# Project Proposals for
# Online Learning Applications

In the following, you can find three project proposals. All the project proposals share a common part in which you need to resort to social influence techniques. The three project proposals distinguish for the second part, in which, instead, you need to resort to pricing, advertising, or matching techniques respectively.

## Part Common to the Three Project Proposals

Imagine an ecommerce website which can sell an unlimited number of units of 5 different items without any storage cost.

In every webpage, a single product, called *primary*, is displayed together with its price. The user can add a number of units of this product to the cart. After the product has been added to the cart, two products, called *secondary*, are recommended. When displaying the secondary products, the price is hidden. Furthermore, the products are recommended in two slots, one above the other, thus providing more importance to the product displayed in the slot above. If the user clicks on a secondary product, a new tab on the browser is opened and, in the loaded webpage, the clicked product is displayed as primary together with its price. At the end of the visit over the ecommerce website, the user buys the products added to the cart.

We assume that the user has the following behavior:

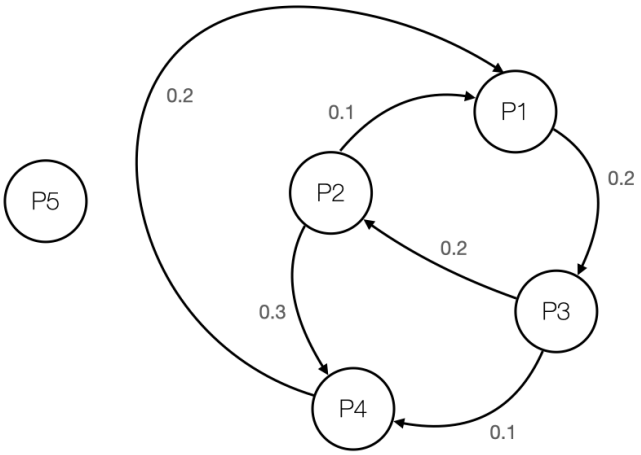
- she/he buys a number of units of the primary product if the price of a single unit is under the user' reservation price; in other words, the users' reservation price is not over the cumulative price of the multiple units, but only over the single unit;
- once the primary product has been bought, the user clicks on a secondary product with a probability depending on the primary product except when the secondary product has been already displayed as primary in the past, in this case the click probability is zero (thus, in practice, excluding the case in which a product is displayed as primary more than once --- as a result the number of webpages visited by the user is finite);
- when observing the secondary products, the user initially observes the first slot and, after having observed that slot, observes the second slot. Assume that the probability with which the first slot is observed is 1, while the probability with which the second slot is observed is lambda < 1. The value of lambda is assumed to be known in all the three project proposals.

Notice that the probability of a click on a secondary product depends on:

- the purchase probability of the primary product,

- the probability to observe the slot in which the secondary product is displayed, and the click probability of the secondary product conditioned to the purchase of the primary. CTR

We provide the following simple example to clarify the setting better.



