	May-August	September-December
Sunscreen			
Ice Cream			
Scarf			

NON-STATIONARY SCENARIO - III

The regret for the timestep is computed as:

$$R_T = \sum_{i=1}^{n^*} \theta_\varphi^* \cdot p_\varphi^* - \sum_{j=1}^{m} \theta_{j,\varphi} \cdot p_{j,\varphi}$$

With the subscript φ denoting the fact that the conversion rate curves are dependent on the current phase, thus changing the optimal arm and the mean values for the Bernoulli samples. The equation and the variables stay the same of the stationary scenario.

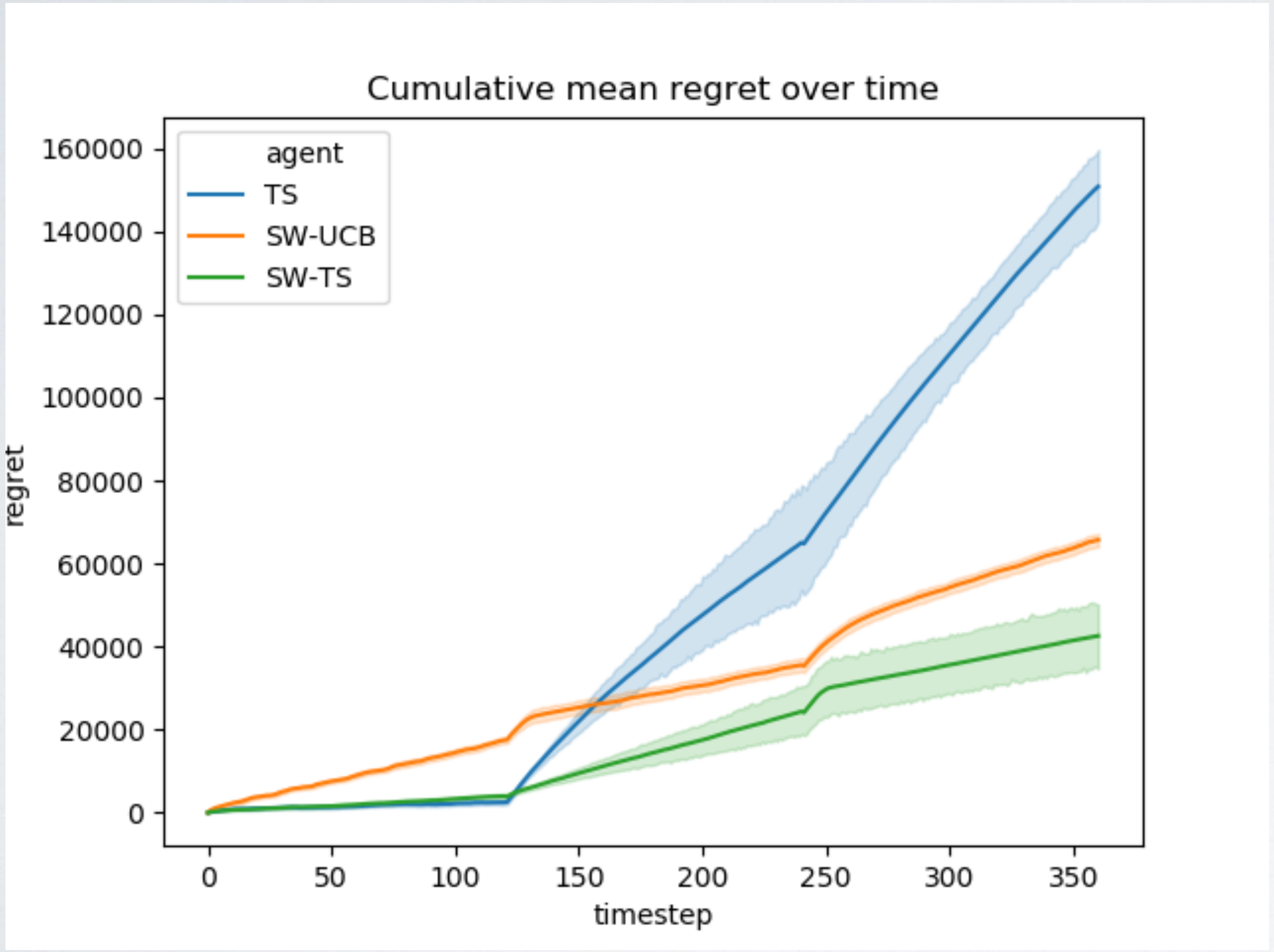
PRICING LEARNERS - II

In order to have a performance comparison we applied sliding window algorithms, together with the traditional UCB1 and Thompson Sampling. In particular Sliding Window Thompson Sampling and Sliding Window UCB have been used.

