POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Online Advertising Campaign Management for Hotel Booking

Master of Science in Computer Science and Engineering - Ingegneria Informatica

Author: Luca Alessandrelli

Advisor: Prof. Francesco Trovò

Co-advisors: Ph.D. Alessandro Nuara, Dott. Giulia Romano

Academic year: 2020-2021

1. Introduction

The Online Advertising revenue in the United States increased from 126.4 billion in 2019 to 139.8 billion U.S. dollars in 2020. A further sign of its growth is that the worldwide spending in 2020 has been estimated to be 378 billion U.S. dollars and is forecast to increase in the following years, reaching 646 billion by 2024. All the opportunities that Online Advertising has brought for the advertising market have drawn a lot of attention from the scientific community. In particular, the *Artificial Intelligence* (AI) field played a crucial role in providing automatic mechanisms to support both the publishers and advertisers in their tasks. We specifically focus our attention on the advertiser's task, consisting in optimizing an online advertising campaign. A campaign is composed of multiple sub-campaigns, each one potentially having a different ad and target. The advertiser takes part in an auction carried out by the publisher, setting a bid value for each sub-campaign. A bid is the maximum amount of money the advertiser is willing to pay for a performed action on her ad, where the performed action depends on the payment scheme (e.g., pay-per-click). The goal of this work is to apply state-of-the-art AI technologies in a real-world Online Advertising scenario for the company AdsHotel, which is responsible for the optimization of the advertising campaigns of multiple hotels all around the world.

We built a real-world system exploiting existing AI's methods to perform safe bid optimization (i.e., guaranteeing a given return-on-investment (ROI) constraint) and extending them to adapt their behavior as the hotel environment changes. This is crucial due to the many reasons why it might present non-stationarity over time (e.g., seasonality, national holidays, and unpredictable events like COVID-19). More details are provided in Section 2. We also propose aggregation strategies to dynamically merge data generated by similar sub-campaigns. The need for such solutions is rooted in the scarcity of data caused by the traveling restrictions imposed for the COVID-19 pandemic. More details are provided in Section 3. Finally, we integrate into the system a strategy to provide suggestions on how to expand an ongoing campaign by selecting the most promising sub-campaigns a hotel might open. To accomplish this task, we combine a ranking system and change detection techniques, to monitor the performance trend of closed sub-campaigns. More details are provided in Section 4. By implementing our system in the AdsHotel environment, we evaluate its capabilities and measure its performance over multiple real-world hotel advertising campaigns.

Related Works The joint bid-budget optimization problem has been addressed for the first time by Zhang et al. [3]. They model it as a constrained optimization problem, maximizing the expected revenue while satisfying budget and bid range constraints. To solve such problem, the expected revenue as a function of the bid must be estimated for each sub-campaign. They estimate such functions by building a probabilistic model for Ad Ranking: exploiting search advertising log data, they compute the probability for a given item and bid to be ranked at a given slot position in the auction. However, this type of data is rarely available to the advertiser, thus this solution shows limitations in its applicability.
Nuara et al. [1] propose a different approach for the joint bid-budget optimization problem that does not require ad ranking data. They formulate the optimization problem as a combinatorial-bandit problem, in which a *superarm* corresponds to a combination of bid/budget pairs for each sub-campaign satisfying combinatorial budget constraints. Eventually, we narrowed our attention to the few works providing safe bid optimization methods. Specifically, we adopted the solution designed by Spadaro et al. [2] and built our system around it. Indeed, they focus on safe bid optimization in a scenario very similar to ours, in which ROI and budget constraints must be satisfied. For this reason, we implement their solution into our system and extend it to the specific case of hotel advertising.

2. Safe Bid Optimization with Return-on-Investment Constraints

We are given an advertising campaign $\mathcal{C} = \{C_1, \dots, C_N\}$, with $N \in \mathbb{N}$, where C_j is the j-th subcampaign, and a finite time horizon of $T \in \mathbb{N}$ days. In this work, as common in the literature on ad allocation optimization, we refer to a subcampaign as a single ad or a group of homogeneous ads requiring to set the same bid. For each day $t \in \{1, \dots, T\}$ and for every subcampaign C_j, the advertiser needs to specify the bid $x_{j,t} \in X_j$, where $X_j \subset \mathbb{R}^+$ is a finite set of bids we can set in subcampaign C_j. The goal is, for every day $t \in \{1, \dots, T\}$, to find the values of bids that maximize the overall cumulative expected revenue while keeping the overall ROI above a fixed value $\lambda^* \in \mathbb{R}^+$ and the overall budget below a daily value $y_t \in \mathbb{R}^+$. This setting is modeled as a constrained optimization problem at a day t, as follows:

$$\max_{x_{j,t} \in X_j} \sum_{j=1}^{N} v_j \, n_j(x_{j,t}) \tag{1a}$$

$$\text{s.t.} \frac{\sum_{j=1}^{N} v_j \, n_j(x_{j,t})}{\sum_{j=1}^{N} c_j(x_{j,t})} \geq \lambda^* \tag{1b}$$

$$\sum_{j=1}^{N} c_j(x_{j,t}) \leq y_t \tag{1c}$$

where $n_j(x_{j,t})$ and $c_j(x_{j,t})$ are the expected number of clicks and the expected cost given the bid $x_{j,t}$ for subcampaign C_j, respectively, and v_j is the value per click for subcampaign C_j. Moreover, Constraint(1b) is the ROI constraint, forcing the revenue to be at least λ^* times the incurred costs, and Constraint(1c) keeps the daily spend under a predefined overall budget y_t.
The available options consist in the different values of the bid $x_{j,t} \in X_j$ satisfying the combinatorial constraints of the optimization problem, while $n_j(\cdot)$ and $c_j(\cdot)$ are unknown functions, defined on the feasible region of the variables, that we need to estimate within the time horizon T.

2.1. Proposed Method

Deploying the solution described above in a real scenario requires its adaptation to the specific setting. A problem often encountered when deploying an AI system in a real-world scenario is that the distribution generating the data changes over time. To adapt the model to the environment changes, we use a sliding window approach. It consists in using only the data that has been collected recently to build the *costs* and *number of clicks* function estimates. Specifically, we consider only the observations generated in the last $w = 60$ days, as suggested by AdsHotel experts. As a matter of fact, they believe that it is rare for a concept drift to occur in intervals of less than two months.
AdsHotel requested our bid recommendations not to deviate too much from their current value so that they can evaluate them by hand and see whether or not they are reasonable and coherent. Thus, based on the performance p_j of sub-campaign C_j, we define an exploration percentage ϵ_j determining how much we can deviate from x_j.
Under the hardest periods of the COVID-19 pandemic, AdsHotel asked us to solve the problem in

Equations 1 with the highest possible ROI value $\bar{\lambda} \geq \lambda^*$. We address this problem through a binary search approach in the *DynamicROIOpt*(μ, λ^*) algorithm. It consists in running in a binary search fashion the optimization procedure, defined by Spadaro et al. [2], solving the problem in Equations 1 with a different λ value each time. Whenever a feasible solution is found with a given λ, we discard all ROI values lower or equal than λ. On the other hand, when no feasible solution is found by, we discard all ROI values higher or equal than λ.

2.2. Experimental Evaluation

The bids suggested by our system are not applied automatically to the various sub-campaigns. First, they are proposed to hotel managers through notifications in the AdsHotel platform. After that, it is up to them to employ such bids or not. Unfortunately, no hotel has been applying our suggestions consistently over time yet, and thus we cannot show and evaluate the performance of our algorithm. However, we can still show the bid values that our system would suggest for a real-world hotel campaign. We run the algorithm on a dataset composed of the observations generated in the last $w = 60$ days, specifically from 19 August 2020 to 19 October 2021. The *costs* and *revenue* function estimates of each sub-campaign are shown in Figure 1. Note that the *revenue* function estimate is obtained by multiplying the *number of clicks* by the value-per-click.
We show the results of the experiment in Table 2.

3. Context Aggregation

A campaign $\mathcal{C} = \{C_1, \ldots, C_N\}$ is composed by $N \in \mathbb{N}$ sub-campaigns. We define the value function $v\colon 2^{\mathcal{C}} \rightarrow \{0, 1\}$ that, given a subset of $\mathcal{C}$, returns either 0 or 1. It returns 1 if, by combining the data of all the sub-campaigns in the subset, the estimated *costs* and *number of clicks* functions are accurate enough. It returns 0 otherwise. We define a solution $S = \{A_1, \ldots, A_K\}$ as a partition of the set $\mathcal{C}$ s.t.: $\bigcup_{i=1}^{K} A_i = \mathcal{C} \wedge (A_i \cap A_j = \varnothing \; \forall 1 \leq i, j \leq K, i \neq j)$. The goal is to find the solution S^* such that:

$$S^* = \underset{S \in \mathcal{P}(\mathcal{C})}{\operatorname{argmax}} \sum_{A \in S} v(A), \tag{2}$$

where $\mathcal{P}(\mathcal{C})$ is the powerset of $\mathcal{C}$, and $\sum_{A \in S} v(A)$ is the cumulative value of the solution S.

Note that, if the *costs* and *number of clicks* functions were known, the solution S^* would be the trivial partition $S^* = \{A_1, \ldots, A_N\}$, where $A_i = \{C_i\} \; \forall i \in \{1, \ldots, N\}$. However, in our scenario, these functions are unknown and must be estimated. Therefore, combining together multiple sub-campaigns having poor data can actually be advantageous.

3.1. Proposed Method

First of all, it is important to say that, according to AdsHotel's experts, only the sub-campaigns targeting the same customer's country (user-country) can be aggregated together. The rationale behind this claim is that customers from different countries behave too differently from each other.
To address the problem described in Section 3, we designed the *DynamicCountryAggregation* algorithm which finds the best partition of a given campaign as described in Equation 2. Additionally, the algorithm must be able to divide the aggregations within that partition into two categories: those that pass the test and those that fail. In fact, AdsHotel wants us to provide a bid suggestion for a given sub-campaign only if we can build accurate function estimates for it.
Therefore, for each user-country of the given campaign, the *DynamicCountryAggregation* algorithm individually tests every sub-campaign through the value function v to see whether or not they have enough data on their own to accurately estimate the *costs* and *number of clicks* functions. A singleton aggregation is assigned to each sub-campaign passing this test. The ones failing it are aggregated together instead and then tested again. In the end, the optimal campaign partition is found, along with the categorization of its aggregations.
Unfortunately, due to the exploration constraints described in Subsection 2.1, imposed by AdsHotel in later stages of the work, the *DynamicCountryAggregation* algorithm cannot be used in our system. Indeed, if an aggregation is composed of multiple sub-campaigns, the defined intervals of allowed bids may not share elements in common.
For this reason, we designed the *SingletonAggregation* algorithm, which is the one actually implemented in our system. Given a campaign, the *SingletonAggregation* algorithm individually tests every sub-campaign and divides them according to the result into two categories: those passing the test and those failing. In this way, we can suggest a bid

only for sub-campaigns have enough data on their own to accurately estimate the *costs* and *number of clicks* functions. Note that, since we only allow singleton aggregations, the best partition is always the trivial one.
Now, we describe the tests we designed to check whether or not an aggregation has enough data to compute good estimates. The *Data-Driven* test consists of directly evaluating the data before actually performing any estimation. Specifically, given an aggregation, we check its observations to see whether or not there are at least two unique bid values, each one with at least seven non-zero *costs* and *number of clicks* samples. The idea is that, to have the most simple form of estimation of a function $f(x)$ (i.e., a line), we need at least two points, each related to a different x value. Moreover, we required to have a minimum of 7 non-zero samples corresponding to the same x, meaning that the bid has been tested at least for a week.

3.2. Experimental Evaluation

In this Subsection, we compare the results obtained by the *DynamicCountryAggregation* algorithm with those obtained by *SingletonAggregation*. We consider a dataset composed of observations generated in the last $w = 60$ days, specifically from 19 August 2020 to 19 October 2021. Of course, w is the sliding window presented in Subsection 2.1. Last but not least, the day we ran the experiments, 20 October 2021, there were 535 active campaigns with an overall number of sub-campaigns equal to $18,781$.
The results of the experiments are reported in Table 1. In the experiment run with the *SingletonAggregation* algorithm, 163 sub-campaigns passed the test, while $18,618$ did not. Thus, approximately 0.867% of the total sub-campaigns passed the test. On the other hand, the *DynamicCountryAggregation* algorithm got the following results: 297 sub-campaigns passed the test, while $18,484$ did not. Therefore, out of the total number of sub-campaigns, 1.58% passed the test. This quantity increased by 82.2% compared to the first experiment, showing that *DynamicCountryAggregation* would have been a much better alternative to *SingletonAggregation*.

	Singleton Aggregation	Dynamic Country Aggregation
Accepted	163	297
Rejected	18618	18484
Acceptance %	0.867%	1.58%

Table 1: Performance comparison between *SingletonAggregation* and *DynamicCountryAggregation* algorithms using the Data-Driven test.

4. Country Exploration

AdsHotel's platform does not allow us to expand campaigns by adding one sub-campaign at a time. It only allows us to add all sub-campaigns targeting a particular customer's country (user-country) at once. For this reason, we suggest the opening of a user-country rather than of a sub-campaign.
Country exploration consists into expanding a running campaign by opening promising user-countries that are not included in the campaign yet.

4.1. Problem Formulation

Given a finite time horizon of $T \in \mathbb{N}$ days, we define an advertising campaign $\mathcal{C}_t = \{A_t, B_t\}$ at day $t \in \{1, \dots, T\}$, where A_t and B_t are the set of its open and closed user-countries, respectively.
For each day $t \in \{1, \dots, T\}$, our goal is to specify the closed user-country $b \in B_{t-1}$ to be opened, such that the overall cumulative revenue of the expanded campaign $\mathcal{C}_t$ is maximized, while keeping the overall ROI above a fixed value $\lambda^* \in \mathbb{R}^+$.
At a given day t, the problem is modeled as follows:

$$\underset{b \in B_{t-1}}{\text{argmax}} \quad r(b) + \sum_{a \in A_{t-1}} r(a) \tag{3a}$$

$$\text{s.t.} \frac{r(b) + \sum_{a \in A_{t-1}} r(a)}{c(b) + \sum_{a \in A_{t-1}} c(a)} \geq \lambda^* \tag{3b}$$

where $r(a)$ and $c(a)$ are the expected revenue and the expected cost given the open user-country a, respectively. Moreover, Constraint (3b) is the ROI constraint, forcing the revenue to be at least λ^* times the incurred costs.

4.2. Proposed Method

As already outlined in Section 1, we propose two methods for opening a user-country: *Global Rankings* and *Performance Trend Monitoring*. The former is based on a ranking system, while the latter is based on change detection techniques.
The challenge here is again the lack of information. As a matter of fact, sub-campaigns belonging to

closed countries certainly do not generate samples, thus we cannot directly evaluate their performances to understand which one is the most promising. However, we can use the observations produced by other hotels in which those user-countries are open instead. More specifically, we use observations generated by hotels situated in the same country (hote-country) of the hotel for which we need to suggest the opening. However, this approach needs a strong assumption: all hotels located in the same hote-country have a similar influence over the customers.

For both methods, the performance measure is based on impressions (i.e., the number of times the ad is viewed by customers). The reason behind this decision is that, under COVID-19 restrictions, conversions are extremely rare, thus very unreliable. Clicks are not rare as conversions but they are still uncommon. Therefore, we narrow our attention to the impressions.

The *Global Rankings* algorithm first builds one global ranking for each existing hotel country. Each global ranking contains a descending ordering of user-countries in terms of total impressions received over the last w days by all hotels located in the given hote-country. Of course, w is the sliding window presented in Section 2.1. The impressions used to build the global rankings are normalized over the number of hotels generating those impressions. Furthermore, as suggested by the experts in this field, the ranking is weighted according to the continent of the user-countries. Then, given the set of hotels with at least one active campaign, the algorithm proposes, for each campaign of every hotel, the first closed user-country in the ranking.

To present the *Performance Trend Monitoring*, we first need to clarify what it is going to monitor. Given a pair *(hote-country, user-country)*, we monitor the time series of daily total impressions generated, over the last 6 months and for the given user-country, by all the hotels located in that hote-country. Also here, the monitored impressions are normalized over the number of hotels generating those impressions. Given the set of hotels with at least one active campaign, the *Performance Trend Monitoring* algorithm monitors, for each hotel, the performance trend of every user-country through a change detection procedure (i.e., CUSUM or ADWIN) and reports, for each campaign, the latest concept drift type of each closed user-country.

4.3. Experimental Evaluation

Unfortunately, we cannot evaluate the performance of any user-country that we suggested to open because our opening recommendations have not been applied by AdsHotel yet. Indeed, opening a user-country is a very delicate action to make, as AdsHotel's contract bindings with Hotels do not allow them to expand a campaign at their will. The Hotel Manager's authorization is needed to perform such action. For these reasons, the following experimental evaluations just show the global rankings built by our algorithm to suggest user-country openings.

Regarding the *Global Rankings* algorithm, we show in Figure 2 the three phases undergone by the Italian Global Ranking, built using the total impressions generated over the last w days by every Italian hotel on each possible user-country. More precisely, the rankings are built with data generated between 19 August 2021 and 19 October 2021. In the first phase, the global ranking is dictated by just the total impressions collected by each user-country. Then, in the second phase, each of them is normalized by the number of hotels contributing to the given user-country. Finally, the normalized total impressions are weighted according to the continent of the user-country. The latter is the global ranking that, given an Italian hotel, will be used to suggest the most promising closed user-country.

Regarding the *Performance Trend Monitoring*, we show in Figure 3 the results of the CUSUM and ADWIN tests on the same data stream, composed of the whole history of normalized daily total impressions, generated by Italian hotels on Italian customers. The whole data stream spans from 1 May 2020 to 19 October 2021. We performed the test on the whole history of data just for display purposes. As you can see, CUSUM seems to be more reactive to changes, detecting them a little bit earlier. Moreover, CUSUM has significantly lower computational complexity compared to ADWIN. For these reasons, we decided to adopt the CUSUM test as change detection test for our *Performance Trend Monitoring* algorithm. Unfortunately, we do not receive any feedback on our change detections. Thus, we have no way of evaluating the performance in a sound way. However, almost all the drifts detected by our algorithm can be explained by real-world environment changes, like COVID-19 restrictions and national holidays.

5. Conclusions

In this work, we applied state-of-the-art AI technologies in a real-world online advertising scenario for the company AdsHotel, which is responsible for the optimization of the advertising campaigns of multiple hotels all around the world.

We designed a real-world system built on top of the algorithm proposed by Spadaro et al. [2] to perform safe bid optimization for online hotel campaigns. The system we built is in charge of interfacing the before-mentioned algorithm with AdsHotel's real-world environment and implementing all the specific features required by such an environment. Specifically, we adopted a sliding window approach to deal with the non-stationary nature of the environment. Moreover, at the request of AdsHotel, we defined some policies, limiting our algorithm to suggest bid values not differing too much from the current ones. Then, we designed the *DynamicROIOpt* algorithm, which solves the bid optimization problem while satisfying the highest possible ROI value constraint, exploiting the optimization algorithm introduced by Spadaro et al. [2] in a binary search fashion. A very important block of our system is in charge of evaluating whether or not the observations generated by sub-campaigns are good enough to build accurate model estimates. To solve the task, we defined two different algorithms, namely *DynamicCountryAggregation* and *SingletonAggregation*. We compared their performance on real-world data, showing that *DynamicCountryAggregation* is the best alternative. At last, we defined the *Global Rankings* and *Performance Trend Monitoring* algorithms, providing suggestions on how to expand an ongoing campaign by selecting the most promising user-countries a hotel might open. Then, we display the results of both algorithms run on multiple real-world hotel advertising campaign data.

Our work opens up several interesting directions. First of all, a brand new CDT algorithm could be designed to actively detect concept drifts and adapt the model to the non-stationary environment. This task could be tackled by defining strategies to monitor the distributions of the GPs. Moreover, our system could be expanded, considering the Google sub-campaigns at their finest granularity during the optimization procedure. To do so, a new safe Gaussian Combinatorial Multi-Armed Bandit could be designed, having the various multipliers as arms instead of the bid values.

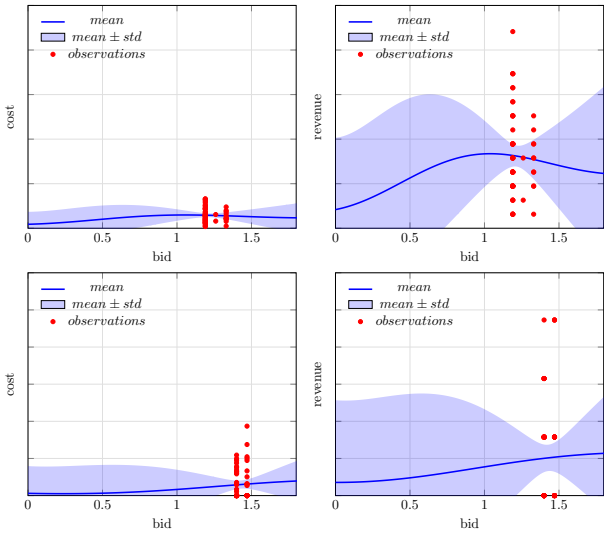

Figure 1: Cost and Revenue GPs of each sub-campaign in the given campaign. Due to NDAs with hotels, the actual cost and revenue values are not displayed. (top left) Cost GP of the (mexico, mobile) sub-campaign. (top right) Revenue GP of the (mexico, mobile) sub-campaign. (bottom left) Cost GP of the (united states, desktop) sub-campaign. (bottom right) Revenue GP of the (united states, desktop) sub-campaign.

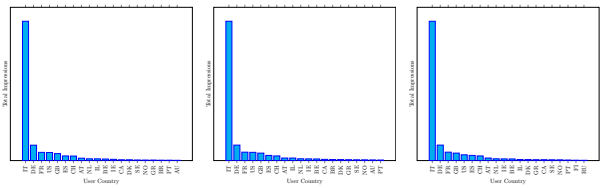

Figure 2: The three phases of Italian global ranking built with the total impressions of every user-country. Due to NDAs with hotels, the actual impression amounts are not displayed. (left) Not yet normalized total impressions. (center) Normalized total impressions. (right) Normalized and weighted total impressions.

References

[1] Alessandro Nuara, Francesco Trovo, Nicola Gatti, and Marcello Restelli. A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] Giorgio Spadaro. Online bid optimization with return-on-investment constraints. `http://hdl.handle.net/10589/170793`.

Table 2: Details of the experiment run with our algorithm on a real campaign. In the first column we identify the two sub-campaigns by specifying the country and the device of the targeted customer. *mx* stands for Mexico, while *us* stands for United States. Moreover, X_j is the initial set X_j of allowed bid values, x_j is the current bid value, p_j is the performance, ϵ_j is the exploration percentage, $\bar{X}_j$ identifies the new set of allowed bid values, $\hat{x}_j$ is the suggested bid, and finally, $\bar{\lambda}$ is the satisfied ROI value.

	X_j	x_j	p_j	ϵ_j	$\bar{X}_j$	$\hat{x}_j$	$\bar{\lambda}$
(mx, mobile)	$[0.00, 2.21]$	1.19	53.6%	5%	$[1.13, 1.25]$	1.13	4.01
(us, desktop)	$[0.00, 2.21]$	1.47	7%	14.5%	$[1.25, 1.69]$	1.69	

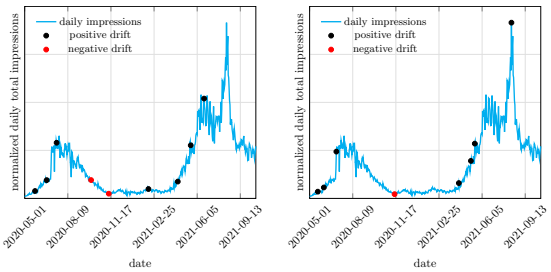

Figure 3: Comparing the performance of CUSUM and ADWIN tests, both executed on the whole history of normalized daily total impressions generated by Italian hotels on Italian customers. Due to NDAs with hotels, the actual impression amounts are not displayed. (left) CUSUM test. (right) ADWIN test.