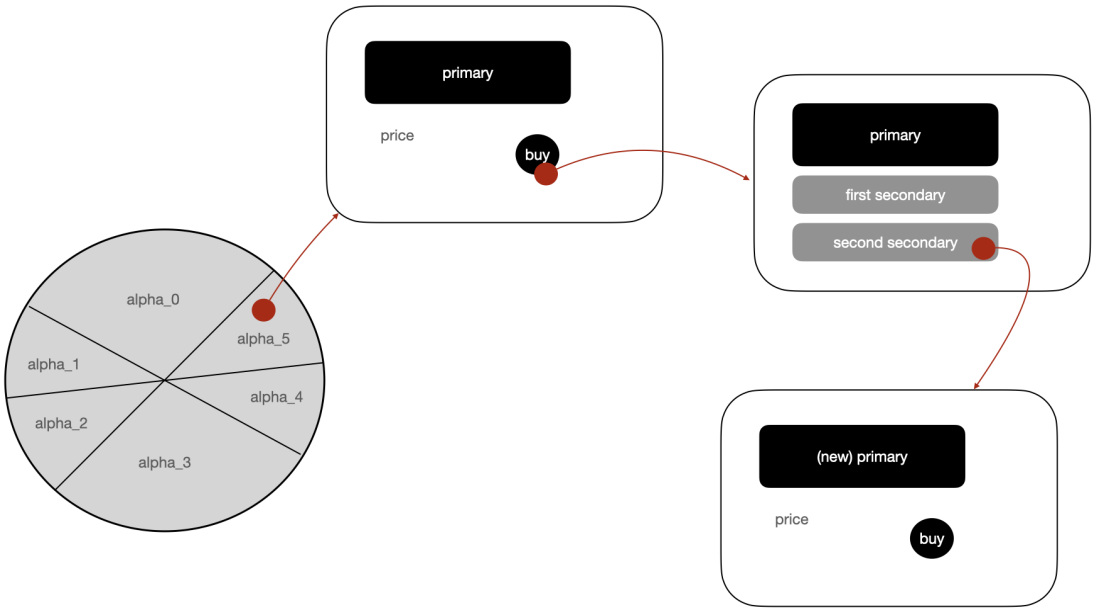

We are given five products P1, P2, P3, P4, P5, each corresponding to a node of the graph above. Every directed edge between two products is associated with a weight. Given a primary product P and a secondary product P', the weight of the edge starting from P and ending in P' provides the probability with which the user clicks on P' while displayed in the first slot with product P as the main product. For instance, if the primary product is P2 and the secondary products are P1, in the first slot, and P4, in the second slot, the probabilities that the user clicks on P1 and P4 are 0.1 and 0.3 * lambda. The absence of an edge means that the corresponding probability is zero. Consider now another case. When the primary product is P2 and the secondary products are P1 and P4, the user can buy a number of units of P2 and then click on one of both secondary products. Assume, for instance, that the user clicks on P4 and therefore P4 is displayed as the primary product, along with, e.g., P1 and P3 displayed as secondary products. Notice that, from now on, if P2 is displayed as a secondary product, the probability that the user clicks on it is zero independently of the slot in which it is displayed.

For simplicity, we make the following assumptions:

- the clicks on the secondary products are independent of each other and the user can click on multiple secondary products, so as to activate multiple parallel paths over the graph depicted above;
- the number of items a user will buy is a random variable independent of any other variable; that is, the user decides first whether to buy or not the products and, subsequently, in the case of a purchase, the number of units to buy;
- the actions performed by the users are perfectly observable by the ecommerce website.

Every day, there is a random number of potential new customers (returning customers are not considered here). In particular, every single customer can land on the webpage in which one of the 5 products is primary or on the webpage of a product sold by a (non-strategic) competitor. Call $\alpha$_i the ratio of customers landing on the webpage in which product Pi is primary, and call $\alpha$_0 the ratio of customers landing on the webpage of a competitor. In practice, you can only consider the $\alpha$ ratios and disregard the total number of users. However, the $\alpha$ ratios will be subject to noise. That is, every day, the value of the $\alpha$ ratios will be realizations of independent Dirichlet random variables.

The following picture summarizes the overall scenario.




In the following project proposals, we ask you to develop two different settings differing for the graph weights. First, assume that the graph is fully connected and therefore all the edges have strictly positive probabilities (in practice the secondary products displayed are just two and thus many of the edges are useless). In the second, assume that the graph is not fully connected and therefore some edges have zero probability.

The behavior of the user in the graph is similar to that of the social influence. Thus, in the following project proposals, you need to resort to social influence techniques to evaluate the probabilities with which the user reaches the webpage with some specific primary product.

## Project Proposal #1 : Pricing

Consider the scenario in which:

- for every primary product, the pair and the order of the secondary products to display is fixed by the business unit and cannot be controlled,
- the price of every primary product is a variable to optimize,
- the expected values of the $\alpha$ ratios are known.

For simplicity, assume that there are four values of price for every product and that the price can be changed once a day. For every product, order the prices in increasing levels. Every price is associated with a known margin. On the other hand, for every product, the conversion probability associated with each price value is a random variable whose mean is unknown.

*Step 1: Environment.* Develop the simulator by Python. In doing that, imagine a motivating application and specify an opportune choice of the probability distributions associated with every random variable. Moreover, assume that there are 2 binary features that define 3 different user classes. The users' classes potentially differ for the demand curves of the 5 products, number of daily users, $\alpha$ ratios, number of products sold, and graph probabilities. That is, for every random variable, you need to provide three different distributions, each one corresponding to a different users' class.

*Step 2: Optimization algorithm*. Formally state the optimization problem where the objective function is the maximization of the cumulative expected margin over all the products. Design a greedy algorithm to optimize the objective function when all the parameters are known. The algorithm works as follows. At the beginning, every item is associated with the corresponding lowest price. Then, evaluate the marginal increase obtained when the price of a single product is increased by a single level, thus considering 5 potential different price configurations at every iteration, and choose the price configuration providing the best marginal increase (a price configuration specifies the price of every product). The algorithm stops when no new configuration among the 5 evaluated is better than the previous one. For instance, at the beginning, evaluate the 5 price configurations in which all the products are priced with the lowest price except for one product which is priced with the second lowest price. If all these price configurations are worse than the configuration in which all the products are priced with the lowest price, stop the algorithm and return the configuration with the lowest price for all the products. Otherwise, choose the best price configuration and re-iterate the algorithm. Notice that the algorithm monotonically increases the prices as well as the cumulative expected margin. Therefore, the algorithms cannot cycle. However, there is not guarantee that the algorithm will return the optimal price configuration. Develop the algorithm by Python.