The experiments we performed highlighted that whereas SW-UCB has lower variance, it is outperformed in mean value by SW-TS. Both learners showed sub-linear cumulative regret performance, as expected.

As a benchmark, we also employed the classic version of TS which, despite being the best performing algorithm in the first phase, showed cumulative linear regret and globally performed worse than the other two. This is due to the fact that in the first phase the algorithm can exploit all the samples to better estimate the conversion rates, whereas later on the SW algorithms are able to "forget" samples from previous phases and considering for the estimation only the most recent samples.

NON- STATIONARY SCENARIO RESULTS

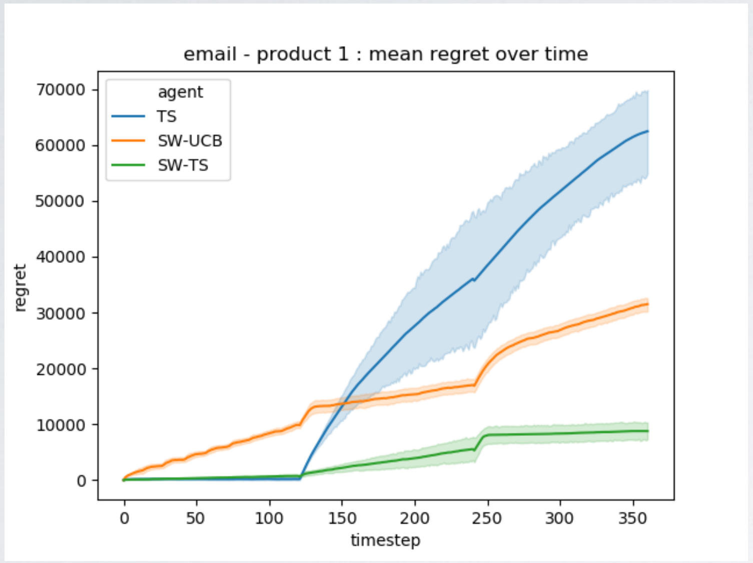


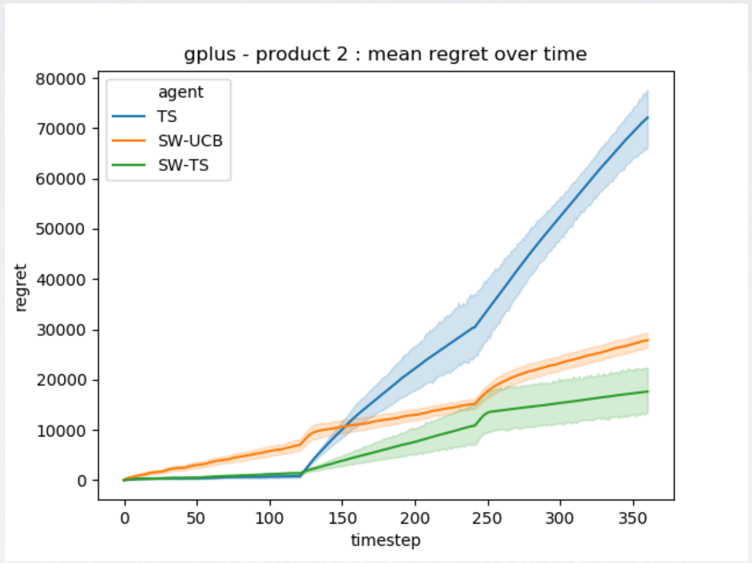
For this scenario, 15 experiments were performed with the following parameters:

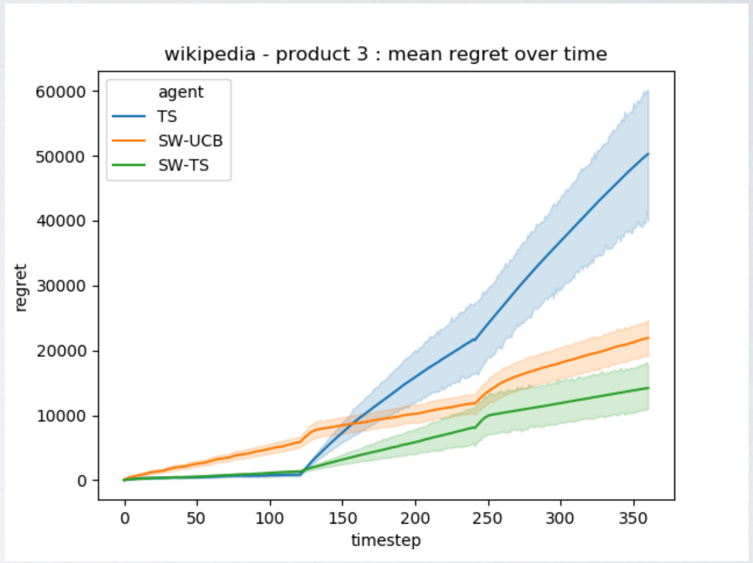
- 300 nodes
- Total budget to allocate: 5
- 3 propagation cascade steps
- Time horizon: 360 days

In the figure we plot the mean cumulative sum of regret for the single social networks, product pairs.

SOCIAL NETWORK, PRODUCT PAIR PERFORMANCES







It is evident from the plots that the sliding-window algorithms outperform classic Thompson Sampling in the long run, whereas the latter is capable of exploiting all the available information for the first phase, thus outperforming the former.

PUTTING IT ALL TOGETHER - I

For the last scenario we combined what we implemented in both Social Influence and Pricing. In particular, the estimation of the prices in a stationary environment has been done while gradually learning the graphs' activation probabilities and maximising the social influence.

We deployed two classes of learners, one for learning the edge activation probabilities and one for the conversion rate estimation.

We used the **LinUCB** bandit algorithm to estimate the edge activation probability matrices, considering as reward a sample from a Bernoulli distribution with probability equal to the edge activation probability.

To estimate the stationary conversion rate probabilities we used again **Thompson Sampling**.

PUTTING IT ALL TOGETHER - II

Starting from unknown probability matrices, at each timestep we iterated the following:

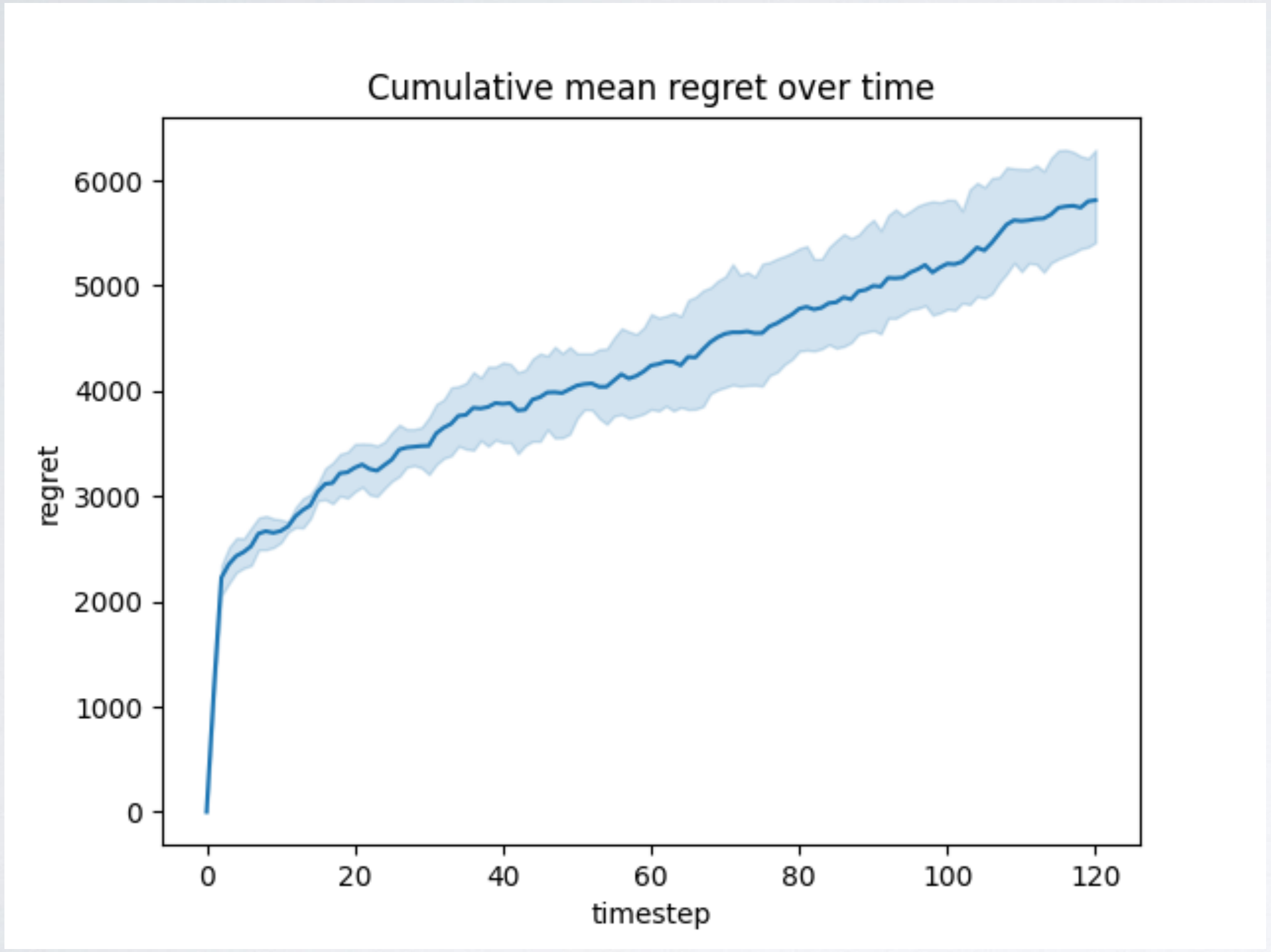
1. Allocation of the budget between the three social networks through joint influence maximisation, using the estimated probability matrices and currently best performing price.
2. Simulation of the effect of the selected seeds on the social network graphs, observing the activated edges.
3. Pricing bandit for each user:
    A. Selection of the price to offer to the user
    B. Observation of the reward and update of the bandit distributions/bound
4. Sample of the reward for the LinUCB algorithm and update of the activation estimated matrices.

REGRET

$$R_T = \sum_{i=1}^{n^*} \theta^* \cdot p^* - \sum_{j=1}^{m} \theta_j \cdot p_j$$

The regret is calculated as in exercise 5. This time the number of clicks *m* is computed also considering the estimated matrices, leading to a larger uncertainty (double error source).