[3] Weinan Zhang, Ying Zhang, Bin Gao, Yong Yu, Xiaojie Yuan, and Tie-Yan Liu. Joint optimization of bid and budget allocation in sponsored search. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1177–1185, 2012.

POLITECNICO DI MILANO
Dipartimento di Elettronica e Informazione
Master of Science in Computer Science and Engineering

Online Advertising Campaign Management for Hotel Booking

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Advisor: Prof. Francesco Trovò
Co-advisor: Ph.D. Alessandro Nuara
Co-advisor: Dott. Giulia Romano

Master Graduation Thesis by:
Luca Alessandrelli, matricola 928231

Academic Year 2020-2021

Abstract

The Online Advertising revenue in the United States increased from 126.4 billion in 2019 to 139.8 billion U.S. dollars in 2020. A further sign of its growth is that the worldwide spending in 2020 has been estimated to be 378 billion U.S. dollars and is forecast to increase in the following years, reaching 646 billion by 2024. All the opportunities that Online Advertising has brought for the advertising market have drawn a lot of attention from the scientific community. In particular, the *Artificial Intelligence* (AI) field played a crucial role in providing automatic mechanisms to support both the publishers and advertisers in their tasks. We specifically focus our attention on the advertiser's task, consisting in optimizing an online advertising campaign.

The goal of this work is to apply state-of-the-art AI technologies in a real-world Online Advertising scenario for the company AdsHotel, which is responsible for the optimization of the advertising campaigns of multiple hotels all around the world. We built a real-world system exploiting existing AI's methods to perform safe bid optimization (i.e., guaranteeing a given return-on-investment (ROI) constraint) and extending them to adapt their behavior as the hotel environment changes. This is crucial due to the many reasons why it might present non-stationarity over time (e.g., seasonality, national holidays, and unpredictable events like COVID-19). Finally, we integrate into the system a strategy to provide suggestions on how to expand an ongoing campaign by selecting the most promising sub-campaigns a hotel might open. By implementing our system in the AdsHotel environment, we evaluate its capabilities and measure its performance over multiple real-world hotel advertising campaigns.

Sommario

Il mercato pubblicitario online statunitense ha registrato entrate pari a 126.4 miliardi di dollari a fine 2019, per poi raggiungerne 139.8 l'anno seguente. Inoltre, il valore di mercato mondiale è attualmente pari a 378 miliardi di dollari ed si stima che possa raggiungere i 646 miliardi nel 2024, a ulteriore prova della costante crescita di questo settore. Nel corso dell'ultimo decennio, le numerose opportunità introdotte da tale mercato hanno alimentato l'interesse della comunità scientifica. In particolare, il campo della Intelligenza Artificiale (IA) ha avuto un ruolo fondamentale nel fornire meccanismi automatici a supporto sia degli inserzionisti, sia di coloro che forniscono spazi per annunci online. In quanto segue, il nostro focus è rivolto agli inserzionisti, i quali hanno il compito di ottimizzare le proprie campagne pubblicitarie online.

L'obiettivo di questo lavoro consiste nell'applicare tecnologie appartenenti allo stato dell'arte della IA per dare supporto all'azienda AdsHotel, la quale si occupa della gestione di campagne pubblicitarie online per molteplici hotel in tutto il mondo. Sfruttando alcune delle metodologie esistenti nel campo dell'IA, abbiamo realizzato un sistema in grado di eseguire una ottimizzazione sicura delle bid, garantendo quindi di superare una determinata soglia minima di ritorno sugli investimenti (ROI) all'azienda. Inoltre, abbiamo esteso tali metodi per adattare il sistema ai cambiamenti dell'ambiente a esso circostante; questo è infatti di estrema importanza date le svariate ragioni per cui esso può risultare non stazionario nel tempo, tra cui: stagionalità, festività nazionali ed eventi imprevedibili come la diffusione del COVID-19. In aggiunta, abbiamo integrato in tale sistema una strategia per espandere le attuali campagne pubblicitarie. Nello specifico, essa consiste nel suggerire a un hotel l'apertura delle sotto-campagne attualmente non attive che però risultano essere le più promettenti. Infine, implementando il nostro sistema all'interno dell'ambiente di AdsHotel, ne valutiamo le capacità e le prestazioni su reali campagne pubblicitarie relative ad hotel.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Advertising is all about promoting the sale of a product or service to customers. It has been around since ancient civilizations and has evolved in various ways through time. In the nineties, a new form of advertising called *Online Advertising* (OA) emerged, exploiting the Internet as its marketing channel. Since then, this Industry has been continuously growing year by year to the point in which it took over the TV Advertisement market in 2016. The Online Advertising revenue in the United States increased from 126.4 billion in 2019 to 139.8 billion U.S. dollars in 2020 [14]. The yearly revenue increase in U.S. has been on average 18.65% over the last 5 years. A further sign of its growth is that the worldwide spending in 2020 has been estimated to be 378 billion U.S. dollars and is forecast to increase in the next years reaching 646 billions by 2024 [13]. All the opportunities that OA has brought for the advertising market in recent years have drawn a lot of attention from the scientific community. In particular, the *Artificial Intelligence* (AI) field played a crucial role in providing automatic mechanisms to support both the publishers and advertisers in their tasks.

1.1 Problem description

In this work, we focus our attention on the advertiser's task, which consists in optimizing an online advertising campaign. Performing such optimization is very complex and it involves several minor sub-problems. First of all, a campaign is composed of multiple sub-campaigns, each one designed to target a specific type of audience. Indeed, due to the number of information people disclose when navigating the internet, it is possible to match a visiting customer to a specific audience type. Thus, the first task of the advertiser is to identify the typology of customers that are more likely to be interested in their product or service. After that, the advertiser should pick the most suitable advertising channel (e.g., display, search, social) for their product or service. The next step is to assign a sub-campaign, along with a specifically tailored ad, to each audience type. The goal of every sub-campaign is to gain the highest possible customer engagement to maximize the total advertiser revenue.

The creative part of the task ends here, leaving space for the technical sub-

problems that are going to be tackled by AI methods. To get an ad shown to customers, the advertisers must take part in an auction carried out by the publisher. Here, the advertiser places a bid and a daily budget to win the highest visibility spot for each of her sub-campaigns. More specifically, a bid is the maximum amount of money the advertiser is willing to pay for a performed action on her ad, where the action depends on the adopted payment scheme (e.g., pay-per-click), and the daily budget is the maximum amount of money that can be spent in a day for the given sub-campaign. This means that, given a campaign, the advertiser must decide each day the bid and daily budget values for each sub-campaign. In the scientific literature this problem is called *joint bid-budget optimization* [38]. However, in some online advertising scenarios, it is also very important to satisfy return-on-investment (ROI) constraints during the campaign optimization, thus introducing a further degree of complexity to the problem. Moreover, there are also other crucial aspects to include in the optimization process, e.g., seasonality, unpredictable changes in the market economy, and competition with other advertisers.

If we consider that an advertiser usually takes care of multiple campaigns, each made by several sub-campaigns, the number of variables that need to be set during the optimization process becomes incredibly large. For this reason, the task is almost impossible to be carried out by humans and an automatic mechanism is needed to deal with its massive complexity. This is where the Artificial Intelligence field comes into help.

1.2 Goal

This work aims at applying state-of-the-art AI technologies in a real-world Online Advertising scenario for the company AdsHotel [2], which is responsible for the optimization of the advertising campaigns of a multitude of hotels all around the world. The main goal is to develop an AI method that obtains good performances while still satisfying the several constraints imposed by the hotel industry. However, we are also interested in understanding the eventual limitations of modern technologies regarding this specific setting, and ultimately propose interesting future directions that could lead to performance improvements.

1.3 Contribution

As already outlined, the contribution of this work consists of applying state-of-the-art AI technologies to provide AdsHotel with a tool capable of performing automatic online campaign optimization for hotels.

In doing so, we had to deal with some constraints imposed by AdsHotel's contract bindings. Indeed, the hotel industry is constantly seeking immediate results. This translates into minimum ROI constraints that must be satisfied during the whole advertising process. However, the vast majority of state-of-the-art methods for campaign optimization focus on maximizing the revenue while satisfying budget constraints only. Eventually, we narrowed our attention to the

few works providing safe bid optimization methods. Specifically, we adopted the solution designed by Spadaro et al. [45] in which the ROI constraints are satisfied with high probability during the campaign optimization routine. Thus, one of our contributions consists of making some minor tweaks to their algorithm to adapt it to our specific scenario, wrapping it around a system that is in charge of interfacing with the already existing AdsHotel's Platform, and, finally, performing safe bid optimization with return-on-investment constraints.

Moreover, a particular aspect of our scenario is that the environment changes over time due to multiple reasons like seasonality, festivities, unpredictable events (e.g., COVID-19), and many others. Thus, we are interested in techniques that do not just learn the model, but they also need to adapt it to the changing environment. Indeed, it is critical to take into account the non-stationarity of the environment to maintain good performances over time.

Another problem we had to deal with was the scarcity of data due to travel restrictions imposed for the COVID-19 pandemic [54, 56]. Indeed, a fundamental preliminary step for performing campaign optimization in our setting consists of estimating a model for each sub-campaign. Moreover, the performance of the campaign highly depends on the accuracy of these estimations as they guide the bid optimization phase. Thus, it is crucial to have accurate models and, for that to happen, not only do we need clean data but we need plenty of them as well. To overcome the data scarcity problem, we had to define context aggregation strategies to group up the little data available in each sub-campaign. By doing so, we managed to feed our model with enough generalized, but still meaningful, information to be able to build good models for a higher number of sub-campaigns. A further contribution regards the possibility of expanding an ongoing campaign by adding promising sub-campaigns that are not included in the campaign yet. Given a hotel, we estimate the performance of its closed sub-campaigns exploiting the data generated by similar hotels. Through Online Monitoring techniques we are then able to understand which sub-campaigns are promising and, if so, suggest their opening to the hotel manager.

1.4 Document Outline

The document is structured as follows. Chapter 2 introduces the relevant background theory and notions frequently used in the document about *Hotel Online Advertising* and then describes some crucial aspects of AdsHotel's Platform. Chapter 3 presents the relevant background theory and notions frequently used in the document about *Online Learning and Monitoring*. Chapter 4 discusses state-of-the-art methods that have been proven useful to achieve the goals of this thesis. Chapters 5, 6, and 7 are dedicated to the three contributions of this work, respectively: safe bid optimization with return-on-investment constraints, context aggregation, and country exploration. A problem formulation, a detailed description of our solution approaches, and experimental results can be found in each of these chapters. Finally, Chapter 8 draws conclusions on this work and opens new directions for further projects.

Chapter 2

Hotel Advertising Background

This Chapter provides necessary concepts and notations to let the reader understand and familiarize with the topics covered in the rest of the document. Section 2.1 further explores the Online Advertising concepts presented in the Introduction. Section 2.2 describes the Hotel Online Advertising setting, the differences with respect to classical OA, and, finally, the main hotel advertising channels. Finally, section 2.3 discusses some details of AdsHotel's Platform that have been key factors for our algorithm design during the project.

2.1 Online Advertising

In Section 1, we introduced Online Advertising by showing the growth of its economy. Indeed, OA has become the first choice when it comes to advertise products or services online. This is due to the fact that it comes with a lot of advantages. First of all, compared to other form of advertising, it is known to cover a massive amount of audience. Indeed, nowadays most people navigate the Internet daily, be it with desktop or mobile devices, leading to high exposure to online ads. If reaching to a huge audience is important, having the possibility to display tailored advertisements based on the characteristics of the visiting customers is even more important. The targeting possibilities are nearly endless over the Internet due to the amount of information people disclose, such as economic status, age, personality traits, lifestyle, and hobbies. A crucial advantage is that it is very easy to track the performance of an online campaign. In this way, advertisers can easily understand which sub-campaign is profitable and which is not. Moreover, advertising on the Internet is also cost-effective: there is no need to massively invest on a campaign, as opposed to what happens in other forms of advertising. Depending on the payment scheme, you only pay when a precise action is performed by the user, e.g., when the customer clicks on the ad. Moreover, most publishers allow the advertisers to modify their bid and budget as the campaign is running. Thus, an advertiser has a complete control over the amount of money invested in the campaign.

Let we introduce the different roles in an Online Advertising scenario. Typically, three figures are involved:

- **Advertisers** aim at selling their product or service to customers. To do that, they perform multiple tasks. First, they identify the different types of audience that might be interested to their product or service. They provide tailored ads for each audience group, defining in this way the various sub-campaigns. Then, they take part to the publisher's auction to win the best visibility spot. In doing so, they need to set a bid and a daily budget value for each sub-campaign. The way in which those values are assigned (i.e., joint bid-budget optimization) depends on the specific goal of the advertiser. Some typical goals are: (1) to maximize the revenue over the costs (i.e., maximizing the ROI), (2) to maximize the sales (i.e., the amount of product or services sold), or finally (3) to maximize sales under ROI constraints;

- **Customers** are people navigating the Internet that are exposed to the advertisers' ads and that can potentially buy their products or services;

- **Publishers** are the middle-man. They make money by showing the advertisers' ads to customers visiting their online content or websites. Their role is to maximize their revenue, thus they will show the ads that are going to pay more on average [20]. To do so, publishers run auctions in which the advertisers compete for the highest visibility.

Due to the numerous tasks advertisers must perform, nowadays it is common to see a fourth figure: **media agencies**. They are commissioned by advertisers to fulfill part of their tasks, like running the joint bid-budget optimization for their campaigns.

2.1.1 Performance Indices, Formats, and Payment Schemes

Online Advertising has four main **performance indices**:

1. *impressions* represent the number of times the ad has been viewed by customers;

2. *clicks* represents the number of times customers clicked on the ad to visit the advertiser's website;

3. *leads* show the number of potential sales (i.e., registration form filled by the user);

4. *conversions* express the number of sales for the advertised product or service.

These indices clearly recall the funnel model defined in [47] and displayed in Figure 2.1. The idea behind the model is that the advertiser should guide the customers through all the following layers: *awareness* is the first, *interest* is the second, *decision* is the third and at last there is the *action* layer. The customer starts from the first layer and can either move towards the next one or leave. This means that only a small portion of the customers will actually reach the

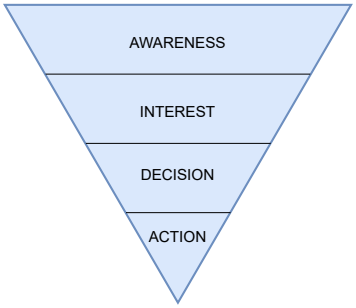

Figure 2.1: Marketing funnel model.

action layer. The main goal of an advertiser, or media agencies, is to maximize the amount of customers reaching the last layer (i.e., maximize conversions). However, every other layer plays an crucial role in reaching the final task.

Formats specify the channel over which the advertisement occurs. They differ according to the available users' information and the layer of the funnel model in which they operate. We can distinguish three main formats:

1. *Search.* The customer types a query on a search engine (e.g., Google) utilizing some keywords. Consequently, the publisher generates a web page containing the organic content along with advertisements. This allows the advertiser to target their ads to customers who are already willing to purchase through their search query. Therefore, there is a high chance to lead the user to generate a conversion. For this reason, search advertising stands at the last layer of the funnel model;

2. *Social.* The ads get displayed to the customer on social network pages or apps. The advertisement messages include posts, videos, and banners. This channel provides the advertisers with specific information about the user's interests and behaviors, allowing them to target their ads to very specific audiences. For this reason, social advertising stands at the mid-levels of the funnel model;

3. *Display.* The ads get displayed to the customer on web pages as banners, images, videos, etc. The main goal of the advertiser is to increase the awareness of their brand as much as possible. For this reason, display advertising usually stands at the first level of the funnel model.

In OA there are three commonly used **payment schemes**, each specifying when the advertiser should pay the publisher:

1. *Pay-per-impression.* The advertiser pays each time the ad registers a new impression. It is commonly used in social and display advertising, where the goal is to accumulate impressions;

2. *Pay-per-click.* The advertiser pays each time a customer clicks on the ad. This payment scheme is suited for search and social advertising, where the main goal is to accumulate clicks;

3. *Pay-per-conversion.* The advertiser pays every time a user generates a conversion through one of their ads. This scheme is suited for search advertising, where the goal is to accumulate conversions.

Relations exist between performance indices, formats, and payment schemes. Indeed, not all the performances indices are used for all the formats, and the same holds for the payment schemes. For instance, search advertising is mainly run on a pay-per-click scheme. In what follows, we will assume that the underlying format and payment scheme are search advertising and pay-per-click, respectively, since our project with is placed mainly in this setting.

2.1.2 Advertising Campaign

An advertising campaign is usually run on a single channel (e.g., search) to advertise a specific product or service. The campaign is composed of multiple sub-campaigns, each one specifying:

1. the target audience, i.e., geographic area, the age, the interests, customer information;

2. the *bid*, i.e., the maximum amount of money that the advertiser is willing to pay for each single impression, click or conversion, depending on the payment scheme;

3. the *daily budget* which specifies the total amount of money the advertiser is comfortable spending each day for the given sub-campaign.

Usually, a campaign also has a total daily budget constraint, that is equal to the sum of the daily budget for each sub-campaign.

Depending on the specific advertising channels, different user information are available. Unfortunately, the available information in search advertising is very little because it is directly derived from the customer's query. Usually, it includes the customer's IP address, the device she/he is using, its language and other information that is application dependent.

The campaign optimization process is a fundamental task for the advertiser. Indeed, a proper setting of the bid values for each sub-campaign is crucial to balance the trade-off between high volumes (i.e., sales) and high profitability (i.e., ROI). More specifically, a high bid will lead into winning most auctions, increasing the volume, while a low bid will lead to higher return over the costs (i.e., ROI). On the other hand, a proper assignment of the daily budget values is crucial to satisfy the budget constraints of the campaign.

2.1.3 Publisher's Optimization Problem

In this section, we focus on the optimization problem from the publisher's point of view. Her/His role is to produce the web page containing the organic content, queried by the customer, along with the advertisers' ads. We show an example of such web page in Figure 2.2. The publisher can assign ads to several positions divided in slates. Every slate is further split in one or more slots in which a single

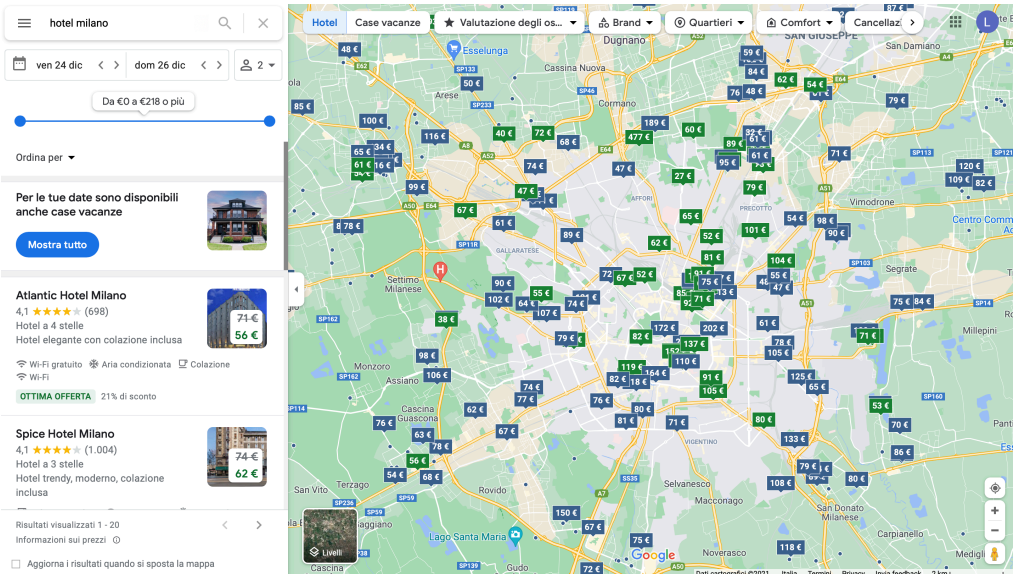

Figure 2.2: Example of a web page created by the publisher Google Hotel Ads containing multipl advertiser's ads.

ad is displayed. In some cases, a single ad is allowed to appear over multiple slots, however we do not consider this case in our work. The goal of the publisher is to find the ad allocation over the available slots that maximizes his revenue.

The formalization of the problem is centered around the estimated value of allocating an ad to a slot. In particular, the advertiser's expected value of displaying ad a on slot s is: $\Lambda_S q_a v_a$, where Λ_S is the probability that the customer observes the slot S (i.e., slot prominence), q_a is the click probability on ad a given that it has been observed by the user (i.e., ad quality), and v_a is the advertiser's value-per-click (vpc) (i.e. ad value) for ad a. $\Lambda_S q_a$ is commonly called click through rate (CTR) and it is the click probability of ad a placed on slot Λ_S. The value provided by an allocation is given by:

$$\sum_a \Lambda_{S(a)} q_a v_a,$$

where $S(a)$ is the slot in which ad a is allocated. If an ad a is not allocated, then we have $\Lambda_{S(a)} = 0$.

The idea behind the definition of slot prominence is that costumers follow a cascade model, i.e., they start observing the first slot, then with a given probability they move to the second one, and so on. Thus, if there are k slots in total, they can be sequentially ordered from 0 to k. Moreover, if the customer

is viewing slot S_i her/his probability of observing also slot S_{i+1} is given by:

$$\frac{\Lambda_{S_{i+1}}}{\Lambda_{S_i}},$$

where $i \in \{1, \ldots, k\}$. Finally, the solution for the optimization problem (i.e., the best allocation) is:

$$\underset{a}{\operatorname{argmax}} \sum_{a} \Lambda_{S(a)} q_a v_a.$$

The optimization procedure of the allocation turns out to be computationally easy: first, sort the the ads in decreasing order of $q_a v_a$, then allocate ads to slots according to such order. The computational complexity is linear in the number of ads n, and logarithmic in the number of slots k, i.e., $O(n \log k)$.

The values of the parameters used in the problem formulation are crucial to find the best allocation. However, the publisher usually does not know them, and, thus, needs to estimate them. In particular:

- Λ_S and q_a are estimated by the publisher by using data coming from all the allocated ads;
- v_a is a private information of the advertiser which is sent to the publisher in the form of a bid. Note that, in principle, the advertiser may communicate a false value to get advantages. Moreover, a large number of samples is needed to estimate v_a, thus most of the times the advertiser cannot perform an accurate estimation.

2.1.4 Auction Mechanism

Whenever the publisher generates a web page it runs an auction, which is very important as it defines the ads allocation and the pay-per-click payments of the advertisers. For simplicity we assume that each ad comes from a different advertiser. Each time an auction is run, advertiser reports a bid that represents the maximum amount of money they will pay for a click. Given the bids, the auctioneer chooses the allocation and the payments. The allocation is performed as explained in Section 2.1.3, while the payments are typically decided through a Generalized Second Price auction (GSP) [16]. Formally, the payment p for ad a is given by:

$$p_a = \frac{q_{a+1}}{q_a} v_{a+1} \left(\leq \frac{q_a}{q_a} v_a = v_a \right), \tag{2.1}$$

where the ads are sorted in decreasing order by $v_a q_a$. Since the publisher usually does not know the values v_a the bid communicated by the advertiser will be used instead. However, such a mechanism does not enforce a true-telling optimal strategy for the advertiser, thus bidding the true value may not be the optimal strategy for every advertiser taking part to the auction.