*Step 3: Optimization with uncertain conversion rates*. Focus on the situation in which the binary features cannot be observed and therefore data are aggregated. Design bandit algorithms (based on UCB and TS) to face the case in which the conversion rates are unknown. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 4: Optimization with uncertain conversion rates, $\alpha$ ratios, and number of items sold per product*. Do the same of Step 3 when also the alpha ratios and the number of items sold per product are uncertain. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 5: Optimization with uncertain graph weights*. Do the same as Step 3 when the uncertain parameters are the graph weights. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 6: Non-stationary demand curve*. Now assume that the demand curves could be subjected to abrupt changes. Use a UCB-like approach with a change detection algorithm to face this situation and show whether it works better or worse than using a sliding-window UCB-like algorithm. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 7: Context generation*. Do the same of Step 4 when the features can be observed by the ecommerce website. For simplicity, run the context-generation algorithms only every 2 weeks. When we have multiple contexts, the prices of each single context can be chosen and thus optimized independently of the others. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

For the Steps 3-7, in the algorithm evaluation, report:

- the average regret and reward computed over a significant number of runs,
- their standard deviation,
- when theoretical bounds are available, also report the ratio between the empiric regret and the upper bound.

## Project Proposal #2 : Advertising

Consider the scenario in which:

- for every primary product, the pair and the order of the secondary products to display is fixed by the business unit and cannot be controlled,
- the price of every primary product is fixed and this price is equal, for simplicity, to the margin,
- there are 5 advertising campaigns, one per product, and, by a click to a specific ad, the user lands on the corresponding primary product.

The ecommerce website has a budget cap B to spend to advertise its products. For simplicity, assume that the automatic bidding feature provided by the platforms is used and therefore no bidding optimization needs to be performed. The change of the budget spent on the campaign for the product Pi changes the expected value of the corresponding $\alpha$_i, and therefore the number of users landing on the webpage in which product Pi is primary. For every campaign, you can imagine a maximum expected value of $\alpha$_i (say $\alpha$_i_bar) corresponding to the case in which the budget allocated on that campaign is infinite. Therefore, for every campaign, the expected value of $\alpha$_i will range from 0 to $\alpha$_i_bar, depending on the actual budget allocated to that campaign.

*Step 1: Environment.* Develop the simulator by Python. In doing so, imagine a motivating application and choose the probability distributions associated with every random variable. Assume that there are 2 binary features that define 3 different users' classes. The users' classes potentially distinguish for the $\alpha$ ratio functions. More precisely, every class is characterized by a profile of $\alpha$ functions, one per campaign. Given a campaign, the three classes may have different $\alpha$ functions.

*Step 2: Optimization algorithm*. Formally state the optimization problem where the objective function is the profit, defined as the difference between the expected margin and the spent in advertising. Design an exact, dynamic-programming algorithm to optimize the function when all the parameters are known. Notice that the value per click provided by an advertising campaign is the expected margin from landing to the corresponding product. Furthermore, in the dynamic-programming algorithm, you can find the best way to spend the budget, for every feasible value of the budget, neglecting that the budget spent is to subtract from the objective function. Thus, you can use the standard dynamic-programming algorithm. After you have filled the entire dynamic-programming table, you can subtract the corresponding budget spent from the value of every cell of the last row and then look for the best allocation maximizing the difference between expected value and budget spent. Deveop the algorithm by Python.

*Step 3: Optimization with uncertain $\alpha$ functions*. Focus on the situation in which the binary features cannot be observed and therefore data are aggregated. Design bandit algorithms (combining GP with UCB and TS) to face the case in which the alpha functions are unknown. Deveop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 4: Optimization with uncertain $\alpha$ functions and number of items sold*. Do the same of Step 3 to the case in which also the number of items sold per product are uncertain. Deveop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 5: Optimization with uncertain graph weights*. Do the same as Step 3 in the case the only uncertain parameters are the graph weights. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