RESULTS

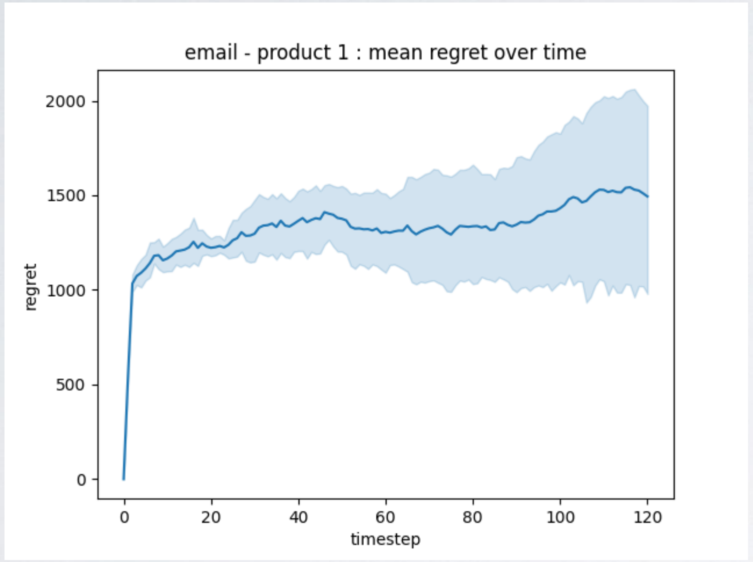


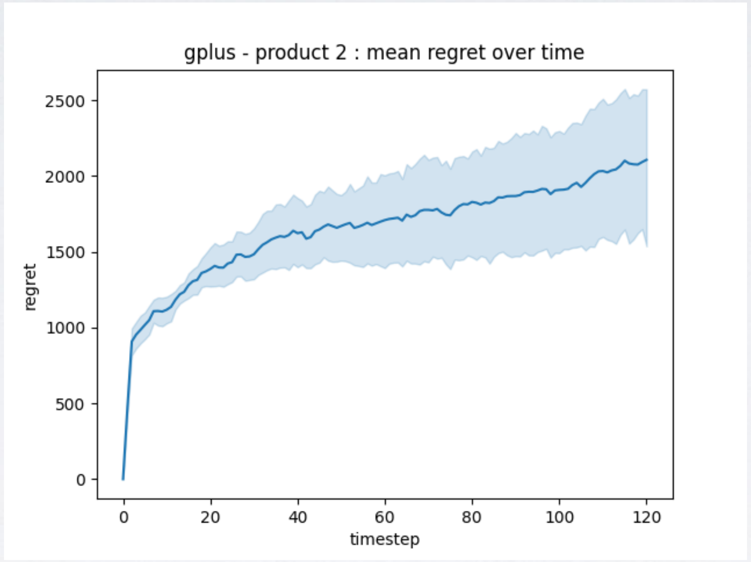
For this scenario, due to the long execution time, only 5 experiments were performed with the following parameters:

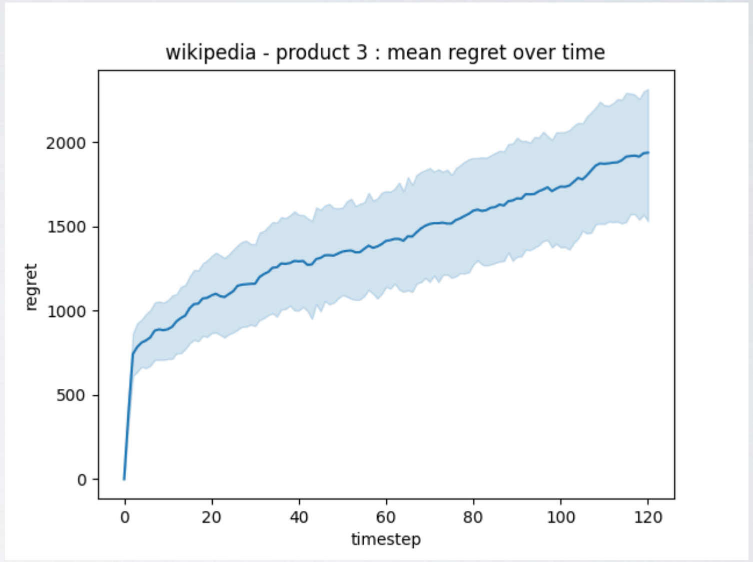
- 300 nodes
- Total budget to allocate: 5
- 2 propagation cascade steps
- Time horizon: 120

In the figure we plot the mean cumulative sum of regret for the single social networks, product pairs.

SOCIAL NETWORK, PRODUCT PAIR PERFORMANCES







As expected, the regret shows a sub-linear behaviour even when a double source of uncertainty is present.

Politecnico di Milano
A.Y. 2019-2020

POLITECNICO
MILANO 1863

THANK YOU

Diego Piccinotti, Federica Ilaria Cesti, Lorenzo Casalini,
Matteo Carretta, Umberto Pietroni