Another commonly used auction is called the Vickrey-Clarke-Groves auction (VCG) [32]. The payment p for ad a is given by:

$$p_a = \frac{1}{\Lambda_{S(a)} q_a} \left(X_a - Y_a \right), \tag{2.2}$$

$$X_a = \max_{S(a')} \sum_{a' \neq a} \Lambda_{S(a')} q_{a'} v_{a'}, \tag{2.3}$$

$$Y_a = \sum_{a' \neq a} \Lambda_{\bar{S}(a')} q_{a'} v_{a'}, \tag{2.4}$$

$$\bar{S}\left(a'\right) = \operatorname*{argmax}_{S(a')} \sum_{a'} \Lambda_{S(a')} q_{a'} v_{a'}. \tag{2.5}$$

Recall that unlike GSP, the Vickrey-Clarke-Groves auction admits only an optimal true-telling strategy for the advertisers.

In practice, publishers run multiple auctions throughout the day, either GSP, or VCG. Every advertiser has a daily budget and takes part in the auction if her remaining daily budget is not expired. During the day, the publisher may allow the advertisers to change their bid, however the delay with which this modification is applied is not known. On the other hand, the daily budget can be set only once per day. As a final remark, we recall that VCG is truthful, but only if run once. In a repeated scenario where we have budget constraints, VCG is not truthful anymore. Moreover, GSP allows non-truthful optimal strategies even in its single auction formulation. This implies that advertisers need systems to determine the best bid to be communicated to the platform. This is the reason why an automatic method able to learn the optimal bid is crucial to optimize an online advertising campaign.

2.1.5 Advertiser's Optimization Problem

In this section, we focus on the optimization problem from the advertiser's point of view. At day t, given the daily budget of the campaign, we need to find the best bid and daily budget values for each of its sub-campaigns.

Before we formalize the problem we need to make two assumptions. First, we assume that the performance of every sub-campaign is independent from each others. For details to include campaign dependencies, please refer to [35]. Second, the bid and daily budget values are finite and constrained to some given intervals. This is not true in general but, in practice, it is reasonable since experts of the domain usually fixed these interval values before the start of the campaign.

In what follows we used the formalization of the optimization problem provided by [37], which is an extended version of the Knapsack Problem. Formally:

$$\max_{x_{j,t}, y_{j,t}} \sum_{j=1}^{N} v_j n_j \left(x_{j,t}, y_{j,t}\right) \tag{2.6a}$$

$$\text{s.t.} \sum_{j=1}^{N} y_{j,t} \leq \bar{y}_t \tag{2.6b}$$

$$\underline{x}_{j,t} \leq x_{j,t} \leq \bar{x}_{j,t} \qquad \forall j \tag{2.6c}$$

$$\underline{y}_{j,t} \leq y_{j,t} \leq \bar{y}_{j,t} \qquad \forall j \tag{2.6d}$$

where:

- j denotes the index of a sub-campaign;
- N is the number of sub-campaigns of our problem;
- $x_{j,t}$ is the bid of sub-campaign j at time t, usually expressed in days;
- $y_{j,t}$ is the daily budget of sub-campaign j at day t;
- v_j is the estimated value per click of sub-campaign j;
- n_j is the expected number of clicks of sub-campaign j, given the budget and the bid of of sub-campaign j;
- $\bar{y}_t$ is the daily budget for the campaign at day t;
- $\underline{x}_{j,t}$ and $\bar{x}_{j,t}$ are the lower bound and upper bound on the bid for sub-campaign j at day t;
- $\underline{y}_{j,t}$ and $\bar{y}_{j,t}$ are the lower bound and upper bound on the daily budget for sub-campaign j at day t.

The objective function stated in Equation (2.6a) is the sum of the expected revenue generated by each sub-campaign j. The constraint in Equation (2.6b) is a budget constraint, forcing the sum of sub-campaigns' daily budgets not to exceed the daily campaign budget. Instead, constraints in Equations (2.6c) and (2.6d) force the bid and daily budget variables to assume values in their respective ranges. In the literature, this problem has been tackled in various ways. Some works [60, 37] address bid and budget optimization at the same time (i.e., joint bid-budget optimization), while others [15, 58, 50, 45] address them separately. However, these methods cannot be applied directly to the setting of Hotel Online Advertising. In what follows, we will describe such a setting and highlight why we need a more refined method specifically crafted for this application.

2.2 Hotel Online Advertising

In Chapter 1, we outlined the specific setting of our work: Online Advertising for Hotels. Here, the services to be advertised are hotel rooms. Even if everything we explained for the general OA setting is still valid for the Hotel industry, some characteristics are present only in this specific setting. For instance, a peculiar aspect of this business is that advertisers are *performance advertisers*: they mainly focus on short-term returns. Indeed, the goal of an hotel manager is having their rooms fully booked. The only exception to this behavior may be that of Hotel Chains, who may care more about long-term visibility. In the literature, it has been observed that, in a display advertising scenario with performance advertisers and second price auctions, a consistent slice of the advertisers behave as they had minimum ROI constraints [21]. Even though our work does not take place in display advertising, the previous observation turned out to be true in our scenario as well. Indeed, AdsHotel specifically demanded our optimization algorithm to guarantee a minimum ROI value. This means that the goal of the optimization is to achieve a good trade-off between high volumes (i.e., conversions) and high profitability (i.e., ROI). Moreover, these ROI constraints should be satisfied during the entire advertising period. Another interesting aspect, that proved to be true during our experience, is that most advertisers prefer to adopt safe bidding strategies that satisfy the ROI constraints even during the initial phase of the learning process, whose nature is almost purely exploratory.

2.2.1 Google Hotel Ads

One of the advertising channels used in our work is Google Hotel Ads [24]. Its role is to be the publisher and it does so in the following way: a customer searches for some hotel similar to the ones advertised in the campaign and our ad has a chance of being displayed to her. The search can be performed both in Google Search and Google Maps.

In Google Hotel Ads a campaign is defined by [22]:

- The campaign start and end dates, along with the total budget;
- The preferred payment scheme;
- The countries of the targeted customer, in this way the ads are shown only in specific user countries;
- The customer's devices over which ads will be shown. The possible devices are: desktop, mobile, and tablet.

The payment scheme of choice is defined at campaign level and can be one among the followings [23]:

- **Manual cost-per-click (CPC)**: pay a fixed amount of money each time a customer clicks on the ad;
- **% CPC**: pay a percentage of the hotel's room booking price each time a customer clicks on the ad;

- **Optimized CPC**: the pay-per-click payment may vary because the bid is automatically adjusted to maximize the number of conversions;
- **Commissions (pay-per-conversion)**: pay a fixed percentage of the hotel's room booking price each time a customer performs a booking;
- **Commissions (pay-per-stay)**: pay a fixed percentage of the hotel's room booking price each time a customer reaches to the hotel. This removes cancellation costs.

Bids are defined at sub-campaign level, according to the chosen payment scheme. Up until now, one could figure out that a sub-campaign is identified by the tuple *(hotel, user-country, device)*. However, Google's sub-campaigns go far beyond that targeting level. More specifically, there are three more booking parameters that one might take into account:

- **Length of Stay (los)**: it indicates the number of days for which the customer would like to book the hotel room;
- **Check-In Day**: it is the hypothetical booking check-in day;
- **Booking Window Days**: it indicates the number of days between the hypothetical booking and the check-in.

To summarize, a Google sub-campaign is identified by the tuple *(hotel, user-country, device, los, check-in day, booking window days)*. Thus, a different bid can be set for each combination of those parameters.

2.2.2 Tripadvisor

The other publisher targeted by the thesis project is Tripadvisor [49]. Running an advertisement campaign in this setting is similar to what has been described for Google Hotel Ads. Indeed, a customer visiting Tripadvisor's website searches for a travel destination or directly the hotel's name. Then, a list of hotel ads will be shown to the user. Tripadvisor's campaigns are defined in the same way they are defined in Google Hotel Ads. Also the payment scheme and total budget are still defined at campaign level. The available payment schemes are *pay-per-impression* and *pay-per-click*. Bids are defined at sub-campaign level, according to the chosen payment scheme.

Differently from what has been presented for Google Hotel Ads, Tripadvisor does not allow the advertiser to specify some of the booking parameters. Indeed, a Tripadvisor sub-campaign is identified by the tuple *(hotel, user-country, device)*, and a different bid can be set for each combination of those parameters.

Notice that the system we are going to describe in what follows can be applied to any metasearch, as long as it allows to partition the advertising campaign as described above and that the data corresponding to day-by-day bid, impression, click, and impressions are available.

Finally, we remark that the optimization of the above described problem is not trivially solved by dynamic programming techniques since some of the

elements in it are unknown to the advertiser, e.g., the number of clicks given a bid $n_j(x_{j,t}, y_{j,t})$ for each sub-campaign. This requires the design of specifically crafted algorithms able to learn such quantities while minimizing the loss due to the learning process. This thesis builds on existing online learning techniques to this setting and adapt them w.r.t. the additional constraints required by the Hotel Advertising setting.

2.3 AdsHotel's Platform

To fully understand the remainder of the document, we present some aspects regarding the Platform of AdsHotel. Specifically, we describe the way in which campaigns have been set up in both advertising channels (i.e., GoogleAds and Tripadvisor), the way in which bid values are assigned in the Platform, and, finally, the Dataset provided by AdsHotel.

2.3.1 Google Campaigns in AdsHotel's Platform

Following the structure described in Section 2.2.1, every Google campaign is defined by AdsHotel in the following way:

- The start and end dates, along with the total budget are decided by the specific hotel;
- The payment scheme is *% CPC*: pay a percentage of the hotel's room booking price each time a customer clicks on the ad;
- The countries of the targeted customer can be modified during the campaign by AdsHotel. From now on those countries will be called *user-countries*;
- All devices are always included (i.e., desktop, mobile, and tablet).

We know that a Google sub-campaign is identified by the tuple *(hotel, user-country, device, los, check-in day, booking window days)*. This means that, the AdsHotel's Platform should allow a bid to be defined for each possible combination of those parameters. However, AdsHotel decided to go through a different route: those bids are indirectly defined by a set of multipliers. First of all, AdsHotel allows to specify a bid at user-country level called *Base Bid*. The so-called *multipliers* are used to enable bidding at a finer granularity. There are four multiplier archetypes: (1) *Device Type*, (2) *Length of Stay*, (3) *Check-In Day*, and (4) *Booking Window Days*. Each of these archetypes has several predefined sub-levels. For instance, Device Type has three of them: desktop, mobile and tablet. Check-In Day trivially has seven sub-levels (i.e., the week days). Booking Window Days has several predefined sub-levels, e.g., $\leq$ than 15 days, $\leq$ than 30 days. Finally, Length of Stay has predefined sub-levels like $\geq$ 1 day, $\geq$ 2 days, $\geq$ 3 days. Each sub-level is a multiplier and its value can be assigned on the Platform. They are in the form of percentages and can be either positive or negative (e.g., $+25\%$ or -25%). Note that, the sub-levels are constructed in

such a way that, for each archetype, only one multiplier is going to be applied for a specific display of the ad. Then, given a specific sub-campaign, the four multipliers of interest are taken into account along with the Base Bid to compute the final bid, that will be used in the auction. Formally, given sub-campaign C_j, the set $M = \{m_{devicetype}, m_{los}, m_{daysofweek}, m_{advancebooking}\}$ of the four multipliers corresponding to C_j and the Base Bid b_j, we compute the final bid x_j in the following way:

$$x_j = b_j \prod_{m \in M} \left(1 + \frac{m}{100}\right). \tag{2.7}$$

Note that, in this scenario, the Base Bid is a percentage and, thus, the final bid will be a percentage as well, in line with the *% CPC* payment scheme.

Regarding the opening and closing of sub-campaigns, AdsHotel's Platform allows such actions only at user-country level. This means that, when we open a country, all sub-campaigns associated to that user-country will open as well. On the other hand, when we close a country, all its associated sub-campaigns will close too.

2.3.2 Tripadvisor Campaigns in AdsHotel's Platform

Following the structure described in Section 2.2.2, every Tripadvisor campaign is defined by AdsHotel in the following way:

- The start and end dates, along with the total budget are decided by the specific hotel;
- The payment scheme is *Manual CPC*: pay a fixed amount of money each time a customer clicks on the ad;
- The user-countries can be modified during the campaign by AdsHotel;
- All devices are always included (i.e., desktop, mobile, and tablet).

We know that a Tripadvisor sub-campaign is identified by the tuple *(hotel, user-country, device)*. This means that, AdsHotel's Platform should allow a bid to be defined for each possible combination of those parameters. However, as already explained in details in Section 2.3.1, bids are indirectly defined by a set of multipliers. Fortunately, that is much easier for Tripadvisor. Indeed, we only have the Base Bid and one archetype, namely the Device Type. Its sub-levels are three, i.e., desktop, mobile, and tablet. This means that, given a specific sub-campaign C_j, we retrieve the associated Base Bid b_j and the device multiplier m, then we compute the final bid x_j by:

$$x_j = b_j \left(1 + \frac{m}{100}\right). \tag{2.8}$$

Note that, in this scenario the Base Bid is a fixed amount of money that will be modified by the device multiplier, thus, the final bid will still be a fixed amount of money, in line with the *Manual CPC* payment scheme.

The opening and closing of Tripadvisor sub-campaigns work as previously described for Google.

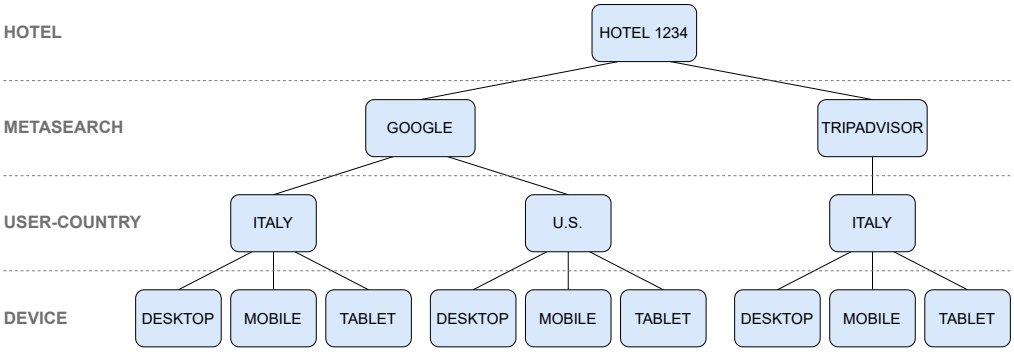

Figure 2.3: Simple example of the tree-structured campaigns for a specific hotel.

2.3.3 Campaign Definition in our Algorithm

Our algorithm defines a campaign differently from what have explained in Sections 2.3.1 and 2.3.2. Indeed, our AI model needs to optimize a given campaign, regardless of the advertising channel. For this reason, in our algorithm a campaign is identified by the tuple *(hotel, metasearch)*, where *metasearch* stands for the advertising channel (i.e., Google or Tripadvisor).

We also consider sub-campaigns in a different way. Specifically, our algorithm identifies a sub-campaign by the tuple *(hotel, metasearch, user-country, device)* for both publishers. This comes to no surprise for Tripadvisor sub-campaigns. However, it may seem strange that, for Google, we do not take into consideration the archetypes Length of Stay, Check-In Day and Booking Window Days. The motivation behind this choice are explained in detail in Chapter 5.

The way in which we identify campaigns and sub-campaigns can be better understood if seen in a tree-structured scheme. In Figure 2.3, we show an example of the tree structure that can be used to represent the campaigns belonging to the fictitious Hotel 1234. In this tree, each campaign can be identified as a sub-tree rooted at the metasearch layer. Moreover, each root-to-leaf path of those sub-trees identifies a sub-campaign. Thus, by looking at the example, we can easily say that Hotel 1234 has two campaigns. The Google campaign has six sub-campaigns, while Tripadvisor has three.

2.3.4 Dataset

In what follows we will describe how the Dataset used to feed the following AI algorithm is stored, its structure, how it is generated, and, finally, the way in which our algorithm retrieves it.

AdsHotel provides us the data in the form of a MariaDB Database (DB). This means that our system will need to access the database to import, clean, and exploit the dataset.

MariaDB is a relational database management system, thus the database is **structured** into multiple tables. The most important ones are:

- *Observations*: this table contains the the full history of observations generated by all hotels. The table contains the following records: date, hotel-

id, metasearch-id, user-country, device, check-in day, los, booking window days, eligible impressions, impressions, clicks, costs, conversions, bid and bid-device. The record eligible-impressions field is estimated by the publisher and it indicates the number of impressions we would have received by winning the auction (i.e., ad displayed in the highest visibility slot). The record bid contains the value of the final bid, which is obtained by applying all the multipliers to the base bid. The record bid-device is instead the value obtained by applying only the device multiplier to the base bid.

- *Constraints*: this table contains the budget and minimum ROI constraints belonging to each campaign. It contains also the min-bid and max-bid constraints corresponding to each at sub-campaign.

- *Hotels*: useful information regarding the hotels can be found here. The most important one is the country in which the hotel resides, that we will denote as *hotel-country* in the rest of the document.

- *MetasearchesCountries*: this table is very important as it discloses, each day and for each campaign, which countries are open and which are closed. This distinction is fundamental for two reasons. First, this gives our algorithm a way to understand which sub-campaigns are currently open, and thus which ones to include in the campaign optimization. Second, exploiting this information our algorithm knows which countries are currently closed in a given campaign, and thus it can propose opening suggestions.

- *OpportunitySuggestions*: this is the table in which we provide our opening suggestions, so that we can expand already running campaigns.

The multiple running campaigns **generate** new data each day on the publishers' platforms. Google and Tripadvisor are able to communicate an observation to AdsHotel two days later from the day the observation actually occurred. As soon as the new observations are received from the publishers, AdsHotel writes them on the Observation table. For this reason, the DB is updated up to the last two days. Note that, all the observations generated in a given day are uploaded on the DB by AdsHotel, as one chunk, at a precise time of the day. In this way, by executing our algorithm after that time, we are sure that the new daily observations have already been uploaded in the DB.

Chapter 3

Online Learning and Monitoring Background

This Chapter provides necessary concepts and notations to let the reader understand and familiarize with the topics covered in the rest of the document. Section 3.1 introduces the Online Learning setting and its main techniques. Section 3.2 presents the field of Online Monitoring, explains why it is fundamental for our work and introduces some of its techniques.

3.1 Online Learning

Online Learning is the sub-field of ML tackling the problem of automatic learning in a setting in which the agent continuously receives new data in sequential order and updates itself at each step to improve its performance. This is different from the classical ML setting, in which the entire training dataset can be accessed to adjust the model at once. Since the model potentially needs to be updated every time new data is received, online learning algorithms need to be fast and efficient both in the updating and predicting phase.

3.1.1 Multi-Armed Bandit

Multi-Armed Bandit (MAB) techniques model a setting in which the agent has a finite set of actions at her disposal and, at each step, she needs to perform one of those actions. Each time the agent performs an action, a random reward, drawn from a distribution specific to that action, is received. These distributions are not known beforehand by the agent. Thus, her goal is to balance the trade-off between exploitation (i.e., perform the action that is believed to yield the highest reward) and exploration (i.e., acquiring more information about the probability distribution of the other actions). MABs are named after slot machine gamblers. Indeed, they have to decide which slot machine to play (i.e., also called one-armed bandit), each one having its reward probability distribution that is unknown by the gambler.

Multi-Armed Bandit techniques are used to address a multitude of real-world

problems like clinical trials, finance [55], recommender systems, budget optimization [36], pricing [51, 39].

3.1.2 MAB: stochastic and stationary setting

We formally present the MAB setting within a stochastic and stationary environment. Given a set of $K \in \mathbb{N}$ arms, at each round $t \in \{1, \dots, N\}$, with $N \in \mathbb{N}$, the agent pulls an arm $I_t \in \{1, \dots, K\}$ and gets a reward $g_{I_t,t} \sim D_{I_t}$, drawn from the distribution of the arm I_t.

The agent's goal is to maximize the cumulative reward over a given time horizon, equivalently, to minimize the regret. The regret is the difference between the optimal reward obtained by always pulling the best arm in every round and the reward of the agent policy, which tries to balance the trade-off between exploration and exploitation.

Given the optimal arm $i^* = \text{argmax}_i E[g_i]$, the expected reward $\mu_i = E[g_i]$ and the optimal expected reward $\mu^* = \mu_{i^*}$, we define the *regret* of the agent A over N rounds as:

$$R_N(A) = \sum_{t=1}^{N} \mu^* - E[\mu_{I_t}], \tag{3.1}$$

where the expectation is with respect to the agent only. By defining $\Delta_i = \mu^* - \mu_i$ as the expected difference between the reward of the optimal arm and the i-th arm, and $P_{i,n}$ as the number of times arm i is selected after n rounds, we can define the regret as:

$$R_N(A) = \sum_{t=1}^{N} \mu^* - E[\mu_{I_t}] = \sum_{t=1}^{N} E[\mu^* - \mu_{I_t}] = \sum_{i=1}^{K} \Delta_i E[P_{i,N}]. \tag{3.2}$$

This definition of regret highlights the fact that, in order to minimize it, we need to limit the number of pulls on sub-optimal arms.

Lai et al. [30] proved in their work that it is impossible to obtain a null regret. Moreover, the lower bound for the regret is asymptotically in the order of $\log(n)$: a *sub-linear regret*.

3.1.3 UCB1

The Upper Confidence Bound (UCB1) [5] is a deterministic algorithm based on the frequentist approach. It manages to achieve a sub-linear regret by taking into consideration the uncertainty of the arms' reward estimation. In fact, UCB1 resorts to the so-called *optimism in the face of the uncertainty*. The main idea is the following: arms that have been pulled a small number of times have large uncertainty in their reward estimate. On the other hand, arms that have been pulled a high amount of times have small uncertainty in their rewards. In this way, UCB1 tackles the trade-off between exploitation and exploration. The expected reward of each arm is paired with a bound, which takes into consideration its uncertainty. Then, at a generic step n, instead of selecting the

arm having the largest sample mean $\bar{g}_{i,n}$, we pull the arm having the highest confidence bound, which is an optimistic estimate of the real mean μ_i:

$$U_n(i) = \bar{g}_{i,n} + B_n(i), \tag{3.3}$$

where $B_n(i)$ is the beforementioned bound.

These bounds are computed exploiting the Hoeffding inequality:

Definition 3.1.1 (Hoeffding Bound)**.** *Given a set of rewards, their expected value $\mathbb{E}[X_i] = \mu_i$, and their sample mean $\bar{x}_i$, we can state that:*

$$P\{\mu_i > \bar{x}_i + \epsilon\} \leq \exp\left(-2t\epsilon^2\right). \tag{3.4}$$

Now, by substituting ϵ with $B_n(i)$, t with the number of pulls $P_{i,n}$, and by assuming that the probability in Equation 3.4 decreases with time as $p(n) = \exp(-4n)$, we can derive the following bound values for UCB1:

$$B_n(i) = \sqrt{\frac{2log(n)}{P_{i,n}}}. \tag{3.5}$$

The asymptotic regret upper bound for the UCB1 algorithm is provided by Auer et al. [5]:

Theorem 3.1.1 (UCB1 Regret Upper bound)**.** *Given a MAB solved by using the UCB1 policy, the expected regret is bounded by:*

$$R_N(UCB1) \leq 8\log(N) \sum_{i,\Delta_i>0} \frac{1}{\Delta_i} + \left(\frac{\pi^2}{3} + 1\right) \sum_{i,\Delta_i>0} \Delta_i. \tag{3.6}$$

3.1.4 Thompson Sampling

The Thompson Sampling (TS) [48] algorithm is based on a Bayesian approach: the arm is not pulled in a deterministic way as it happened for UCB1. Each expected reward is associated with a distribution that we update when we get new rewards. This requires using conjugate prior and posterior distributions so that, using the likelihood of the reward, we can obtain a posterior that can be used as prior in the next iteration. For instance, if the rewards follow a Bernoulli distribution, we use a Beta distribution as prior for the expected value of the arm.