maybe the expected margin

*Step 6: Non-stationary demand curve*. Now assume that the demand curves could be subjected to some abrupt changes. Use a UCB-like approach with a change detection algorithm to face this situation and show whether it works better or worse than using a sliding-window UCB-like algorithm. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 7: Context generation*. Do the same of Step 4 when the features can be observed by the ecommerce website. For simplicity, activate the context-generation algorithms every 2 weeks. With multiple contexts, each single context can be optimized independently of the others. Notice that the split is performed simultaneously over all the campaigns. For instance, if you decide to split over a binary feature, then the split will be performed over all the 5 campaigns simultaneously. This is done to simplify the number of possible splits one can deal with. Once you splitted in multiple contexts, every context can be used to target the ads at best. More precisely, features can be used as target information of the campaigns, thus leading to replicate the campaigns. For instance, if the algorithms split according to a binary feature generating two contexts, then the 5 campaigns are replicated for every single context, say C1 and C2. Reasonably, C1 and C2 will be characterized by different $\alpha$ functions. In this case, the budget optimization problem includes 10 campaigns. This is equivalent to say that we need to decide whether to assign more budget, e.g., to the campaign of P1 for C1 rather than to the campaign of P1 for C2. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

For the Steps 3-7, in the algorithm evaluation, report:

- the average regret and reward computed over a significant number of runs,
- their standard deviation,
- when theoretical bounds are available, also report the ratio between the empiric regret and the upper bound.

## Project Proposal #3 : Matching

Consider the scenario in which:

- for every primary product, the pair and the order of the secondary products to display is fixed by the business unit and cannot be controlled,
- the price of every primary product is fixed and the margin is equal to difference between the price and a fixed known cost,
- the $\alpha$ ratios are fixed and known,
- the user can be gifted a promo code after a purchase to incentivize a new purchase next month; the value of the promo code to provide to each user is a variable to optimize.

As customary, the user pays for all the products added to the cart simultaneously. After having paid for the products, the ecommerce website can decide to give or not a promo code, and, in the latter case, which promo code to give. We make the following assumptions:

- the promo code can be used to get for free a single product, that is, the discount is 100%; thus, we can just decide whether to provide a 100% discount promo or no discount;
- selling a product for free is a cost for the ecommerce website;
- a returning user with a promo code lands on the webpage where the gifted product is primary with a given strictly positive probability, whereas she/he lands on the webpage of other products with zero probability;
- users without any promo code return to the ecommerce website with a positive probability, but this probability is much lower than the probability when a promo code is gifted;
- the probability that a user returns landing on the webpage of the gifted product Pi depends on the first primary product she/he bought in the first visit to the website; this results in a probability matrix (M) in which the rows correspond to the first primary product seen by the user, the columns correspond to the product in the promo code, and in the cell we have the returning probability; we also have another probability matrix (M0), in which the cells contain the probability that a user that has seen product Pi as first primary returns landing on the webpage of product Pj in the case no promo code has been gifted;
- the graph weights of the returning users may be different from those of the first-visit users' and may depend on the product gifted by the promo code; the secondary products to display are fixed and cannot be controlled.

*Step 1: Environment.* Develop the simulator by Python. In doing that, imagine a motivating application and design the probability distributions associated with every random variable. In principle, we assume that all the users behave in the same (stochastic) way before the first purchase, while the returning users may have different behaviors, and therefore they are divided in classes. These classes depend on the product gifted by the promo code.

*Step 2: Optimization algorithm*. Formally state the optimization problem of finding the best promo code for every user as a bipartite assignment problem. In the objective function, consider the expected profit due to the first visit of the users and their first return. Neglect the

profit due to the third visit and so on. Design an algorithm to solve such an optimization problem. Develop the algorithm by Python.

*Step 3: Optimization with uncertain M and M0*. Focus on the situation in which matrices M and M0 are unknown. Design bandit (UCB and TS) algorithms to face the case in which M and M0 are unknown. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 4: Optimization with uncertain M, M0 and returning users' graph weights*. Do the same of Step 3 to the case in which also the returning users' graph weights are unknown. Here, aggregate all the data on the returning users' weight graphs and use a single aggregate model. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 5: Optimization with uncertain graph weights*. Do the same as Step 3 in the case the only uncertain parameters are the (first-visit users') graph weights. That is, the graph weights of the returning users are known. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 6: Non-stationary demand curve*. Now assume that matrix M0 and M could be subjected to some abrupt changes. Use a UCB-like approach with a change detection algorithm to face this situation and show whether it works better or worse than using a sliding-window UCB-like algorithm. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

*Step 7: Context generation*. Do the same as Step 4 except that you can dynamically disaggregate over the returning users' graph weights according to the users' classes (they correspond to the promo codes). That is, at the beginning, you assume that every returning user has the same graph weights. Then, dynamically depending on data, you disaggregate data and model if this improves the ecommerce website's expected profit. Develop the algorithms by Python and evaluate their performance when applied to your simulator.

For the Steps 3-7, in the algorithm evaluation, report:

- the average regret and reward computed over a significant number of runs,
- their standard deviation,
- when theoretical bounds are available, also report the ratio between the empiric regret and the upper bound.