The rationale of the TS algorithm is the following. At each round n, it samples the expected reward from the beta distribution of each arm and pulls the arm I_n having the highest one. Then, it observes the actual reward of the arm and updates the parameters of the prior corresponding to arm I_n. If the observed reward is 1, the α parameter is incremented by one. Otherwise, it increments by one the β parameter. Indeed, given the Bernoulli and Beta conjugate prior and posterior distributions, the update formula is:

$$\begin{aligned} P(\mu|x_{i,n}) &\propto Be(x_{i,n}|\mu)\, Beta_{\alpha+1,\beta+1}(\mu) \\ &\propto \mu^{x_{i,n}}(1-\mu)^{1-x_i} \cdot \mu^{\alpha}(1-\mu)^{\beta} \\ &= \mu^{\alpha+x_{i,n}}(1-\mu)^{\beta+(1-x_{i,n})}. \end{aligned} \tag{3.7}$$

The asymptotic regret upper bound for the TS algorithm is provided by Kaufmann et al. [28]:

Theorem 3.1.2 (TS Regret Upper bound)**.** *For every $\epsilon > 0$, there exists a problem-dependent constant $c(\epsilon, \mu_1, \ldots, \mu_k)$ such that the expected regret by using Thompson Sampling satisfies:*

$$R_N(TS) \leq (1+\epsilon) \sum_{i,\Delta_i>0} \frac{\Delta_i (\log(N) + \log(\log(N)))}{KL(\mu_i, \mu^*)} + C(\epsilon, \mu_1, \ldots, \mu_K). \quad (3.8)$$

3.1.5 Combinatorial Multi-Armed Bandit

The Combinatorial Multi-Armed Bandit (CMAB) [10, 1] differs from a standard MAB given that, at each round, it can pull a subset of the finite arms, called *super-arm*. Specifically, the chosen arms must satisfy some combinatorial constraints, hence the name. To pull the best possible super-arm, an optimization procedure is run, returning the best subset of arms satisfying the given constraints. In this setting, two approaches are possible: we can observe the reward of the pulled super-arm or the reward of each single pulled arm. Both instances can be solved with upper bound approaches. However, the latter approach is preferred, as the number of possible super-arms satisfying the combinatorial constraint may be exponentially large.

CUCB

The CUCB algorithm is the extension of the Upper Confidence Bound (UCB), adapted to the CMAB setting. The construction of the upper confidence bound is similar to UCB one, specifically:

$$U_n(i) = \bar{g}_{i,n} + B_n(i), \quad (3.9)$$

$$B_n(i) = \sqrt{\frac{3\,log(n)}{2\,P_{i,n}}}. \quad (3.10)$$

3.2 Online Monitoring

Machine learning (ML) is a subfield of AI whose aim is the design of algorithms able to learn from data exploiting statistical tools. Typically, in ML, the data are assumed to be independent and identically distributed (i.i.d.) realizations of an unknown process. Therefore, the only focus is centered around how to let data-driven models extract information out of the training data. However, in our scenario we cannot make that assumption because the environment changes through time due to reasons like seasonality, festivities, and unpredictable events (e.g., COVID-19). Thus, we are interested in techniques that do not only learn the model, but also adapt it to changes. In the literature, this environment is commonly called *non-stationary*: at each time instant t we receive an input x_t from a potentially infinite stream $X = \{x_0, x_1, \ldots\}$, we generate a prediction $\hat{y}_t$, then we may retrieve a feedback y_τ ($\tau < t$), and at last we update the model. Therefore, the order of the samples $x \in X$ is relevant and cannot be disregarded.

3.2.1 Problem Statement

The problem of classification over a datastream can be formalized in the following way. $X = \{x_0, x_1, \ldots\}$ is a potentially infinite stream of data derived from the process $\mathcal{X}$, which generates tuples

$$(x_t, y_t) \sim \phi(X, Y),$$

where $\phi(X, Y)$ is the joint probability distribution and Y is the set of true labels. The typical assumption is that data are i.i.d. and that an initial training set R is provided to train a classifier K. Then, at each time instant t, K predicts the label $\hat{y}_t$ given the sample x_t in input. Finally, a classification error e_t is taken into account to measure how good the trained model K matches the distribution $\phi(X, Y)$, formally:

$$e_t = \begin{cases} 1, & \text{if } \hat{y}_t \neq y_t \\ 0, & \text{otherwise} \end{cases}.$$

In real world scenarios, the process $\mathcal{X}$ may change unpredictably, and, thus, the process is modeled such that each sample comes from its own distribution $(x_t, y_t) \sim \phi_t(X, Y)$. Note that, during some time intervals, the distributions may not change or have small changes over time. When this happens we can assume stationary conditions (i.e., also called *concept*) and a traditional classifier K can be reasonably trained over such interval. Instead, we say that a *concept drift* occurs, or equivalently that $\mathcal{X}$ becomes non-stationary, if:

$$\phi_t(X, Y) \neq \phi_{t+1}(X, Y).$$

The task is to learn and adapt a classifier K_t to predict labels $\hat{y}_t = K_t(x_t)$ in an online manner, having a small classification error $\frac{1}{T}\sum_{t=1}^{T} e_t$ over a given time interval T. Thus, it is key to determine when a concept drift happens and adapt the classifier accordingly, otherwise the classification error will gradually increase over time. An example is shown in Figure 3.1

3.2.2 Concept Drift Taxonomy

We defined what a concept drift is in Section 3.2.1, however we did not explain the various drift types yet. Let us recall that, using the Kolmogorov's conditional probability, the join probability distribution $\phi(X, Y)$ can be written as:

$$\phi(X, Y) = \phi(Y|X)\,\phi(X).$$

A *real concept drift* happens when the conditional probability $\phi(Y|X)$ is affected, while $\phi(X)$ might be affected or not. Instead, a *virtual concept drift* occurs if only the input distribution $\phi(X)$ is affected.

Note that there can be also multiple ways in which the underlying process changes during a drift:

- **Abrupt** changes lead into a permanent shift in the state of the process $\mathcal{X}$:

$$\phi_t(X, Y) = \begin{cases} \phi_0(X, Y), & \text{if } t < \tau \\ \phi_1(X, Y), & \text{if } t \geq \tau \end{cases};$$

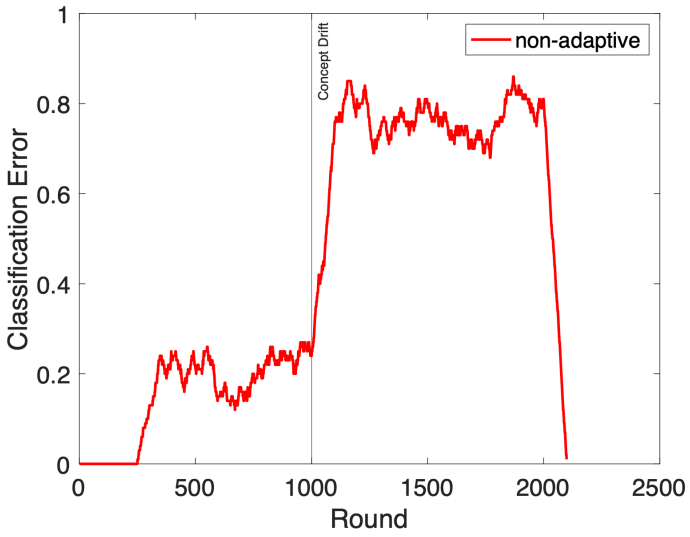

Figure 3.1: Example showing the increase in the classification error right after a concept drift occurring at round 1000.

- a continuous shift condition is called an **incremental** change:

$$\phi_t(X,Y) = \begin{cases} \phi_0(X,Y), & \text{if } t < \tau \\ \phi_t(X,Y), & \text{if } t \geq \tau \end{cases}.$$

  Note that the continuous shift might reach an end, after which there is another stationary state such as $\phi_1(X,Y)$.

- a **recurring** change happens if, after a concept drift, it is possible for $\mathcal{X}$ to go back to the previous concept. It is a sort of alternation between different states, for instance between ϕ_0, and ϕ_1.

- a **gradual** change happens when the process definitively switches to a new state after having anticipated some short drifts:

$$\phi_t(X,Y) = \begin{cases} \phi_0(X,Y) \text{ or } \phi_1(X,Y), & \text{if } t < \tau \\ \phi_1(X,Y), & \text{if } t \geq \tau \end{cases}.$$

3.2.3 Adaptation

We already stated in Section 3.2.1 that, if some precautions are not taken into account, the classification error is going to increase over time after a concept drift happens. One of the crucial contributions for preserving the performances is due to *Adaptation*. Indeed, applying a trivial solution such as continuously updating the model K with new data is not enough: samples belonging to the old concept are still going to be taken into account, thus the classification error will decay too slowly. At the same time, applying a sliding window solution such as training the model using only the last N samples, still leads into some problems: during long concepts the classifier is not able to learn anything it could, since it limits

itself by using only N samples for training. Therefore, more elaborated solution must be taken into account to maintain good performances in non-stationary settings.

Two main adaptation strategies have been presented in the literature, which one is best depends on the problem at hand and on the memory and computational availability. These are:

- **Active Approaches**: the classifier K_t is combined with statistical tools to detect concept drifts and pilot the adaptation. To be coherent with the new concept, they rely on what is called Change Detection Test (CDT), and they specify post-adaptation strategies to isolate recent data generated after the change. Note that these approaches also provide the information that the concept drift has occurred, and they can improve their performances in stationary conditions, since the classifier adapts only after the detection. A downside of active approaches is that incremental and gradual changes are very difficult to detect. The simplest approach consists in *monitoring the classification error* or similar performance measures. Once a change has been detected, we adapt and update the model by providing a new training set. Note that, in this way, the model is updated only when the performance is affected, which is a very good property. On the other hand, the availability of supervised samples is needed to perform CDT and, in a real world scenarios, they are usually provided with significant delays or not provided at all.

  An alternative to the classification error is the one of *monitoring the input distribution*: the input stream goes directly into the CDT to see if it is consistent with the current concept's $\phi(X)$. If a change is detected then you might invoke adaptation and update the model. Monitoring the input distribution does not require supervised samples and enables the detection of both real and virtual concept drifts. However, note that real drifts not affecting $\phi(X)$ cannot be discovered. Moreover, we could be considering changes that do not require adaptation because $\phi(Y|X)$ might not be affected. The biggest downside however, is that it is difficult to design a CDT when the monitored distribution is unknown.

- **Passive Approaches**: the classifier K_t undergoes continuous adaptation determining at each new sample which supervised information to preserve. They do not include a CDT mechanism, thus concept drifts are not detected at all. These approaches are preferred when the changes are of incremental or gradual nature.

3.2.4 Change Detection

In a change detection problem it is usually assumed that data are initially generated under stationary conditions by a process $\mathcal{P}_A$. Then, after a concept drift data are going to be generated by a different process $\mathcal{P}_B \neq \mathcal{P}_A$. The goal is to monitor a stream $X = \{x_t\}_{t \in \mathbb{N}}$ of realizations and to detect the change point τ

such that:

$$x_t \sim \begin{cases} \phi_0, & \text{if } t < \tau \text{ (in control state)} \\ \phi_1, & \text{if } t \geq \tau \text{ (out of control state)} \end{cases},$$

where for $t < \tau$ data are i.i.d. , and $\phi_0 \neq \phi_1$. In other words, we would like to answer the following question: given the previously estimated model and the arrival of new data, is it able to explain those samples? If it is not, a concept drift may have happened [11].

Most of the algorithms to perform CDT are composed of two elements:

1. a *statistic* S that has a known response to normal data (e.g., sample mean, sample variance, log-likelihood);

2. a *decision rule* to analyze S (e.g., a threshold).

Statistics and decision rules are both sequential in the sense that they take into account, in principle, all the data received so far. Integrating information over time makes these algorithms able to detect subtle changes as well. The typical solution consists in refining the raw input stream into a statistic over which decision rules are applied. Decision rules such as thresholds are very simple and they affect performances in terms of having more true positives (TP) or more false positives (FP). For this reason, they should be defined with a sound statistical criteria.

In this work, two state-of-the-art algorithms are used, i.e., CUSUM [26] and ADWIN [9]. We opted to define them in detail further on in the thesis, since their application in our setting requires the definition of quantities which will be presented later on in the thesis.

3.2.5 Performance Measures

Controlling the false positive rate in a CDT is very important as it is a performance measure of the algorithm itself. False positives are controlled by the Average Run Length, formally defined as:

$$ARL_0 = E_x \left[\hat{T} | x \sim \phi_0 \right],$$

where $\hat{T}$ is the detection time. The ARL_0 denotes the expected amount of time between multiple false positives under stationary conditions (i.e., the expected number of samples, generated under ϕ_0, for your algorithm to give you a detection). Indeed, keep in mind that in this setting we are almost sure to face some FP sooner or later, therefore we want to control the number of times that happens. Note that, if a detection happens more frequently than the ARL_0 quantity, which is defined in stationary conditions, then we might be in the situation in which the process is changing or has already changed. Therefore, a good CDT algorithm should always be paired with a table that, given the needed value of ARL_0, produces the related threshold value γ. Notice that in general it is hard to find the proper value for the threshold γ since it depends both on the distribution of the statistic S and on the data distribution $\phi(X)$.

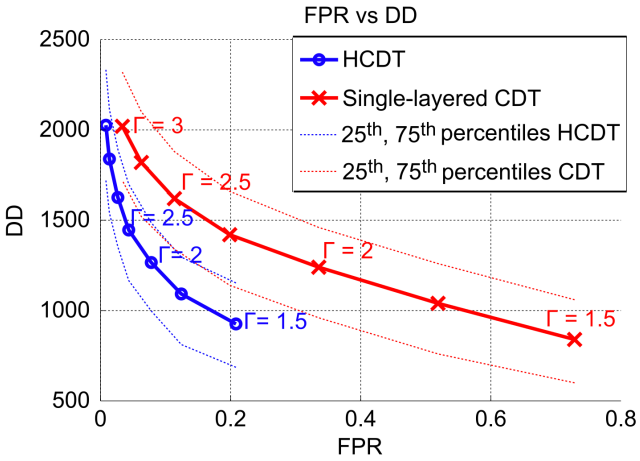

Figure 3.2: FPR vs DD curve example. Figure from [4].

Another performance measure for CDT is the is the expected time delay for detecting the change $\phi_0 \to \phi_1$, or formally:

$$ARL_1 = E_x \left[\hat{T} | x \sim \phi_1 \right].$$

Unfortunately, computing ARL_0 and ARL_1 is not always possible. In such cases, one resorts to performing several simulations on finite sequences containing a change at a known location τ to compute:

1. the detection delay $DD = E_x \left[\hat{T} - \tau | \hat{T} \geq \tau, x \sim \phi_1 \right]$;
2. the false positive rate FPR (i.e., the amount of sequences not containing a drift where a change was detected, divided by the number of sequences not containing a drift);
3. the false negative rate FNR (i.e., the amount of sequences containing a drift that has not been detected, divided by the number of sequences containing a drift).

After computing these quantities for multiple threshold γ values, FPR-DD curves are usually drawn. The FPR values are plotted on the x-axis, while DD ones on the y-axis. This plot is useful due to the fact that these two quantities are in a trade-off: the lower the DD the higher the FPR, the higher the DD the lower the FPR. Therefore, the best solution is to choose the threshold γ leading to the best DD-FPR trade-off for the application at hand. Further information can be found in Alippi et al. [4] work. They present Hierarchical Change-Detection Tests (HCDTs), coupling a validation layer to CDTs. HCDTs achieve a far more advantageous trade-off between false-positive rate and detection delay than their single-layered, more traditional counterpart. An example of the FPR vs DD curve taken from their work is reported in Figure 3.2.

Chapter 4

Related Works

In Chapter 2 and 3, we presented useful background information about the *Online Advertising* and the *Online Learning and Monitoring* fields. We also described the real-world scenario that we face in this work. In this chapter, we present a review of the literature related to the problem we will address in the rest of the document.

4.1 Online Advertising

A core part of an online advertising campaign optimization is the bid-budget optimization problem. In this problem, the advertiser must decide each day the bid and daily budget values for each of her sub-campaigns. This is a crucial part of an online advertising campaign optimization process, for this reason it has be been widely studied by the scientific community. However, most of the state-of-the-art works are general purpose, and do not translate very well in our scenario. The description of the systems currently used by large media agencies, for generic marketing or hotel advertising, are not publicly available. We think this is due to contracts binding research groups with companies, i.e., NDAs, not allowing these works to be published.

4.1.1 Joint bid-budget Optimization

The joint bid-budget optimization problem is addressed for the first time by Zhang et al. [60]. They model the problem as a constrained optimization problem, maximizing the expected revenue subject to the constraints of the budget and of the bid ranges. To solve such problem, the expected revenue as a function of the bid must be estimated for each sub-campaign. They estimate such functions by building a probabilistic model for Ad Ranking: through search advertising log data, they compute the probability for a given item and bid to be ranked at a given slot position in the auction. However, this type of data is rarely available to the advertiser, thus this solution shows limitations in its applicability.

Nuara et al. [37] propose a different approach for the joint bid-budget optimization problem that does not require ad ranking data. They formulate the op-

timization problem as a combinatorial-bandit problem [10], in which a *superarm* corresponds to a combination of bid/budget pairs for each sub-campaign satisfying combinatorial budget constraints. In order to estimate the expected revenue, they use Gaussian Processes (GPs) [57] to estimate, for each sub-campaign, the expected daily number of clicks for each bid/budget pair and the value-per-click. Then, they balance exploration and exploitation through Bayesian Bandit techniques returning samples of the stochastic variables estimated by the GPs. These samples are used to feed a special case of the Multiple-Choice Knapsack problem [44], solved in polynomial time by dynamic programming, returning the superarm maximizing the cumulative expected revenue. However, we cannot adopt this solution for our specific scenario. Indeed, our setting requires additional constraints, e.g., that the bid does not vary too much between consecutive days, and the highly non-stationary environment is in contrast with the assumptions required by this method.

4.1.2 Bid Optimization

Ding et al. [15] addressed the bid optimization problem as a Budgeted Multi-Armed Bandit (BMAB) problem. In this setting, there is a finite set of arms. Each arm corresponds to a bid value and, if pulled, incurs a cost. Moreover, the learning process is constrained by a fixed overall budget. Arms are pulled according to an Upper Confidence Bound (UCB) approach. Xia et al. [58] addressed the problem as a BMAB as well. However, instead of using a UCB approach, they exploited Thompson Sampling to select which arm to pull. Both [15] and [58] solutions have a regret bound of $O(\ln B)$, where B is the overall budget. Trovò et al. [50] proposed, for the first time, to solve the problem as a Budgeted Continuous-Armed Bandit (BCAB), where the set of arms is a continuous set over the range of prices. This algorithm suffers a higher regret compared to the BMAB ones but is more robust to adverse settings.

4.1.3 Safe Bid Optimization

In our scenario the advertisers are *performance advertisers*: they focus on short-term returns. This can be modeled by adding minimum ROI constraints in the campaign optimization problem. Spadaro et al. [45] focus on safe bid optimization in a scenario very similar to ours, in which ROI and budget constraints must be satisfied. They use Gaussian Processes (GPs) to estimate, for each sub-campaign, the expected daily number of clicks and costs for each bid value. The value-per-click is assumed to be known instead. Then, they propose a variant of the GCB algorithm called GCBsafe. It exploits the GPs to return optimistic estimates for the parameters used in the objective function, and to return pessimistic ones for the parameters used in the constraints. More specifically, the lower bounds of the number of clicks and the upper bounds of the costs are used in the ROI and budget constraints, while the upper bounds of the number of clicks are used in the objective function. Then, these estimates are fed into a special case of the Multiple-Choice Knapsack problem, solved by dynamic

programming, returning the superarm maximizing the cumulative expected revenue, while satisfying ROI and budget constraints with high probability. Given the similarities between our scenario and the one for which this algorithm was conceived, we decided to adopt this solution within our system and extend it to match the requirements of the specific application we faced.

Finally, we remark that the concept of *safety* here used is different from the one commonly used in the online learning literature, e.g., [8], due to the specific nature of the problem we are addressing.

4.2 Monitoring

Most Machine Learning techniques assume the environment to be stationary. However, in most real applications, it is likely that the distribution generating the data changes over time. This happens for various reasons, e.g., seasonality and festivities. Thus, under these circumstances it is critical to take into account the non-stationarity of the environment to maintain good performances over time. For this reason, adaptation and monitoring techniques have been extensively addressed in the scientific literature. Adaptation strategies are divided into two categories (i.e., active and passive), each with its own advantages and disadvantages.

4.2.1 Active approaches

Active approaches combine the underlying predictive model with statistical tools to detect concept drift and pilot the adaptation. They rely on outlier detection or change detection test (CDT), and they specify post-adaptation strategies to isolate recent data generated after the change, in order to be coherent with the new concept. Some active approaches consist in monitoring the stream of classification errors, or similar performance measures, of the predictive model. Other approaches monitor the input distribution instead.

Gama et al. [18] propose the Drift Detection Method (DDM) which consists in detecting a concept drift as an *outlier* in the classification error. They assume that in stationary conditions the classification error reduces as the predictive model gets trained with more and more samples. On the other hand, they assume that when a drift occurs the classification error increases. A binomial distribution is used to keep up to date the probability of misclassifying a new sample and its standard deviation. Then, they compare these values with the minimum ones registered in the current concept. Then a warning level and a drift level are defined based on this comparison. The warning level is used to identify a reasonable training set for the new concept. When the drift level is reached, a new classifier is trained using the previously identified training set. Baena-Garcıa et al. [7] propose a slightly different version of DDM called Early DDM (EDDM), which improves the detection in presence of gradual drifts while keeping good performance with abrupt drifts. They perform a similar monitoring on the average distance between misclassified samples instead of considering only the number of errors. A big downside of both DDM and EDDM is heuristic

nature of their monitoring scheme, i.e., they have no control over the FPR. Ross et al. [41] propose a method for detecting concept drifts which uses the exponentially weighted moving average (EWMA) as test statistic to monitor the classification error. Unlike DDM and EDDM, the EWMA method allows the rate of false positive detections to be controlled and kept constant over time. Hinkley in [26] uses the cumulative sum (CUSUM) scheme to detect the point in time in which the mean of the data stream changes. This method can be used both to monitor the classification error and the input. Moreover, it is a two-sided test as it detects both increases and decreases in the mean. However, CUSUM can only detect concept drifts caused by a change in the mean and it cannot detect the ones caused by a change in the variance, for instance.

The classification error can be monitored also by comparing windows. The main idea of this approach is to compare a reference window, ideally containing stationary data, with a window containing recent data. Nishida et al. [34] propose the STEPD method, in which they compare two windows: one containing recent samples, and one containing every past data after the last detection. Bifet et al. [9] presented ADWIN. This method analyzes the whole stream of scalar samples and automatically grows or shrinks the window depending on whether or not a change has been detected. It can be used both to monitor the classification error and the input. Their idea is that whenever two large enough sub-windows of the monitored stream exhibit distinct enough averages, a change is detected and the samples belonging to the old window are dropped. Bach et al. [6] deal with concept drifts by pairing a stable learner with a reactive one. The stable learner is trained over the whole stream, while the reactive one is trained only over a window of recent samples. When the reactive learner outperforms the stable one over recent test samples, a drift is detected. The adaptation is carried out by replacing the stable learner with the reactive one. The underlying idea is that, after a concept drift happens, the stable learner has poor performances while the reactive one adapts very fast to the new concept. The main disadvantage of comparing window techniques is that it is difficult to control the FPR since often they consist of iterating hypothesis tests over non-independent samples (i.e., overlapping windows).

Change-Point Methods (CPMs) are sequential monitoring schemes that can be used both in parametric and non-parametric settings. They do not require training samples and they provide ARL_0 guarantees. These techniques introduced by Hawkins et al. [25] verify whether or not a stream contains a change-point (i.e., the time instant a concept drift happens) by analyzing all possible partitions of the data sequence into two adjacent sub-sequences. These techniques have very good performances, however they require high computational complexity, which makes their use in an online scenario costly. Ross et al. [42] present an approximated formulation of CPMs to allow their use in online settings as well, however the complexity of these solutions is still critical. Alippi et al. [3] address adaptation in non-stationary environments by proposing Just-in-time classifiers. They detect abrupt changes, both real and virtual, and to identify a new training set for the new concept. Moreover, by exploiting practical formalization of the concept representation and the definition of a set of

operators working on such representations, they can identify recurrent concepts and in such a case exploit also past samples, achieving very high performances even right after a drift occurs.

We remark that change point methods have also been applied to the MAB problem to tackle non-stationary environments. See [40, 31] for more details.

4.2.2 Passive approaches

In passive approaches the classifier undergoes continuous adaptation determining at each new sample which supervised information to preserve. There are two main categories of passive approaches, those based on single classifier models and those based on ensemble models. Single classifiers usually provide lower computational cost with respect to ensembles. Many single classifier methods have been presented in the literature [27, 12, 59].

Ensemble methods are composed of multiple models, each typically trained on a different training set. The final prediction is given by a weighted aggregation of the individual predictions. Many ensemble methods have been presented in the literature [46, 29, 17, 33]. They differ in the way new data is incorporated into the ensemble, for instance by updating the already existing ones or by adding new individuals, by the way old data is discarded from the ensemble, or by the way in which the individual predictions are taken into consideration to form the final prediction. Finally, also passive approaches have been used in the classical MAB setting. See [52, 19] for details.

Chapter 5

Safe Bid Optimization with Return-on-Investment Constraints

Bid optimization is one of the main tasks composing an online campaign optimization. The bid value must be assigned to each sub-campaign and it is the main responsible for the performance of the campaign. Usually, the bid must be set to balance the trade-off between achieving high volumes (i.e., maximizing the sales of the advertised product or service) and achieving high profitability (i.e., maximizing the return-on-investment (ROI)). However, we take a particular approach to tackle this trade-off. As a matter of fact, our scenario is different from the usual one. Hotel advertisers are performance advertisers: they mainly focus on short-term returns. Indeed, their goal is to have all their rooms fully booked. This translates into having to satisfy minimum ROI constraints not only at the end but for the whole advertising period. Thus, safe strategies are needed to satisfy these constraints with high probability. In this chapter, we address the safe bid optimization problem in the real-world scenario of hotel advertising. We adopt the solution proposed by Spadaro et al. [45] and develop a system on top of it to interface with the already existing AdsHotel's Platform. Moreover, our system tackles the multiple requirements derived by a deployment in a real setting, which were not considered in the theoretical work by Spadaro et al. [45].

This chapter is structured as follows. In Section 5.1 we report the formal description of the problem, provided by Spadaro et al. [45] for the sake of completeness. In Section 5.2, we describe the numerous problems of the deployment in a real-world hotel advertising scenario, and we provide the approaches and algorithms we exploit to solve them. Finally, in Section 5.3, we show the experimental results.

5.1 Problem Formulation

We are given an advertising campaign $\mathcal{C} = \{C_1, \ldots, C_N\}$, with $N \in \mathbb{N}$, where C_j is the j-th subcampaign, and a finite time horizon of $T \in \mathbb{N}$ days. In this

work, as common in the literature on ad allocation optimization, we refer to a subcampaign as a single ad or a group of homogeneous ads requiring to set the same bid. For each day $t \in \{1, \dots, T\}$ and for every subcampaign C_j, the advertiser needs to specify the bid $x_{j,t} \in X_j$, where $X_j \subset \mathbb{R}^+$ is a finite set of bids we can set in subcampaign C_j. The goal is, for every day $t \in \{1, \dots, T\}$, to find the values of bids that maximize the overall cumulative expected revenue while keeping the overall ROI above a fixed value $\lambda^* \in \mathbb{R}^+$ and the overall budget below a daily value $y_t \in \mathbb{R}^+$. This setting is modeled as a constrained optimization problem at a day t, as follows:

$$\max_{x_{j,t} \in X_j} \sum_{j=1}^{N} v_j \, n_j(x_{j,t}) \tag{5.1a}$$

$$\text{s.t.} \frac{\sum_{j=1}^{N} v_j \, n_j(x_{j,t})}{\sum_{j=1}^{N} c_j(x_{j,t})} \geq \lambda^* \tag{5.1b}$$

$$\sum_{j=1}^{N} c_j(x_{j,t}) \leq y_t \tag{5.1c}$$

where $n_j(x_{j,t})$ and $c_j(x_{j,t})$ are the expected number of clicks and the expected cost given the bid $x_{j,t}$ for subcampaign C_j, respectively, and v_j is the value per click for subcampaign C_j. Moreover, Constraint (5.1b) is the ROI constraint, forcing the revenue to be at least λ^* times the incurred costs, and Constraint (5.1c) keeps the daily spend under a predefined overall budget y_t.

The available options consist in the different values of the bid $x_{j,t} \in X_j$ satisfying the combinatorial constraints of the optimization problem, while $n_j(\cdot)$ and $c_j(\cdot)$ are unknown functions, defined on the feasible region of the variables, that we need to estimate within the time horizon T. Here the value-per-click v_j is assumed to be known.

A learning policy $\mathfrak{U}$ solving such a problem is an algorithm returning, for each day t, a set of bids $\{\hat{x}_{j,t}\}_{j=1}^{N}$. The policy $\mathfrak{U}$ can only use estimates of the unknown number-of-click and cost functions built during the learning process. Therefore, the returned solutions may not be optimal and/or violate Constraints (5.1b) and (5.1c) computed on the true functions. Some platforms allow the advertisers to set a daily budget constraint. If this feature is allowed, Constraint (5.1c) is always satisfied, and no safety requirement for that constraint is necessary.

5.2 Proposed Method

In Section 5.1, we reported the safe bid optimization problem formulation under ideal theoretical conditions. However, as one can expect, deploying a solution in a real scenario requires its adaptation to the specific setting. In this section, we give a general overview of our system *Meta-Algorithm*. In what follows, we expand some of its details, each of which concerning a key problem encountered during the system deployment, and provide a solution for each one of them.

The *Meta-Algorithm* pseudo-code is provided in Algorithm 1. We remind the reader that, as we previously stated, our system is wrapped around Spadaro et

Algorithm 1 Meta-algorithm

Require: set H of active hotels

1: set sliding window size w
2: **for** $h \in H$ **do**
3: retrieve cleaned set O_h of observations
4: **for each** $\mathcal{C}$ **do**
5: retrieve $O_{\mathcal{C}}$
6: estimate $v_{\mathcal{C}}$
7: preprocess $O_{\mathcal{C}}$
8: $S_P = SingletonAggregation(\mathcal{C})$
9: retrieve λ^*, y_t and X_j $\forall C_j \in S_P$
10: define new exploration constraints $\bar{X}_j$ $\forall C_j \in S_P$
11: initialize the GCBsafe agent
12: $\{\hat{x}_{j,t}, \forall C_j \in S_P\} = DynamicROIOpt(\mu, \lambda^*)$
13: write $\{\hat{x}_{j,t}, \forall C_j \in S_P\}$ on DB
14: **end for**
15: **end for**

al. [45] solution, thus Algorithm 1 will contain and refer to some of the procedures presented in their work. Given the set H of hotels with at least one active campaign on the current day t, the algorithm assigns, for each campaign of hotel $h \in H$, the bid value of every sub-campaign $C_j \in S_P$ such that Constrains 5.1b and 5.1c are satisfied with high probability.

At the beginning, we initialize the sliding window size w, that will be used later to cope with the non-stationarity of the environment (Line 1). More information about it will be provided in Section 5.2.3. Then, we cycle through every hotel h having at least one active campaign (Lines 2-15). We retrieve all the observations O_h, generated by hotel h in the last w days, by querying the DB (Line 3). During the query, we discard incomplete and/or erroneous tuples that would otherwise affect the following GPs estimations procedure. This process is detailed in Section 5.2.1. After that, we loop through every active campaign $\mathcal{C}$ of hotel h (Lines 4-14). At Line 5, we retrieve from O_h only the observations $O_{\mathcal{C}}$ corresponding to the current campaign $\mathcal{C}$. Next, we estimate the value-per-click $v_{\mathcal{C}}$ for campaign $\mathcal{C}$ (Line 6). Note that differently from what specified in Equations 5.1, the value-per-click in the original definition of the problem by Spadaro et al. [45] is defined at a campaign level, and it was assumed to be known. Instead, in this thesis we are dealing with a real-world setting in which we do not know the value-per-click, thus we need to estimate it. Moreover, from expert advice and from a preliminary analysis of the available data, we encountered a serious scarcity of data, which has been caused by traveling restrictions imposed for the COVID-19 pandemic. Thus, taking into account singularly the conversions of each sub-campaign would have led to noisy and unreliable value-per-click estimations. For this reason, we estimate the value-per-click at a campaign level, formally:

$$v_{\mathcal{C}} = \frac{k}{c},$$

where k and c are the total amount of conversions and clicks generated in the

last w days by campaign $\mathcal{C}$, respectively. In Line 7, we prepare the raw observations to be handled by the GP estimation routine. This process is detailed in Section 5.2.2. Next, we run the *SingletonAggregation*($\mathcal{C}$) procedure, described in Chapter 6 in Algorithm 4 (Line 8). The goal of this procedure is to identify the set S_P of sub-campaigns having enough data on their own to compute good *clicks* and *number of clicks* function estimates through GPs. Moreover, we will provide bid suggestions only for these sub-campaigns. In Line 9, we retrieve the constraints from the DB table *Constraints*. Specifically we get the minimum ROI value λ^*, the campaign daily budget y_t, and the set X_j of allowed bids for each sub-campaign $C_j \in S_P$. Note that, neither Google nor Tripadvisor allow the specification of the campaign daily budget y_t, they only allow to set the total campaign budget. However, a fictitious y_t is set by AdsHotel so that we still are able to use Spadaro et al. [45] solution. Moreover, this does not in any way alter the performance of our algorithm since the provided y_t is much larger than the real daily budget expenditure. Next, following the request of AdsHotel, we restrict the set X_j of each selected sub-campaign $C_j \in S_P$ depending on their recent performance (Line 10), obtaining a new set $\bar{X}_j$ for each of them. The motivations and the details behind this step are explained in Section 5.2.4.

The Safe Gaussian Combinatorial multi-armed Bandit (GCBsafe) proposed by Spadaro et al. [45] is applied in Line 11. Here, GPs are used to model the *costs* and *number of clicks* functions, as they provide several advantages compared to other regression techniques. Indeed, exploiting arm correlation they are able to provide an estimate on the entire domain by just using a finite amount of samples. Moreover, we use a combinatorial bandit because we need to select more than a single arm. As a matter of fact, we run an optimization procedure to identify the best arm to be pulled while still satisfying the constraints. We assign and train two GPs for each sub-campaign $C_j \in S_P$ by using their corresponding observations. We used a GP to estimate the *costs* function, and one to estimate the *number of clicks* function. For each sub-campaign $C_j \in S_P$ and for every bid value $x \in \bar{X}_j$, we sample the mean and standard deviation estimations for both *costs number of clicks* functions. Finally, we run our optimization algorithm *DynamicROIOpt*(μ, λ^*) to get the set of bids to suggest $\{\hat{x}_{j,t}\ , \forall C_j \in S_P\}$ (Line 12) and we write them on the DB (Line 13).

In what follows, we detail the subroutine used in the definition of the high-level algorithms described above.

5.2.1 Data Cleaning

The data cleaning process is crucial for any successful application of the automatic techniques described in this thesis. Indeed, we need to feed our model with meaningful data, discarding the misleading ones. Some of the data which should not be processed by our algorithms are since they are inconsistent:

- Tuples with $bid = 0$ and $costs, clicksorimpressions > 0$.
- Tuples with $costs > 0$ and $clicks = 0$.
- Tuples with $clicks > impressions$.

Thus, the cleaning procedure consists into accepting only the tuples satisfying any of the following three constraints:

- $bid > 0 \quad \wedge \quad (costs = 0 \wedge clicks = 0 \wedge impressions \geq 0)$;
- $bid > 0 \quad \wedge \quad (costs > 0 \wedge clicks > 0 \wedge impressions \geq clicks)$;
- $bid = 0 \quad \wedge \quad (costs = 0 \wedge clicks = 0 \wedge impressions = 0)$.

5.2.2 Observation Preprocessing

One of the first problem we encountered while interfacing our system with AdsHotel's Platform was the need of data preprocessing. We already described how the Dataset is structured in Section 2.3.4. Convdersely, in this section we describe how we handle raw data to make it ready for the GP estimation. Each day, we receive one observation for each sub-campaign. However, as already described in Section 2.3.3, there is a discrepancy between how our algorithm and AdsHotel's Platform define a sub-campaign. Specifically, we recall that our algorithm identifies both Google and Tripadvisor sub-campaigns by the tuple *(hotel, metasearch, user-country, device)*. On the other hand, AdsHotel's Platform identifies Tripadvisor sub-campaigns by the tuple *(hotel, metasearch, user-country, device)* and Google sub-campaigns by the tuple *(hotel, user-country, device, los, check-in day, booking window days)*.

Now we will explain the reason behind this mismatch. Given a campaign, our algorithm assigns a bid value for each sub-campaign $C_j \in S_P$. However, the AdsHotel's Platform does not allow to set a bid for each sub-campaign: the bid is indirectly defined by the Base Bid and a combination of multipliers. Therefore, we need to blend the definitions of bids in our algorithm and those of the platform. More specifically, we need to convert the platform Base Bid plus a combination of multipliers into the bid chosen by the algorithm, and vice versa. Obtaining the bid value starting from the Base Bid and the multipliers is always possible and trivial, and is explained in Equations (2.7) and (2.8) for Google and Tripadvisor, respectively. On the other hand, obtaining the Base Bid and the multipliers starting from the bid value is not always possible. For Tripadvisor sub-campaigns, given the bid value x_j that we would like to suggest for sub-campaign C_j, we set the Base Bid $b_j = 1$ and then solve Equation (2.8) for variable m in the following way:

$$\begin{aligned} x_j &= b_j \left(1 + \frac{m}{100}\right), \\ m &= 100x_j - 1. \end{aligned} \tag{5.2}$$

Instead, performing such an action for a Google sub-campaign might be impossible due to the fact that multipliers m change the bid set for different kind of searches at the same time, i.e., changing the los multiplier influence the researches for any check-in date. This is exactly the reason why we must identify a Google sub-camapaign as *(hotel, metasearch, user-country, device)*. By doing this, we can use the Equation (5.2) also for setting a bid for Google sub-campaigns.

Therefore, our algorithm receives each day multiple samples corresponding to the same Google sub-campaign *(hotel, metasearch, user-country, device)*: potentially one for each value of the three multipliers los, check-in day, and booking window days. This is where the preprocessing task gets crucial. Given a sub-campaign *(hotel, metasearch, user-country, device)*, we take all the received new samples and we merge them, creating a single observation. More specifically, the *impressions* of the resulting observation will be equal to the total amount of impressions generated by all those samples. The same happens for *clicks*, *costs* and *conversions*. Instead, the *bid device* is trivially the same for all the samples, so we just replicate that value and place it into the resulting observation.

Notice that what we described above relies on the assumption that all the other multipliers (i.e., los, check-in day and booking window days) must remain untouched during the learning process. As a matter of fact, given a campaign C_j, we estimate the *costs* and *number of clicks* functions, both depending on the bid value. We estimate those functions through GPs by feeding them with the *costs* and *clicks* values of the received observations, each linked to a certain *bid* value that we suggested to AdsHotel. Thus, it is utterly important that the other multipliers remain constant, so that we have a correlation between our suggestions and the received observations, without introducing any latent variable changing the state of the problem. Otherwise, the observations we receive would depend also on external factors, i.e., how the other three multipliers have been changed, and not by our suggested bid. However, experts in the field reported that hotel managers rarely change the multipliers of their campaigns, thus we can make the above assumption.

5.2.3 Non-Stationarity

A problem often encountered when deploying an AI system in a real-world scenario is that the distribution generating the data changes over time. Our case is no exception to this problem due to reasons like seasonality, national holidays and unpredictable events like COVID-19. Thus, under these circumstances, taking into account the non-stationarity of the environment is critical to maintain good performances over time.

To adapt the model to the environment changes, we use a sliding window approach. This adaptation method is simple and effective, and only requires to set one hyperparameter: the size w of the sliding window. It consists in using only the data that has been collected recently to build the *costs* and *number of clicks* function estimates. Specifically, we take into account only the observations generated in the last w days. Noitce that by choosing a value of w that is too small, we are not able to learn accurate function estimates, as the noise in the data may be too high. This means that we would feed inaccurate estimations into the optimization procedure, which in its turn would provide us non-optimal bid values. Instead, assigning a value for w that is too large might include data coming from a different concept, with respect to the current one, into the model estimation.

Therefore, choosing the right sliding window size w is crucial to obtain good

performances. In this setting we rely on the help of AdsHotel field experts, that suggested that a proper sliding window size would be $w = 60$, i.e., approximately two months. As a matter of fact, they believe that it is rare for a concept drift to occur in intervals of less than two months.

5.2.4 Constrained Bid Exploration

A crucial problem of working in a real-world online advertising scenario is that we cannot provide bid suggestions that might alert the advertiser, even if our algorithm would just use them to temporarily explore the bid space X_j to build more accurate estimates and perform better at a later stage. This problem is already tackled by the safe nature of our algorithm, which limits the exploration by providing only bid suggestions satisfying ROI and budget constraints with high probability. However, AdsHotel experts would also like our bid recommendations not to deviate too much from their current value. In this way, at least in the initial stages of testing, they can evaluate our suggestions by hand and see whether or not they are reasonable and coherent.

For this reason, for each sub-campaign $C_j \in S_P$, we decided to further restrict the set X_j of possible bid values, obtaining the new set $\bar{X}_j$. More precisely, $\bar{X}_j$ is defined by taking into account the set X_j, the current bid value x_j and an exploration percentage $\epsilon_j \in [0, 1]$ defining how much we can deviate at most from x_j. As an example, imagine that the interval of possible bid values is $X_j = [0.10, 1.50]$, that the last bid is $x_j = 1.00$ and that the exploration percentage is $\epsilon_j = 20\%$. Our suggestion can be only inside the interval $\bar{X}_j = [0.80, 1.20]$. Note that, the exploration percentage is defined at sub-campaign level. Indeed, using the same ϵ for every sub-campaign is not a viable solution since they can be very different from one another. Therefore, we would like to be much more cautious with our bid suggestion whenever the sub-campaign's performance has been very good recently and to be more flexible if the sub-campaign is not performing well. AdsHotel's experts want the maximum allowed exploration percentage to be 30% and the allowed minimum to be 5%. In this way our recommendation can never be much far from the last bid value. At the same time, even if the sub-campaign at hand is performing very well, we always have some room to explore and, thus, a chance to further increase its performance.

Now, we describe the first approach we took to define the exploration percentage ϵ_j. The first step consists into evaluating the performance of sub-campaign C_j, since ϵ_j depends on it. Here we decided to use the return-on-investment (ROI) as performance measure. We estimate the ROI_j of the current sub-campaing C_j and we compare it to the global ROI_g value of the market (i.e., the ROI over all active campaigns). Specifically, the performance p_j of sub-campaign C_j is computed in the following way:

$$p_j = \frac{ROI_g - ROI_j}{ROI_g},$$

where ROI_j and ROI_g are estimated using the observations received in the last w days (i.e., sliding window $w = 60$ defined in Sectiont 5.2.3). Precisely, we

estimate $ROI_j = \frac{k_j}{y_j}$, where k_j and y_j are the total amount of conversions and costs generated in the last w days by sub-campaign C_j, respectively. On the other hand, $ROI_g = \frac{k_g}{y_g}$, where k_g and y_g are the total amount of conversions and costs generated in the last w days by all active campaigns, respectively. At this point we need to define a function $\varphi(p_j)$ that, providing a sub-campaign's performance p_j, returns the allowed exploration percentage ϵ_j to be applied over the current bid value x_j. We define such function $\varphi(p_j)$ as follows:

$$\varphi(p_j) = \underline{e} + (\overline{e} - \underline{e}) \max\{0, p_j\}, \tag{5.3}$$

where $\underline{e} = 5$ and $\overline{e} = 30$ are the previously introduced minimum and maximum allowed exploration percentages, respectively. Thus, given a campaign C_j, our algorithm computes the exploration percentage $\epsilon_j = \varphi(p_j)$ and, finally, defines the bid suggestion interval:

$$\bar{X}_j = \left[x_j \left(1 - \frac{\epsilon_j}{100}\right), x_j \left(1 + \frac{\epsilon_j}{100}\right)\right]. \tag{5.4}$$

Note that, while defining the new set of allowed bid values $\bar{X}_j$, the following statement $\bar{X}_j \subseteq X_j$ must be satisfied. Indeed, we cannot include bid values in $\bar{X}_j$ that were not allowed in the first place. This approach, however, did not provide good ϵ_j values due to multiple reasons. First, as already stated, using conversions in a period full of traveling restrictions can lead to inaccurate and unreliable estimates. Moreover, comparing the sub-campaign ROI_j to the global one ROI_g is not ideal: if the vast majority of campaigns have low ROI values while the current one has a high one, we are comparing against a very low bar. Another reason is that the $\varphi(p_j)$ function is linear and, according to field experts, a non-linear one may be preferred.

To get rid of these problems we design a second approach. First, rather than using the return-on-investment as performance measure, we use impressions. Then, we evaluate the performance p_j of sub-campaign C_j in the context of its campaign $\mathcal{C}$, rather than in a global one:

$$p_j = 100 \ \frac{i_j}{i_c},$$

where, i_j and i_c are the total impressions collected by sub-campaign C_j and campaign $\mathcal{C}$ in the last n days, respectively. Thus, p_j stands for the impressions percentage of sub-campaign C_j over the ones of campaign $\mathcal{C}$. AdsHotel's experts suggested that the current performance of a sub-campaign can be inferred by looking at the last two weeks, thus we set $n = 14$ days.

Note that, through performance analysis, we have seen a recurrent behavior in the performance of sub-campaigns: mobile ones generally yield larger portions of the total impressions of their campaign. This is due to the fact that it is by far the most used device to search for Hotels on the Internet. Specifically, we have seen that important mobile sub-campaigns yield at least $\overline{p}_m = 30\%$ of the total impressions. Instead, important non-mobile sub-campaigns yield at least

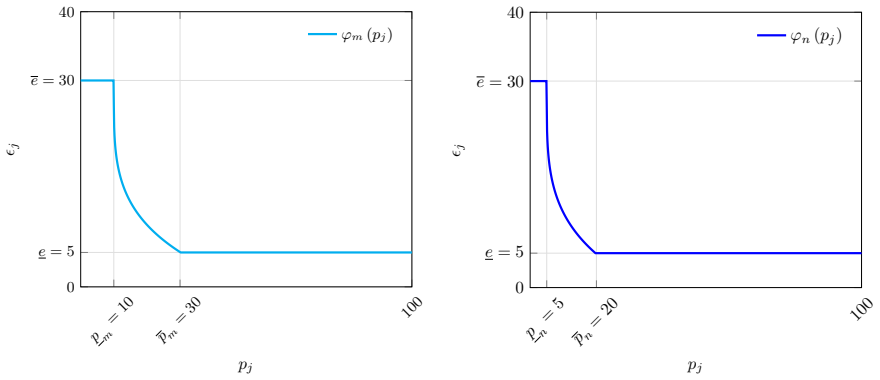

Figure 5.1: Constrained bid exploration functions. (left) Mobile function $\varphi_m(p_j)$. (right) Non-mobile function $\varphi_n(p_j)$.

$\overline{p}_n = 20\%$ of the total impressions. For this reason, we decided to define two different functions: one for mobile sub-campaigns $\varphi_m(p_j)$ and one for non-mobile sub-campaigns $\varphi_n(p_j)$. Both functions, given a sub-campaign performance p_j, return the allowed exploration percentage ϵ_j to be applied over the current bid value x_j. Function $\varphi_m(p_j)$ is defined such that: (1) $\epsilon_j = \underline{e}\ \forall p_j \geq \overline{p}_m$, (2) ϵ_j varies in a non-linear fashion in the interval $\underline{p}_m \leq p_j \leq \overline{p}_m$, and finally (3) $\epsilon_j = \overline{e}$ $\forall p_j \leq \underline{p}_m$, where $\underline{p}_m = 10\%$. On the other hand, function $\varphi_n(p_j)$ is defined such that: (1) $\epsilon_j = \underline{e}\ \forall p_j \geq \overline{p}_n$, (2) ϵ_j varies in a non-linear fashion in the interval $\underline{p}_n \leq p_j \leq \overline{p}_n$, and finally (3) $\epsilon_j = \overline{e}\ \forall p_j \leq \underline{p}_n$, where $\underline{p}_n = 5\%$. Formally, we define $\varphi_m(p_j)$ and $\varphi_n(p_j)$ as:

$$\varphi_m(p_j) = \max\left\{\underline{e}, \overline{e} - \alpha_m \max\left\{0, p_j - \underline{p}_m\right\}^{\beta}\right\}, \tag{5.5}$$

$$\varphi_n(p_j) = \max\left\{\underline{e}, \overline{e} - \alpha_n \max\left\{0, p_j - \underline{p}_n\right\}^{\beta}\right\}, \tag{5.6}$$

where $\underline{e} = 5$ and $\overline{e} = 30$ are the previously introduced minimum and maximum allowed exploration percentages, respectively. Instead, $\alpha_m = \frac{27}{20}18^{\frac{3}{4}}$, $\alpha_n = \frac{27}{20}20^{\frac{3}{4}}$ and $\beta = \frac{1}{4}$ are used to define the non-linear behaviour of $\varphi_m(p_j)$ and $\varphi_n(p_j)$ in intervals $\underline{p}_m \leq p_j \leq \overline{p}_m$ and $\underline{p}_n \leq p_j \leq \overline{p}_n$, respectively. The trend of both functions can be seen in Figure 5.1.

Moving on, given a campaign C_j, our algorithm computes the exploration percentage using $\epsilon_j = \varphi_m(p_j)$ or $\epsilon_j = \varphi_n(p_j)$, depending on the device, and, finally, defines the bid suggestion interval $\bar{X}_j$ as described in Equation (5.4). Note that, also here, while defining the new set of allowed bid values $\bar{X}_j$, the following statement $\bar{X}_j \subseteq X_j$ must be satisfied. Indeed, we cannot include bid values in $\bar{X}_j$ that were not allowed in the first place.

5.2.5 Dynamic ROI Constraint

Given a campaign $\mathcal{C}$, the optimization algorithm $Opt(\mu, \lambda^*)$, defined by Spadaro et al. [45], solves the problem in Equations (5.1). It finds a set of bids, one

Algorithm 2 *DynamicROIOpt*$(\mu, \lambda^*, \bar{S})$

Require: vector of GPs' samples μ, minimum ROI value λ^*, set $\bar{S}$ of latest observed bids

1: $S = Opt(\mu, \lambda^*)$
2: **if** S not feasible **then**
3: **return** $\bar{S}$
4: **else**
5: **while** S feasible **do**
6: $\bar{S} = S$
7: $\lambda^* = 2\lambda^*$
8: $S = Opt(\mu, \lambda^*)$
9: **end while**
10: $I = [\lambda^*, 2\lambda^*]$
11: **repeat**
12: select λ in the middle of interval I
13: $S = Opt(\mu, \lambda)$
14: **if** S feasible **then**
15: $\bar{S} = S$
16: $I = I \setminus \{x \in I : x < \lambda\}$
17: **else**
18: $I = I \setminus \{x \in I : x > \lambda\}$
19: **end if**
20: **until** $I \neq \varnothing$
21: **end if**
22: **return** $\bar{S}$

for each sub-campaign, maximizing the revenue while satisfying the ROI and budget constraints. The minimum ROI value λ^*, that needs to be satisfied, is communicated to us through the *Constraints* table of the DB. However, under the periods of the COVID-19 pandemic, AdsHotel asked us to solve the problem in Equations (5.1) with the highest possible ROI value $\bar{\lambda} \geq \lambda^*$ since some of the ROI set might not be feasible.

We solve this problem through a binary search approach in the *DynamicROIOpt*(μ, λ^*) procedure, whose pseudo-code is provided in Algorithm 2. Given the vector μ containing samples from the GPs, the minimum ROI value λ^* to be satisfied, and given the set $\bar{S}$ of latest observed bids, our task is to find the highest possible ROI value $\bar{\lambda} \geq \lambda^*$ such that $Opt(\mu, \bar{\lambda})$ is able to find a feasible solution $\bar{S}$ (i.e., set of bids satisfying the ROI and budget constraints) and return $\bar{S}$. If there is no such solution for any $\bar{\lambda} \geq \lambda^*$, then we return the set of latest observed bid.

In the first steps, we see whether or not the $Opt(\mu, \lambda^*)$ procedure returns a feasible solution S. Indeed, if it does not, we can stop the algorithm right away and return the latest observed bids (Lines 1-3). Instead, if S is feasible, we may find out $\bar{\lambda} > \lambda^*$ leading to a further feasible solution, thus we go on with the algorithm (Line 4-22). The next goal is to find out the interval I of ROI values in which $\bar{\lambda} > \lambda^*$ may be placed. We do that by running multiple times the $Opt(\mu, \lambda^*)$ procedure (Lines 5-9), each time using a higher λ^* value (Line 7),

until the returned solution S is not feasible anymore. As soon as this happens, we define the interval $I = [\lambda^*, 2\lambda^*]$ (Line 10). After that, until the interval I is not empty, we select λ equal to the middle element of I, we run the $Opt(\mu, \lambda)$ procedure and based on the result we update I (Lines 11-20). Specifically, each time a feasible solution S is found with the given λ, we discard from I all ROI values lower or equal than λ (Line 16). On the other hand, when no feasible solution is found by $Opt(\mu, \lambda)$, we discard all ROI values higher or equal than λ (Line 18). At last, when the interval I becomes empty, we return the latest feasible solution we found $\bar{S}$ (line 22).

5.3 Experimental Evaluation

The bids suggested by our *Meta-Algorithm* are not applied automatically to the various sub-campaigns. First, they are proposed to hotel managers through notifications in the AdsHotel platform. After that, it is up to them to employ such bids or not. Unfortunately, no hotel has been applying our suggestions consistently over time yet, and thus we cannot show and evaluate the performance of our algorithm.

Anyhow, evaluating the performance of our *Meta-Algorithm* in a real-world scenario would be challenging for multiple reasons. First of all, A/B testing strategies cannot be applied in our setting. For obvious reasons, we cannot duplicate a running campaign and apply our suggestion to only one of the two to compare the results. As an alternative, we could look for two active campaigns that have achieved very similar performance over a sustained period of time, then apply our suggestions to just one of them and compare the results. However, finding such campaigns, even relying on expert advice, is not possible in our setting due to the specific characteristics of the different hotels. For these reasons, different campaigns cannot be compared, leaving us with the only option consisting of evaluating the performance of a campaign over time. For instance, we could compare the before/after performance with respect to the adoption of our bid suggestions. However, complications affect this idea as well. Specifically, the non-stationary nature of the environment may invalidate our performance comparison if a concept drift happens close to the adoption of our bid suggestions.

Even if we cannot evaluate the performance of the *Meta-Algorithm*, we can still show the bid values it would suggest for a real-world hotel campaign. We run the algorithm on the dataset previously described in Chapter 2. Precisely, the dataset is composed of the observations generated in the last $w = 60$ days, specifically from 19 August 2020 to 19 October 2021. Given the campaign $\mathcal{C}$ = *(hotel, google)*[1], the $SingletonAggregation(\mathcal{C})$ procedure selects a set S_P composed of the following sub-campaigns:

- *(hotel, google, mexico, mobile)*;
- *(hotel, google, united states, desktop)*.

[1] Due to NDAs, we cannot disclose the actual name of the hotel.

	X_j	x_j	p_j	ϵ_j	$\bar{X}_j$	$\hat{x}_j$	$\bar{\lambda}$
(mx, mobile)	$[0.00, 2.21]$	1.19	53.6%	5%	$[1.13, 1.25]$	1.13	4.01
(us, desktop)	$[0.00, 2.21]$	1.47	7%	14.5%	$[1.25, 1.69]$	1.69	

Table 5.1: Details on the experiment run with our Meta-Algorithm on the real campaign (hotel, google). In the sub-campaign definition, mx stands for Mexico, while us stands for United States.

The *costs* and *revenue* function estimates of each sub-campaign are shown in Figure 5.2. Note that the *revenue* function estimate is obtained by multiplying the *number of clicks* by the value-per-click v_C.[2] As expected, GPs are more accurate in the vicinity of the observations. Specifically, the higher the number of observations, the lower the uncertainty of the estimates. The shape of the *(united states, desktop)* GPs are as expected: almost a linear function with a positive slope and passing from the origin. Instead, the shapes of GPs for sub-campaign *(mexico, mobile)* deviate a little from the beforementioned ideal shape. Thus, ideally, we should explore other bid values to gather information on the most uncertain regions of the estimates. We show the results of the experiment in Table 5.1. Specifically, for each sub-campaign $C_j \in S_P$, we display the initial set X_j of allowed bid values, the current bid value x_j, the impression performance p_j, the exploration percentage ϵ_j, the new set $\bar{X}_j$ of allowed bid values, the suggested bid $\hat{x}_j$, and finally, the satisfied ROI value $\bar{\lambda}$. The performance p_j of sub-campaign *(mexico, mobile)* is quite high as it accounts for 53.6% of the campaign impressions. For this reason, the exploration percentage ϵ_j defined by Equation 5.5 is very low, and the suggested bid cannot differ much from the current value. As a result, we will not be able to explore as we would like. On the other hand, *(united states, desktop)* is not performing well as it account for only 7% of the campaign impressions. As a result, we can explore more with respect to the other sub-campaign.

[2] Due to NDA reasons, we cannot disclose the actual cost and revenue values of the GPs, therefore we opt not to explicit the values on the y axes.

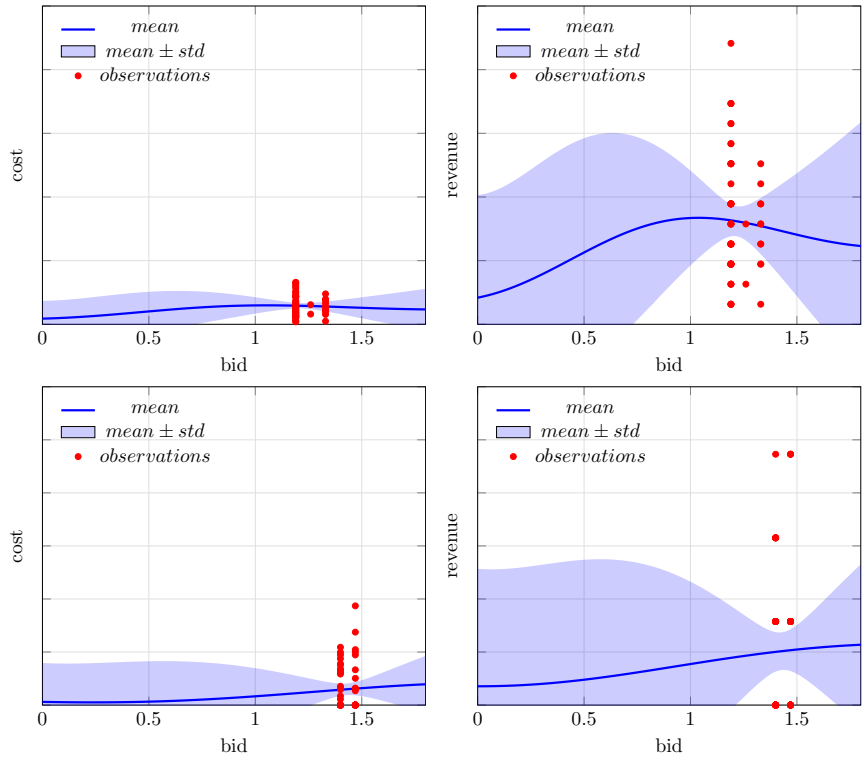

Figure 5.2: Cost and Revenue GPs of each sub-campaign $C_j \in S_P$. (top left) Cost GP of the (mexico, mobile) sub-campaign. (top right) Revenue GP of the (mexico, mobile) sub-campaign. (bottom left) Cost GP of the (united states, desktop) sub-campaign. (bottom right) Revenue GP of the (united states, desktop) sub-campaign.

Chapter 6

Context Aggregation

A fundamental part of the algorithm described in Chapter 5 consists of estimating the following unknown stochastic functions through Gaussian Processes (GPs): the *costs* and the *number of clicks* as functions of the bid. The estimated number of clicks will be then combined with the value-per-click (vpc), to obtain an estimation of the revenue as a function of the bid. Then, we feed the campaign optimization routine with the estimated lower and upper bounds on the revenue and cost functions. For this reason, it is crucial to have accurate estimates and have a reasonably large amount of clean data as well. Unfortunately, for almost the entire duration of the project, we encountered a serious lack of data due to travel restrictions imposed for the COVID-19 pandemic. To overcome this problem, we define context aggregation strategies to group up the data available in each sub-campaign. By doing so, we manage to feed our GPs with enough information to build accurate estimates for a larger number of sub-campaigns, even for those with little to no data.

This chapter is structured as follows. In Section 6.1 we provide a formal description of the problem. In Section 6.2 we describe our approach and algorithms. Finally, in Section 6.3 we present the experimental results.

6.1 Problem Formulation

A campaign $\mathcal{C} = \{C_1, \dots, C_N\}$ is composed by $N \in \mathbb{N}$ sub-campaigns. We define the value function $v\colon 2^{\mathcal{C}} \to \{0, 1\}$ that, given a subset of $\mathcal{C}$, returns either 0 or 1. It returns 1 if, by combining the data of all the sub-campaigns in the subset, the estimated *costs* and *number of clicks* functions are accurate enough. It returns 0 otherwise.[1] We define a solution $S = \{A_1, \dots, A_K\}$ as a partition of the set $\mathcal{C}$ s.t.: $\bigcup_{i=1}^{K} A_i = \mathcal{C} \ \wedge \ (A_i \cap A_j = \varnothing \, \forall 1 \leq i, j \leq K, i \neq j)$. The goal is to find the solution S^* such that:

$$S^* = \underset{S \in 2^{\mathcal{C}}}{\operatorname{argmax}} \sum_{A \in S} v\,(A), \tag{6.1}$$

[1] With $2^{\mathcal{C}}$ we denote the power-set of the set $\mathcal{C}$.

where $\sum_{A \in S} v(A)$ is the cumulative value of the solution S.

Note that, if the *costs* and *number of clicks* functions were known, the solution S^* would be the trivial partition $S^* = \{A_1, \ldots, A_N\}$, where $A_i = \{C_i\}$ $\forall i \in \{1, \ldots, N\}$. However, in our scenario, these functions are unknown and must be estimated. Therefore, combining together multiple sub-campaigns having poor data can be advantageous.

6.2 Proposed Method

The aggregation problem described above is known to be NP-complete [43, 53]. However, it can be simplified by defining in a certain way the granularity level of the aggregations A_i composing the solution S. We recall that the a campaign is identified by the tuple *(hotel, metasearch)*; instead we identify a sub-campaign by *(hotel, metasearch, user-country, device)*. Therefore, given a campaign, its sub-campaigns differ from each other based on the user-country and device. Moreover, there are three types of device in total: desktop, mobile and tablet.

In the early phases of this work, we designed static aggregations. As a matter of fact, the number of campaigns we were optimizing at that time was manageable enough to define ad hoc aggregation strategies for each campaign. More specifically, we designed the following aggregation granularity levels, from best (i.e., finer) to worst (i.e., coarser):

- **Sub-campaign level**: each sub-campaign is considered alone, thus each aggregation is composed by a single sub-campaign (i.e., singleton aggregation). This is applied whenever there are enough observations generated by each sub-campaign. This is the ideal aggregation granularity, as we are able to accurately estimate the *costs* and *number of clicks* functions for each sub-campaign.

- **User-country level**: all sub-campaigns sharing the same user-country are aggregated together. Note that, there are always three sub-campaigns sharing the same user-country, one for each device.

- **Foreign and device level**: each sub-campaign having the user-country equal to the hote-country (i.e., inhouse) is considered alone; on the other hand, all sub-campaigns having a user-country different from the hote-country (i.e, foreign) and the same device are aggregated together.

- **Foreign/inhouse level**: all inhouse and foreign sub-campaigns are aggregated together, respectively.

However, based on the experience of AdsHotel, we came to the conclusion that the only acceptable aggregation granularity levels were the first two: *sub-campaign level* and *user-country level*. The rationale behind this decision is that customers from different countries behave too differently from each other. Therefore, joining data at a coarser grain would not have led to good function estimates anyway. Furthermore, the static approach became prohibitive as soon

Algorithm 3 *DynamicCountryAggregation*($\mathcal{C}$)

Require: Campaign $\mathcal{C} = \{C_1, \ldots, C_N\}$

1: $S_P = \varnothing, S_N = \varnothing$
2: **for** $\mathcal{C}_{c_u} \in \mathcal{C}$ **do**
3: $A = \varnothing$
4: **for** $C \in \mathcal{C}_{c_u}$ **do**
5: **if** $v(\{C\}) = 1$ **then**
6: $S_P = S_P \cup \{C\}$
7: **else**
8: $A = A \cup \{C\}$
9: **end if**
10: **end for**
11: **if** $v(A) = 1$ **then**
12: $S_P = S_P \cup A$
13: **else**
14: $S_N = S_N \cup A$
15: **end if**
16: **end for**
17: $S = \{S_P, S_N\}$
18: **return** S, S_P, S_N

as the number of campaigns increased. Indeed, manually deciding the aggregation granularity of every campaign does not scale easily w.r.t. the number of campaign.

6.2.1 Dynamic Country Aggregation

To address the issues explained above, we designed an algorithm able to perform aggregations at a dynamic granularity level. The idea is that, given a user-country, we individually check each of its sub-campaigns through the value function v to see whether or not they have enough data on their own to accurately estimate the *costs* and *number of clicks* functions. A singleton aggregation is assigned to each sub-campaign passing this test. The ones failing it are aggregated together instead. Note that, it is possible to find the optimal aggregation in those few steps only because we only have three sub-campaigns for each user-country: one for each device type.

Here, we present the *DynamicCountryAggregation*($\mathcal{C}$) algorithm, whose pseudocode is provided in Algorithm 3. Given a campaign $\mathcal{C}$, this algorithm is in charge of returning the best partition S of its sub-campaigns. The algorithm divides the aggregations within that partition into two categories: those that pass the test and those that fail. The reason behind this further task is the following: AdsHotel wants us to provide a bid suggestion for a given sub-campaign only if we are able to build good function estimates for it.

The first step of the algorithm is to initialize S_P and S_N as empty sets (Line 1). These two sets, S_P and S_N, contain the partition's aggregations passing the test and the ones that do not, respectively. S_P and S_N compose a partition of S. During the execution of the algorithm, S_P and S_N will be expanded with

the various optimal aggregations. Now, we need to consider one user-country at a time, as aggregations are allowed only between sub-campaigns belonging to the same user-country. Thus, we loop through each subset $\mathcal{C}_{c_u}$ of sub-campaigns having user-country c_u (Lines 2-16). Note that, given a user-country, there can be only one non-singleton aggregation. This is due to the fact that there are only three sub-campaigns sharing the same user-country (i.e., one for each device type). Therefore, we initialize only one non-singleton aggregation set A as the empty set (Line 3). Now, for each sub-campaign $C \in \mathcal{C}_{c_u}$, we perform the test $v(\{C\})$ for the singleton aggregation $\{C\}$. Then, if the test indicates that $\{C\}$ has enough data to be able to estimate good *costs* and *number of clicks* functions, $\{C\}$ is added to S_P; otherwise, $\{C\}$ is added to A (Lines 4-10). After testing each sub-campaign singularly, aggregation A will contain all sub-campaigns that did not pass the test. Thus, the next step is to test the aggregation A. If the test is successful, A is added to S_P; otherwise, A is added to S_N (Lines 11-15). Next, we define the best partition S composed by all the aggregations contained in S_P and S_N (Line 17). At last, we return S, S_P and S_N (Line 18).

In Figure 6.1, we show all the possible aggregation instances that can happen during the *DynamicCountryAggregation* algorithm. As we know, given a user-country, we only have three sub-campaigns, one for each device type. We represent in green all sub-campaigns passing the test and in red all the failing ones. In instance 1, all sub-campaigns pass the test, and thus we add each singleton aggregation to S_P. As a result, in the optimization algorithm we are able to suggest three different bids. In instance 2, the tablet sub-campaign is the only one not passing the test, and thus we cannot merge it with any other sub-campaign belonging to the same user country. Only the *desktop* and *mobile* singleton aggregations will be added to S_P. In instance 3.1, only the desktop sub-campaign passes the test. Thus, we aggregate the mobile and tablet sub-campaigns. The resulting aggregation *mobile,tablet* passes the test. Hence, we add *desktop* and *mobile,tablet* to S_P. As a result, we will be able to suggest two bid values: one for the singleton aggregation *desktop* and one for *(mobile, tablet)*. Instead, in instance 3.2, the aggregation *mobile,desktop* fails the test. Thus, only the singleton aggregation *desktop* will be added to S_P, and only one bid will be suggested. In instance 4.1, every sub-campaign fails the test. The formed aggregation *(desktop,mobile,tablet)* passes the test, and thus will be added to S_P. As a result, the same bid will be suggested for all sub-campaigns. On the other hand, in instance 4.2, the resulting aggregation fails the test. Unfortunately, no aggregation will be added to S_P, and no bid will be suggested. Note that, in each instance, the device types are specified just for clarity, but they can be interchanged freely. For example, take instance 2: the sub-campaign failing the test can be any among the three, but we chose *tablet* just for clarity.

6.2.2 Singleton Aggregation

Unfortunately, due to requirements imposed by AdsHotel in later stages of the work, *DynamicCountryAggregation*($\mathcal{C}$) could not be used on the real system. These requirements regard the exploration constraints described in Section 5.2.4.

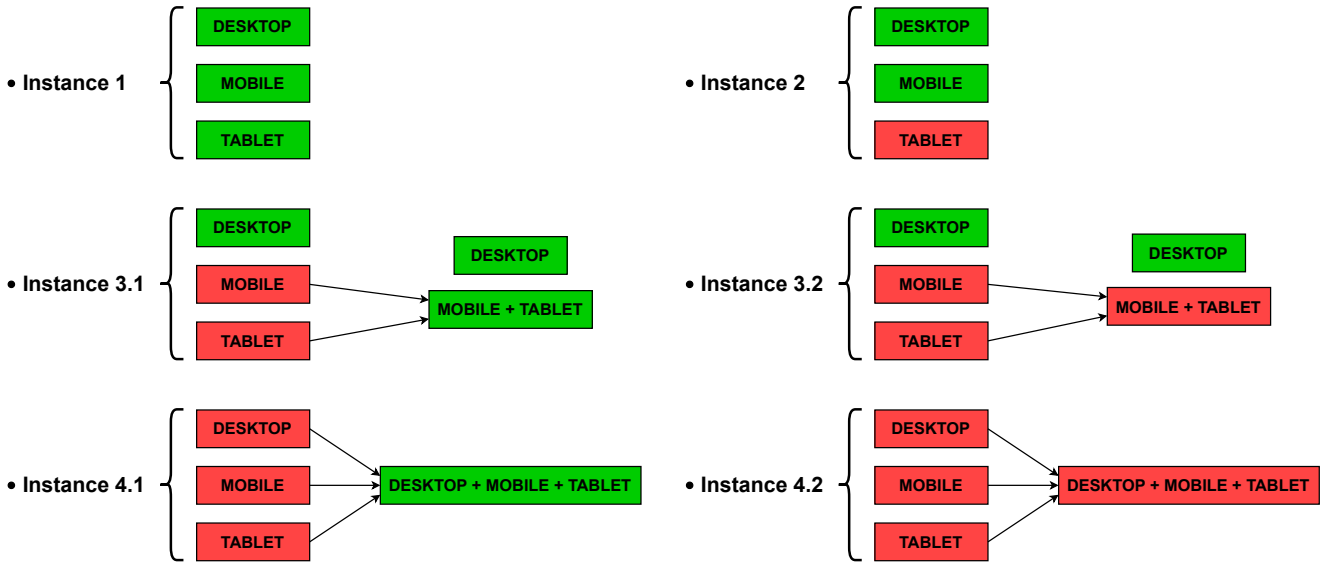

Figure 6.1: Possible DynamicCountryAggregation instances.

Algorithm 4 *SingletonAggregation*($\mathcal{C}$)

Require: Campaign $\mathcal{C} = \{C_1, \ldots, C_N\}$

1: $S_P = \varnothing, S_N = \varnothing$
2: **for** $C \in \mathcal{C}$ **do**
3: **if** $v(\{C\}) = 1$ **then**
4: $S_P = S_P \cup \{C\}$
5: **else**
6: $S_N = S_N \cup \{C\}$
7: **end if**
8: **end for**
9: $S = \{S_P, S_N\}$
10: **return** S, S_P, S_N

Conversely, we present the *SingletonAggregation*($\mathcal{C}$) algorithm, whose pseudo-code is provided in Algorithm 4 which was applied instead. Given a campaign $\mathcal{C}$, this algorithm performs the test on every sub-campaign and, based on the results, assigns them to one of the two sets S_P, and S_N. In this way, we are able to deliver a bid suggestion just for sub-campaigns passing the test. Note that, due to the fact that we only allow singleton aggregations, the best partition S is always the trivial partition.

The first step of the algorithm is to initialize S_P and S_N as empty sets (Line 1). These two sets, S_P and S_N, contain the singleton aggregations passing the test and the ones that do not, respectively. S_P and S_N compose the trivial partition of S. Now, for each sub-campaign $C \in \mathcal{C}$, we perform the test $v(\{C\})$ for the singleton aggregation $\{C\}$. Then, if the test indicates that $\{C\}$ has enough data to be able to estimate good *costs* and *number of clicks* functions, $\{C\}$ is added to S_P; otherwise, $\{C\}$ is added to S_N (Lines 2-8). Next, we define the best partition S composed by all the aggregations contained in S_P and S_N

	SingletonAggregation($\mathcal{C}$)	*DynamicCountryAggregation*($\mathcal{C}$)
Accepted	163	297
Rejected	18618	18484
Acceptance %	0.867%	1.58%

Table 6.1: Performance comparison between SingletonAggregation($\mathcal{C}$) and DynamicCountryAggregation($\mathcal{C}$) algorithms using the Data-Driven test.

(Line 9). At last, we return S, S_P and S_N (Line 10).

6.2.3 Testing Aggregations

In this section, we present the *Data-Driven* test which checks whether or not an aggregation has enough data to compute accurate estimates. It consists of directly evaluating the data before actually performing any estimation. Specifically, given an aggregation, we check its observations to see whether or not there are at least two unique bid values, each one with at least seven non-zero *costs* and *number of clicks* samples. The idea is that, to have the most simple form of estimation of a function $f(x)$ (i.e., a line), we need at least two points, each related to a different x value. Moreover, we required to have a minimum of 7 non-zero samples corresponding to the same x, meaning that the bid has been tested at least for a week.

6.3 Experimental Evaluation

We run the proposed *DynamicCountryAggregation*($\mathcal{C}$) and *SingletonAggregation*($\mathcal{C}$) algorithms on the dataset previously described in Chapter 2 and compare their results. Precisely, we consider a dataset composed of observations generated in the last $w = 60$ days, specifically from 19 August 2020 to 19 October 2021. The day we ran the experiments, 20 October 2021, there were 535 active campaigns with an overall number of sub-campaigns equal to $18,781$.

The results of the experiments are reported in Table 6.1. In the experiment run with the *SingletonAggregation*($\mathcal{C}$) algorithm, 163 sub-campaigns passed the test, while $18,618$ did not. Thus, approximately 0.867% of the total sub-campaigns passed the test. Conversely, the *DynamicCountryAggregation*($\mathcal{C}$) algorithm got the following results: 297 sub-campaigns passed the test, while $18,484$ did not. Therefore, out of the total number of sub-campaigns, 1.58% passed the test. This quantity increased by 82.2% compared to the first experiment. Given these results, we can say that the *DynamicCountryAggregation*($\mathcal{C}$) algorithm would have been a much better alternative to the *SingletonAggregation*($\mathcal{C}$).

Now we show a real aggregation example that took place in the second experiment run with the *DynamicCountryAggregation*($\mathcal{C}$) algorithm. Precisely, we display the details of the Data-Driven test performed on each aggregation and the corresponding GP estimations. The aggregation instance happens within

	Unique Bid	Costs $\neq 0$	Clicks $\neq 0$	Test Result
desktop	0.60	0	0	failed
	0.61	0	0	
	0.65	0	0	
mobile	0.42	23	23	failed
	0.56	4	4	
tablet	0.48	5	5	failed
	0.64	1	1	

Table 6.2: Details about the Data-Driven test run on each singular sub-campaign. Values inside the Costs $\neq 0$ and Clicks $\neq 0$ columns refer to the number of non-zero costs and clicks observations, respectively.

campaign *(hotel, google)*, over the three sub-campaigns targeting British customers.[2] Specifically, the campaigns are:

- *(hotel, google, united kingdom, desktop)*;
- *(hotel, google, united kingdom, mobile)*;
- *(hotel, google, united kingdom, tablet)*.

First, we perform the Data-Driven test on every sub-campaign. However, none of these exceed the thresholds defined by the test. Details are shown in Table 6.2. We show the GPs of each sub-campaign in Figure 6.2. Next, we aggregate the failed sub-campaigns and perform the test again. This time the test passes. Details are displayed in Table 6.3. We show the GPs of the aggregated sub-campaigns in Figure 6.3. This example clearly shows the advantages of the *DynamicCountryAggregation* algorithm compared to *SingletonAggregation*. Indeed, by using *SingletonAggregation*, we would have discarded every sub-campaign, providing zero bid suggestions. Instead, through *DynamicCountryAggregation*, we managed to obtain better GP estimates both from the point of view of the accuracy and of shape. Moreover, we managed to provide a unique bid suggestion for all these sub-campaigns, rather than none at all.

[2] Due to NDAs, we cannot disclose the actual name of the hotel as well as the actual cost and click values of the GPs. Instead we opted to display vaules which are proportional to the real numbers.

	Unique Bid	Costs $\neq 0$	Clicks $\neq 0$	Test result
desktop-mobile-tablet	0.42	1	1	passed
	0.48	14	14	
	0.60	10	10	
	0.61	1	1	
	0.64	4	4	
	0.65	1	1	

Table 6.3: Details about the Data-Driven test run on the aggregated sub-campaigns. Values inside the Costs $\neq 0$ and Clicks $\neq 0$ columns refer to the number of non-zero costs and clicks observations, respectively.

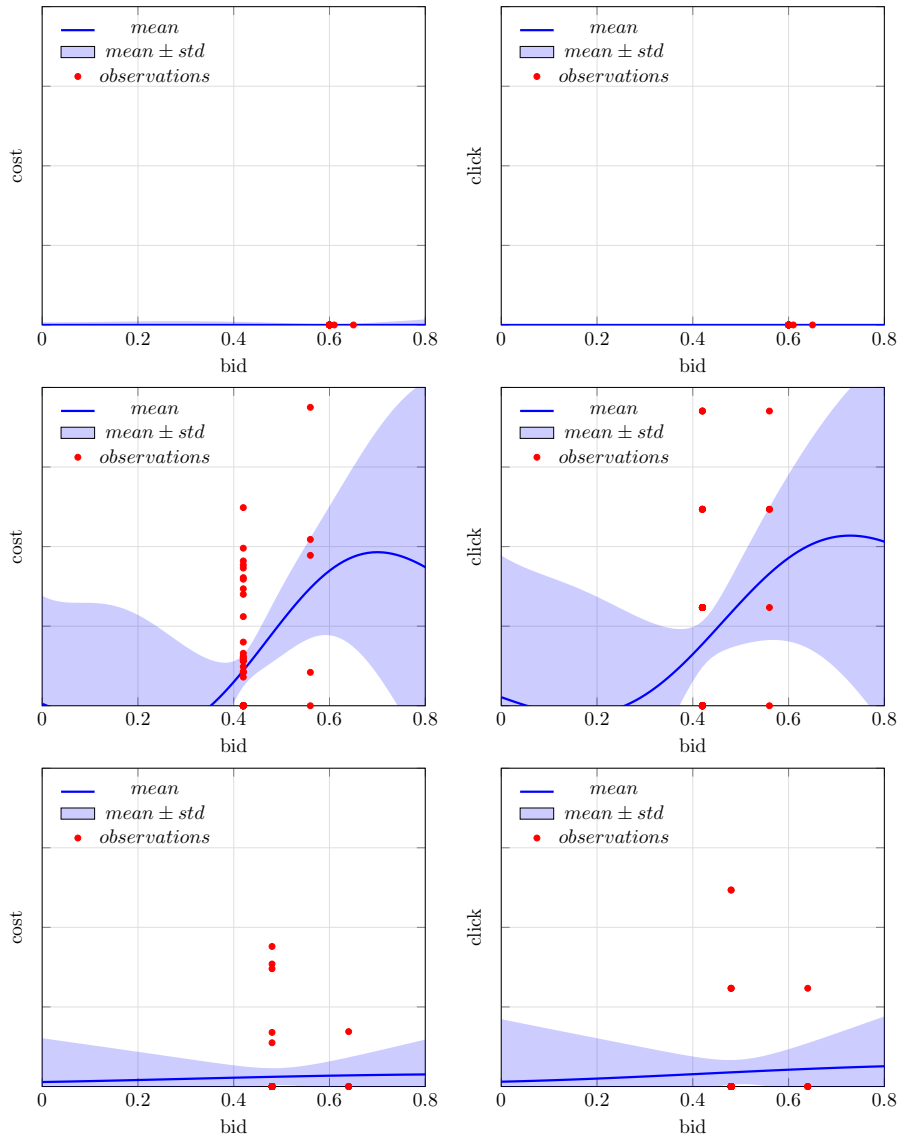

Figure 6.2: Cost and Click GPs of each sub-campaign in (hotel, google, united kingdom). (top left) Cost GP of the desktop sub-campaign. (top right) Click GP of the desktop sub-campaign. (center left) Cost GP of the mobile sub-campaign. (center right) Click GP of the mobile sub-campaign. (bottom left) Cost GP of the tablet sub-campaign. (bottom right) Click GP of the tablet sub-campaign.

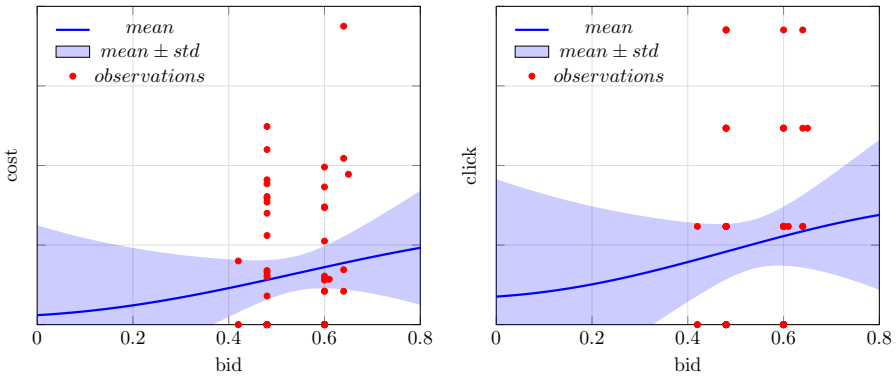

Figure 6.3: Cost and Click GPs of the aggregated sub-campaigns in (hotel, google, united kingdom). (left) Cost GP of the aggregated sub-campaigns. (right) Click GP of the aggregated sub-campaigns.

Chapter 7

Country Exploration

Country exploration consists of expanding a running campaign by opening promising user-countries that are not included in the campaign yet. Indeed, even though we would like to expand a campaign at a finer level (i.e., sub-campaigns), AdsHotel's Platform allows such action only at user-country level, meaning that when we open a country all sub-campaigns associated with that user-country are added to the campaign. This applies both to Google and Tripadvisor campaigns, as explained in Sections 2.3.1 and 2.3.2, respectively. The goal of this task is to not let hotels miss out on any good opportunity that might generate important revenue. The challenge is given by the lack of information. As a matter of fact, the sub-campaigns belonging to closed countries do not generate samples. This means that we do not have a direct way of evaluating how good the performance of a closed country could be once we open it. The first solution that comes to mind is to blindly open every possible country and collect observations. However, due to economical reasons, this cannot be applied in a real-world scenario. Thus, we first need to collect information about a given country and then decide if it is worth opening. To overcome the lack of information, we evaluate how the sub-campaigns of a closed country are performing on other similar hotels. Then, we use two different, but complementing, approaches to provide opening suggestions:

1. we use a ranking system to identify the most promising country to open among the currently closed ones;

2. we study their performance trend through change detection methods.

This chapter is structured as follows. In Section 7.1 we provide a formal description of the problem. In Section 7.2 we describe our approach and algorithms. Finally, in Section 7.3 we present the experimental results.

7.1 Problem Formulation

Given a finite time horizon of $T \in \mathbb{N}$ days, we define an advertising campaign $\mathcal{C}_t = \{A_t, B_t\}$ at day $t \in \{1, \dots, T\}$, where A_t and B_t are the set of its open and closed user-countries, respectively. For each day $t \in \{1, \dots, T\}$, our goal is

to specify the closed user-country $b \in B_{t-1}$ to be opened, such that the overall cumulative revenue of the expanded campaign $\mathcal{C}_t$ is maximized, while keeping the overall ROI above a fixed value $\lambda^* \in \mathbb{R}^+$. At a given day t, the problem is modeled as follows:

$$\underset{b \in B_{t-1}}{\text{argmax}} \quad r(b) + \sum_{a \in A_{t-1}} r(a), \tag{7.1a}$$

$$\text{s.t.} \frac{r(b) + \sum_{a \in A_{t-1}} r(a)}{c(b) + \sum_{a \in A_{t-1}} c(a)} \geq \lambda^*, \tag{7.1b}$$

where $r(a)$ and $c(a)$ are the expected revenue and the expected cost given the open user-country a, respectively. Moreover, Constraint (7.1b) is the ROI constraint, forcing the revenue to be at least λ^* times the incurred costs. Note that if the expected revenue and cost functions were known, the optimal user-country $b \in B_{t-1}$ to be opened could be found by solving the Equations 7.1 in closed form. However, $r(b)$ is unknown to the advertiser, therefore we need to define a proxy to understand which countries are the most promising ones to be added to the advertising campaign.

From now on, we assume that the an action to expand a running advertising campaign consists in the addition of a sub-campaigns targeting a previously closed user-country.

7.2 Proposed Method

As already outlined, we propose two methods for opening a user-country: *Global Rankings* and *Performance Trend Monitoring*. The former is based on a ranking system, while the latter is based on change detection techniques. In principle, these two methods could be used separately since they serve two different purposes. Indeed, while the role of the ranking is to find out the most promising user-country from the point of view of the performance, the role of the second one is to give further information on the performance trend. In this way, AdsHotel has more information to analyze our suggestions. After our algorithm proposes the opening of a user-country, for a certain campaign $(hotel, metasearch)$, AdsHotel is in charge of opening that country for that specific campaign. However, due to contract bindings, this can be done only with the authorization of the Hotel Manager. Anyway, supposing AdsHotel opens the suggested user-country, the multipliers of those sub-campaigns are set to to a default setting (the so called *low visibility*). The idea is that, as always, if we are not sure to satisfy minimum ROI constraints, we do not want to risk spending too much campaign budget. Our safe bid optimization algorithm, explained in Chapter 5, cannot provide bid suggestions for such sub-campaigns as there are no observations available yet. Therefore, by setting the sub-campaign multipliers to the default strategy, we collect useful observations with the goal to include the campaign in the optimization procedure.

Note that an hotel does not generate observations regarding its closed user-countries, thus we cannot directly evaluate their performances to understand

Algorithm 5 Global Rankings

Require: H set of hotels with at least one active campaign

1: query the DB to get the global rankings R
2: normalize R
3: apply continent weighting to R
4: **for** $h \in H$ **do**
5: get hotel-country c_h and set M_h of metasearches for hotel h
6: get ranking R_{c_h} of the hotel-country c_h
7: sort R_{c_h} by descending order
8: **for** $m \in M_h$ **do**
9: retrieve set O_m of open user-countries for metasearch m
10: select the first user-country c_u in the global ranking R_{c_h} s.t. $c_u \notin O_m$
11: insert opening suggestion for c_u on the DB
12: **end for**
13: **end for**

which one is the most promising. We circumvent this problem using the observations produced by other hotels in which those user-countries are open. More specifically, we use observations generated by hotels situated in the same country of the hotel for which we need to suggest the opening. This approach requires to hold the assumption that all hotels located in the same hote-country have a similar influence over the customers.

As a final remark, we highlight that in both *Global Rankings* and *Performance Trend Monitoring*, the performance measure is based on impressions. The reason behind this decision is that, under COVID-19 restrictions, conversions are extremely rare, thus very unreliable. Therefore, also consulting with the expert of the field, we narrow our attention on the first layer of the funnel model (i.e., impressions).

7.2.1 Global Rankings

In this section we present the *Global Rankings* algorithm, whose pseudo-code is provided in Algorithm 5. Given the set H of hotels with at least one active campaign, the algorithm proposes, for each campaign of every hotel $h \in H$, the first closed user-country c_u in the ranking R_{c_h}.

At the beginning (Lines 1-3), we build one global ranking for each existing hotel-country. Initially, each global ranking contains a descending ordering of user-countries in terms of total impressions received over the last w days by all hotels located in the given hotel-country, where w is the sliding window presented in Section 5.2.3. This is done in a one-shot manner by performing one query on the DB (Line 1). Such rankings are built using only cleaned observations, as explained in Section 5.2.1. We normalize every ranking by dividing the total impressions by the number of hotels that contributed those impressions (Line 2) to take into account the different level of adoption of the Adshotel system in different countries. It is important to note that not all user-countries should have the same importance in a global ranking. This is due to the fact that some impressions may be more important than others by leading to more conversions

Continent	Weight
Europe	0.7
North America	0.7
Asia	0.4
Australia	0.4
South America	0.25
Africa	0.25
Antarctica	0.0

Table 7.1: Continent weights.

Algorithm 6 Performance Trend Monitoring

Require: set H of hotels with at least one active campaign

1: **for** $h \in H$ **do**
2: get hotel-country c_h and set M_h of metasearches for hotel h
3: retrieve from memory the set of statistics S_{c_h}
4: $S^*_{c_h} = CDT\,(c_h, S_{c_h})$
5: store $S^*_{c_h}$ on memory
6: retrieve the set Y of last drift types from $S^*_{c_h}$
7: **for** $m \in M_h$ **do**
8: retrieve set C_m of closed user-countries for metasearch m
9: insert Y_{C_m} on the DB
10: **end for**
11: **end for**

(i.e., different *conversion rates*). With the help of AdsHotel we empirically set a weight scheme for the impressions coming from different continents (Line 3). First, user-countries in the same continent of the hotel weight more than all the others. For this reason, the weight of the hotel's continent is always set to 1.0. Instead, the other continents are weighted according to Table 7.1.

The remaining steps of the algorithm (Lines 4-13) consist in selecting the single county on which suggest an opening based on the previously defined ranking. For each hotel having at least one active campaign, we retrieve its hotel-country and its set of metasearches (Line 5). Next, we get the global ranking corresponding to that hotel-country (Line 6-7). We would like to suggest one user-country to each currently ongoing campaign that, as we know, is identified by the tuple $(hotel, metasearch)$. Thus, for each hotel and for each one of his running campaign (i.e., metasearch), we retrieve the set of currently open user-countries (Line 9). Then, we select the first user-country in the ranking that is not open yet for that particular campaign (Line 10). Finally, we write on the DB our opening suggestion (Line 11).

7.2.2 Performance Trend Monitoring

In this section we present the *Performance Trend Monitoring* algorithm, whose pseudo-code is provided in Algorithm 6. Given the set H of hotels with at least one active campaign, the algorithm monitors, for each hotel $h \in H$,

the performance trend of every user-country through a CDT procedure and reports, for each campaign of hotel h, the latest concept drift type of each closed user-country. Given a pair *(hote-country, user-country)*, we monitor the time series of daily total impressions generated, over the last 6 months and for the given user-country, by all the hotels located in that hotel-country. However, as in the ranking case, considering only the total daily impressions is not a good idea. Indeed, by not taking into account also the number of hotels that generated those daily impressions, we might detect concept drifts caused by a sudden change in the number of contributing hotels. For this reason, the time series of daily total impressions are normalized by the number of daily hotels generating the impressions. Another important aspect of the algorithm is that it must be executed every time we receive new observations. Indeed, adding the new data to the old one can lead to a new concept drift detection. In our scenario we receive new samples every day, thus the algorithm must be executed daily after we receive all the new observations. However, running a CDT procedure every day for each pair *(hotel-country, user-country)* is computationally intensive if we take into account the whole history of data. To lighten this complexity, at the end of each execution, we store into a memory file the progress of the monitoring procedure (i.e., statistics) for each pair *(hotel-country, user-country)*. In this way, only the minimum amount of observations are taken into account by the monitoring procedure. .

At first, the steps in Lines 2-10 are repeated for each hotel with at least one active campaign. First, we retrieve the hotel-country c_h and the set of metasearches M_h for the current hotel h (Line 2). Then, we access the the set S_{c_h} of statistics corresponding to each user-country for the current hotel-country (Line 3).[1] Next, we execute the $CDT\,(c_h, S_{c_h})$ subroutine to update the set S_{c_h} with the new daily observations; after that, we delete the old statistics S_{c_h} and insert the new ones $S^*_{c_h}$ in the statistic file (Lines 4-5). Depending on whether we want to perform a CUSUM or an ADWIN test, the $CDT\,(c_h, S_{c_h})$ subroutine can by any of the following two: $CDT\text{-}CUSUM\,(c_h, S_{c_h})$ or $CDT\text{-}ADWIN\,(c_h, S_{c_h})$. The remaining steps of the algorithm (lines 6-10) are dedicated to the communication of the performance trend of each user-country. Specifically, AdsHotel is interested in the latest drift detected: they want to know whether the recent performance is increasing or decreasing, with respect to the past. For this reason, we characterize each drift in one of the following two categories: a *positive* drift occurs when a performance increase is detected, while a *negative* drift occurs when a performance decrease is detected. In line 6, we retrieve from $S^*_{c_h}$ the set Y of last drift types of each user-country. Then, for each running campaign (i.e., metasearch m) of hotel h, we retrieve the set C_m of currently closed user-countries and write their last drift types Y_{C_m} on the DB (lines 7-10). Of course, if no drift has been detected yet, for a given closed user-country, we do not provide any information relative to that user-country.

CDT-CUSUM Subroutine

[1] We will specify the nature of the statistics used by each algorithm later on in the discussion.

Algorithm 7 *CDT-CUSUM* (c_h, S_{c_h}) subroutine

Require: the hotel-country c_h, current set S_{c_h} of statistics, default start date d_{start}, current date d_{today}

1: **if** S_{c_h} not empty **then**
2: retrieve the last update date d_{update} for S_{c_h}
3: $d_{start} = d_{update}$
4: **end if**
5: query the DB to get observations O occurred in $[d_{start}, d_{today}]$
6: normalize O
7: retrieve the set C_u of user-countries contained in O
8: **for** $c_u \in C_u$ **do**
9: retrieve the statistic $s_{c_u} \in S_{c_h}$ and the observations $O_{c_u} \in O$
10: perform *CUSUM* (O_{c_u}, s_{c_u}) test to get the updated statistic $s^*_{c_u}$
11: **end for**
12: define the set $S^*_{c_h}$ by updating the old S_{c_h} with the new $s^*_{c_u}$
13: set $d_{update} = d_{today}$ for set $S^*_{c_h}$
14: **return** $S^*_{c_h}$

In this Section we present the *CDT-CUSUM* (c_h, S_{c_h}) subroutine, whose pseudo-code is provided in Algorithm 7. This subroutine is in charge of retrieving the new daily observations and returning the updated statistics of every user-country. The default start date d_{start} indicates the day from which we start collecting observations, if no statistics have been saved on file yet (i.e., empty statistics). As already introduced, this default date is set to 6 months before the current day.

The first step of the subroutine should be querying the DB to retrieve the new observations. However, the interval $[d_{start}, d_{today}]$ for the retrieval must be set beforehand. In fact, if the statistics S_{c_h} have not been computed yet, the d_{start} date should be the default one; instead, if the statistics have already been computed in the past, the d_{start} date should be the one related to the last update of the statistics. Of course, the queries observations are first cleaned as explained in Subsection 5.2.1. All these steps are performed at Lines 1-5 of the algorithm. In line 6, we perform the normalization, dividing each daily total impressions by the number of daily hotels that generated those impressions. Then, we retrieve the set C_u containing every user-country from which we received observations in the interval $[d_{start}, d_{today}]$ (Line 7). The statistics of these user-countries are the ones that need to be updated. Thus, for each user-country $c_u \in C_u$, we perform the following steps: retrieve its statistic s_{c_u}, retrieve its observations O_{c_u}, and finally perform the *CUSUM* (O_{c_u}, s_{c_u}) test to get its updated statistic $s^*_{c_u}$ (Lines 8-11). Then, we define the set $S^*_{c_h}$ of updated statistics by combining the old ones, for which we did not receive any new observation, with the new ones that have been just updated by the *CUSUM* (O_{c_u}, s_{c_u}) test (Line 12). Next, the date of the last update d_{update} for $S^*_{c_h}$ is set to the current date (Line 13). At last, we return the set of updated statistics $S^*_{c_h}$ (Line 14).

CDT-ADWIN Procedure

Algorithm 8 $CDT\text{-}ADWIN\,(c_h, S_{c_h})$ subroutine

Require: the hotel-country c_h, current set S_{c_h} of statistics, default start date d_{start}, current date d_{today}

1: **if** S_{c_h} not empty **then**
2: retrieve the last update date d_{update} for S_{c_h}
3: $d_{start} = d_{update}$
4: **end if**
5: query the DB to get the set C_u of user-countries for which some observations occurred in $[d_{start}, d_{today}]$
6: **for** $c_u \in C_u$ **do**
7: retrieve the statistic $s_{c_u} \in S_{c_h}$
8: **if** s_{c_u} not empty **then**
9: retrieve the date of the last detection $d_{detection}$ from from s_{c_u}
10: **else**
11: $d_{detection} = d_{start}$
12: **end if**
13: query the DB to get observations O_{c_u} occurred in $[d_{detection}, d_{today}]$
14: normalize O_{c_u}
15: perform $ADWIN\,(O_{c_u}, s_{c_u})$ test to get the updated statistic $s^*_{c_u}$
16: **end for**
17: define the set $S^*_{c_h}$ by updating the old S_{c_h} with the new $s^*_{c_u}$
18: set $d_{update} = d_{today}$ for set $S^*_{c_h}$
19: **return** $S^*_{c_h}$

In this Section we present the $CDT\text{-}ADWIN\,(c_h, S_{c_h})$ subroutine, whose pseudo-code is provided in Algorithm 8. This subroutine is in charge of retrieving the new daily observations and returning the updated statistics of every user-country. The default start date d_{start} indicates the day from which we start collecting observations, if no statistics have been saved on file yet (i.e., empty statistics). As already introduced, this default date is set to 6 months before the current day.

The first steps are dedicated to the retrieval of the set C_u of user-countries, from which we received new observations in the interval $[d_{start}, d_{today}]$ (Lines 1-5). If the statistics S_{c_h} have not been computed yet, the d_{start} date should be the default one; instead, if the statistics have already been computed in the past, the d_{start} date should be the one related to the last update of the statistics. Now, for each user-country $c_u \in C_u$, we do the following steps to update its statistic s_{c_u} (Lines 7-15). First, we retrieve the current statistic $s_{c_u} \in S_{c_h}$ for the given user-country (Line 7). If s_{c_u} is not empty (i.e., the statistic is stored in memory for that specific user-country), we retrieve the date of the last detection $d_{detection}$ for that user-country. On the other hand, if s_{c_u} is empty (i.e., the statistic is not stored in memory yet for that specific user-country), the date of the last detection $d_{detection}$ for that user-country is set to the default one (i.e., d_{start}) (Lines 8-12). Now that we have the interval $[d_{detection}, d_{today}]$, we query the DB to get the observations O_{c_u} occurred in that time interval for that user-country (Line 13). In line 14, we perform the normalization, dividing each daily total impressions by the number of daily hotels that generated those impressions.

Algorithm 9 $CUSUM\,(O_{c_u}, s_{c_u})$ procedure

Require: observations O_{c_u}, current statistic s_{c_u}, days m

1: retrieve $\bar{\mu}_0$, $\bar{\mu}$, g^+_{t-1}, g^-_{t-1}, n and y from s_{c_u}
2: retrieve T and r from O_{c_u}
3: **for** $t \in \{1, \ldots, T\}$ **do**
4: $\quad n = n + 1$
5: $\quad \bar{\mu} = \frac{\bar{\mu}(n-1)+r_t}{n}$
6: $\quad$ **if** $n \leq m$ **then**
7: $\quad\quad \bar{\mu}_0 = \bar{\mu}$
8: $\quad$ **else**
9: $\quad\quad h = \alpha\,\bar{\mu}_0$
10: $\quad\quad \epsilon = \beta\,\bar{\mu}_0$
11: $\quad\quad g^+_t = \max\left\{0, g^+_{t-1} + r_t - \bar{\mu}_0 - \epsilon\right\}$
12: $\quad\quad g^-_t = \max\left\{0, g^-_{t-1} + \bar{\mu}_0 - r_t - \epsilon\right\}$
13: $\quad\quad$ **if** $g^+_t > h$ **then**
14: $\quad\quad\quad y = +1$
15: $\quad\quad\quad n = 0, \bar{\mu} = 0, g^+_t = 0, g^-_t = 0$
16: $\quad\quad$ **else if** $g^-_t > h$ **then**
17: $\quad\quad\quad y = -1$
18: $\quad\quad\quad n = 0, \bar{\mu} = 0, g^+_t = 0, g^-_t = 0$
19: $\quad\quad$ **end if**
20: $\quad$ **end if**
21: **end for**
22: update statistic $s^*_{c_u}$
23: **return** $s^*_{c_u}$

Next, we perform the $ADWIN\,(O_{c_u}, s_{c_u})$ test to get the updated statistic $s^*_{c_u}$ (Line 15). Then, we define the set $S^*_{c_h}$ of updated statistics by combining the old ones, for which we did not receive any new observation, with the new ones, that have been just updated by the $ADWIN\,(O_{c_u}, s_{c_u})$ test (Line 17). Next, the date of the last update d_{update} for $S^*_{c_h}$ is set to the current date (Line 18). At last, we return the set of updated statistics $S^*_{c_h}$ (Line 19).

CUSUM Algorithm

Here, we present the $CUSUM\,(O_{c_u}, s_{c_u})$ procedure, whose pseudo-code is provided in Algorithm 9. It is a version of the CUSUM algorithm, introduced by Hinkley in his work [26], tailored to our specific needs and presented here to the convenience of the reader. The goal of this algorithm is to update the statistic of a given user-country by using its new observations. A statistic of a given user-country s_{c_u} is composed by:

- c_h: the current hotel-country;
- c_u: the current user-country;
- n: the number of observations received since the last detection. Note that, each day we may receive zero or one observation;
- $\bar{\mu}_0$: the impression mean for the current concept;

- $\bar{\mu}$: the impression mean of the latest n observations;
- g^+: the positive CUSUM statistic. When the value of g^+ exceeds a certain threshold value h, a positive drift is detected;
- g^-: the negative CUSUM statistic. When the value of g^- exceeds a certain threshold value h, a negative drift is detected;
- y: the drift type of the latest detection;
- d_{update}: the date in which the statistic has been last updated. This date is equal for every statistic belonging to the same hote-country (i.e., $s_{u_c} \in S_{c_h}$).

By storing these values in memory, we completely avoid monitoring the whole history of observations: we just need to run the CUSUM procedure on the new samples.

The first step consists into retrieving the most important parameters from statistic s_{c_u} and from observations O_{c_u} (Lines 1-2). We retrieve $\bar{\mu}_0$, $\bar{\mu}$, g^+_{t-1}, g^-_{t-1}, n and y from the statistic s_{c_u}. If the statistic s_{c_u} is empty (i.e., no statistic relative to the pair (c_h, c_u) has been stored on file yet), each parameter is set with default value 0. Then, we retrieve the number of observations T and the vector r of normalized total daily impressions, from the observations O_{c_u}. Now that all the parameters have been retrieved, the heart of the CUSUM algorithm updates them by using the new samples (Lines 3-21). The following steps (4-21) are executed for each new sample r_t. First, we take into account that we have a new sample by updating n and the current impression mean $\bar{\mu}$ (Lines 4-5). Then, we check if $n \leq m$, where m indicates the number of observations, under stationary conditions, needed to have a good estimate of the concept mean $\bar{\mu}_0$. If the condition is false, it means our model $\bar{\mu}_0$ is not ready yet to be used for drift detection: it needs more samples to estimate accurately the concept mean. Thus we update $\bar{\mu}_0$ and we skip to the next iteration of the for loop (Line 7). On the other hand, if the conditions is true, it means our model $\bar{\mu}_0$ is a good estimate of the concept mean. Thus, we can go on with the drift detection steps (Lines 9-19). We define the threshold h and the parameter ϵ (Lines 9-10). The parameter ϵ should be set such that the minimum gap, provided by a change, is 3ϵ. The positive and negative CUSUM statistics are updated taking into account their previous values ϵ, $\bar{\mu}_0$ and r_t (Lines 11-12). More precisely: $g^+_t = \max\left\{0, g^+_{t-1} + r_t - \bar{\mu}_0 - \epsilon\right\}$ and $g^-_t = \max\left\{0, g^-_{t-1} + \bar{\mu}_0 - r_t - \epsilon\right\}$. The idea behind the g^+_t update is that its value increases if the observed sample r_t is greater than its expected value $\bar{\mu}_0$ plus an oscillation value ϵ, which is considered a normal oscillation for the current concept. The reversed idea is applied to the g^-_t update. Next, we check if the CUSUM statistics exceed the threshold h (Lines 13-19). Note that, by construction, they can only exceed h one at a time. If h is exceeded by g^+_t a positive drift is detected, we set the drift type $y = +1$, and we reset to 0 the remaining statistic parameters. On the other hand, if h is exceeded by g^-_t a negative drift is detected, we set the drift type $y = -1$, and we reset to 0 the remaining statistic parameters. After looping through all the new

Algorithm 10 $ADWIN\,(O_{c_u}, s_{c_u})$ procedure

Require: observations O_{c_u}, current statistic s_{c_u}

1: retrieve y from s_{c_u}
2: retrieve T and r from O_{c_u}
3: **for** $t \in \{1, \ldots, T\}$ **do**
4: append r_t to W
5: **for each** split of W into $W = [W_0, W_1]$ **do**
6: compute $\bar{\mu}_{W_0}$ mean of W_0
7: compute $\bar{\mu}_{W_1}$ mean of W_1
8: $\epsilon = \eta\,\bar{\mu}_{W_0}$
9: **if** $(\bar{\mu}_{W_1} - \bar{\mu}_{W_0}) \geq \epsilon$ **then**
10: candidate positive drift detected
11: **else if** $(\bar{\mu}_{W_0} - \bar{\mu}_{W_1}) \geq \epsilon$ **then**
12: candidate negative drift detected
13: **end if**
14: **end for**
15: **if** drift detected **then**
16: select the drift with maximum mean difference among the candidates
17: set drift type $y = +1$ or $y = -1$ accordingly
18: **end if**
19: **end for**
20: update statistics $s^*_{c_u}$
21: **return** $s^*_{c_u}$

samples, we update the old statistic s_{c_u} with the new parameters, obtaining $s^*_{c_u}$ (Line 22). At last, we return the updated statistic $s^*_{c_u}$ (Line 23).

ADWIN Algorithm

Here, we present the $ADWIN\,(O_{c_u}, s_{c_u})$ procedure, whose pseudo-code is provided in Algorithm 10. It is a version of the ADWIN algorithm, introduced by Bifet et al. [9], tailored to our specific needs and presented here to the convenience of the reader. The goal of this algorithm is to update the statistic of a given user-country by using its new observations. A statistic of a given user-country s_{c_u} is composed by:

- c_h: the current hotel-country;
- c_u: the current user-country;
- $d_{detection}$: refers to the date of the last drift detection for the current user-country.
- y: the drift type of the latest detection;
- d_{update}: the date in which the statistic has been last updated. This date is equal for every statistic belonging to the same hote-country (i.e., $s_{u_c} \in S_{c_h}$).

By storing these values in memory, we completely avoid monitoring the whole history of observations: we just need to run the ADWIN procedure for the samples generated after the last drift detection.

The first step consists into retrieving the last drift type y from statistic s_{c_u} (line 1). If the statistic s_{c_u} is empty (i.e., no statistic relative to the pair (c_h, c_u) has been stored on file yet), y is set to 0 by default. Then, we retrieve the number of observations T and the vector r of normalized total daily impressions, from the observations O_{c_u} (Line 2). Now, the ADWIN algorithm will find out whether or not a concept drift happened, and set y accordingly (Lines 3-19). First, we append the normalized daily total impressions r_t to the window W (Line 4). Then, for each possible split of W into two adjacent sub-windows W_0 and W_1, we compare the mean of each sub-window to see if there is enough evidence of a concept drift (Lines 5-14). In lines 6-7 we compute the means $\bar{\mu}_{W_0}$ and $\bar{\mu}_{W_1}$ of the two sub-windows. Next, we define the threshold ϵ (Line 8). If the mean of the second window is considerably higher than the mean of the first window (i.e., $(\bar{\mu}_{W_1} - \bar{\mu}_{W_0}) \geq \epsilon$), we detect a candidate positive drift. On the other hand, if the mean of the first window is considerably higher than the mean of the second window (i.e., $(\bar{\mu}_{W_0} - \bar{\mu}_{W_1}) \geq \epsilon$), we detect a candidate negative drift (Lines 9-13). Thus, after we tried every possible split, we have a set of drift candidates and we must choose one of them. If the candidate set is not empty, we select the drift having the highest mean difference among the candidates. Then, if the selected drift is positive we set $y = +1$, otherwise we set $y = -1$ (Lines 15-18). After looping through all the new samples, we update the old statistic s_{c_u} with the new parameters, obtaining $s^*_{c_u}$ (Line 20). At last, we return the updated statistic $s^*_{c_u}$ (Line 21).

7.3 Experimental Evaluation

In this chapter, we presented two country exploration methods, namely *Global Rankings* and *Performance Trend Monitoring*. Both approaches have been tested on active hotel advertising campaigns and, in this section, we show the obtained results for each of them.

7.3.1 Global Rankings

To understand how correct the opening suggestions provided by the *Global Rankings* are, we need to do the following steps. First, we should identify the user-countries that have been opened due to our suggestions. Then, for each of those user-country, we should evaluate the performance in terms of impressions, clicks, costs, conversions, and ROI. Unfortunately, we cannot evaluate the performance of any user-country that we suggested to open because our opening recommendations have not been applied by AdsHotel yet. Indeed, opening a user-country is a very delicate action to make, as AdsHotel's contract bindings with Hotels do not allow them to expand a campaign at their will. The Hotel Manager's authorization is needed to perform such action.

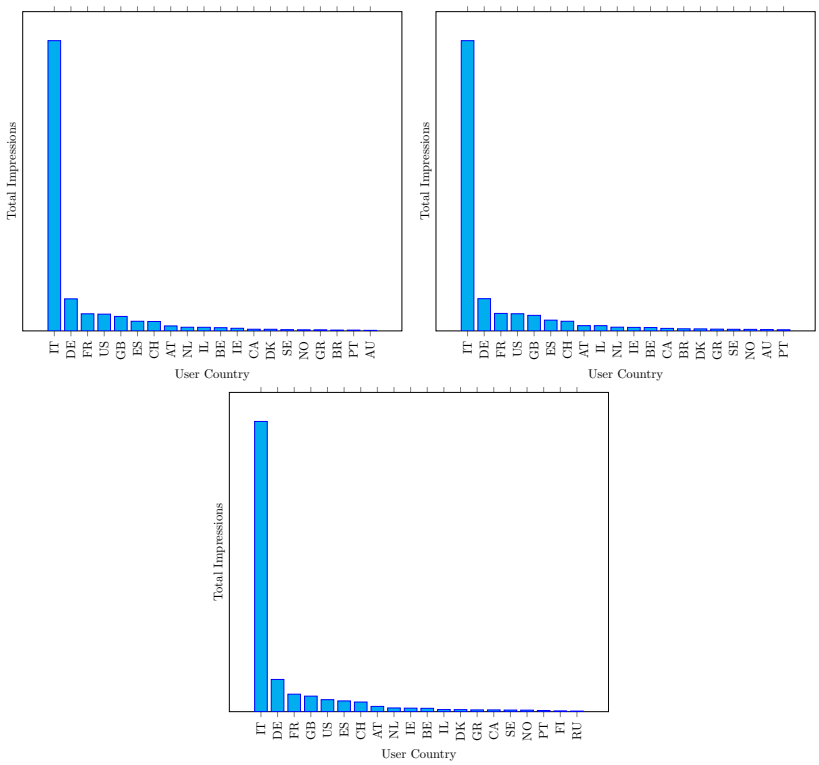

Figure 7.1: The three phases of Italian global ranking built with the total impressions of every user-country. (top left) Not yet normalized total impressions. (top right) Normalized total impressions. (bottom) Normalized and weighted total impressions.

For these reasons, the following experimental evaluations will consist of showing some of the rankings built by our algorithm to suggest user-country openings. As we previously explained, each time we need to suggest an opening, the rankings are built with observations generated over the last w days. More precisely, all the rankings shown in this Subsection are built with data generated between 19 August 2021 and 19 October 2021. Note that, for visibility reasons, only the top 20 user-countries of the ranking are shown in the plots. Moreover, due to NDAs with hotels, we are not allowed to disclose the actual impression amounts in the rankings.

In Figure 7.1, we show the three phases undergone by a global ranking in our algorithm. Specifically, we show the Italian global ranking, which is built using the total impressions generated by every Italian hotel, on each possible user-country. In the first phase, the global ranking is dictated by just the total impressions collected by each user-country. Then, in the second phase, each of them is normalized by the number of hotels contributing to the given user-country. Finally, the normalized total impressions are weighted according to the continent of the user-country. The latter is the global ranking that, given an Italian hotel, will be used to suggest the most promising closed user-country.

Instead, in Figure 7.2, we show the three phases undergone by the Italian global ranking, built with the total clicks generated by every Italian hotel on each

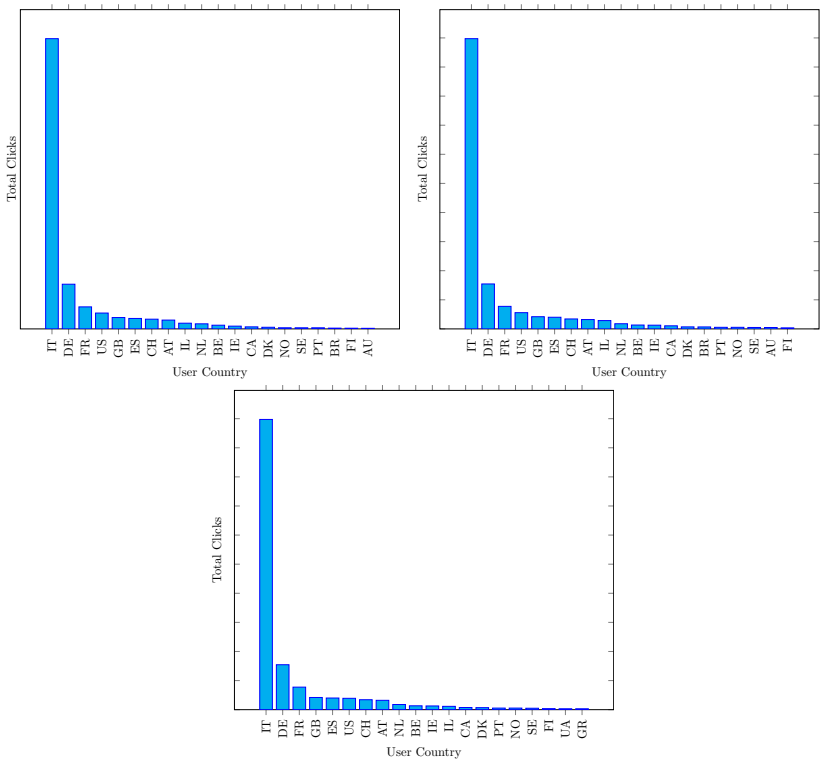

Figure 7.2: The three phases of Italian global ranking built with total clicks of every user-country. (top left) Not yet normalized total clicks. (top right) Normalized total clicks. (bottom) Normalized and weighted daily clicks.

possible user-country. The three phases are exactly the same as in the previous case. Italy is one of the few hotel-country for which we are able to build a global ranking based on clicks. In fact, AdsHotel works for a vast portion of Italian hotels, thus the amount of generated clicks is high enough to build the ranking.

In Figure 7.3, we show and compare the final phases of the Italian global ranking build with total impressions, total clicks, total conversions and total ROI. Specifically, we show that the top 20 user-countries are almost the same for the impressions and clicks rankings. This indicates that the number of impressions are still a good indicator for the performance of a user-country. On the other hand, we can see that the conversions and ROI rankings are very different from the impressions and clicks ones. This proves that, as soon as conversions are taken into account, we obtain very inaccurate and unreliable estimates.

In Figure 7.4, we show the global ranking of an American hotel-country. Specifically, we display the three phases undergone by the Honduran global ranking, built with the total impressions generated by every Honduran hotel on each possible user-country.

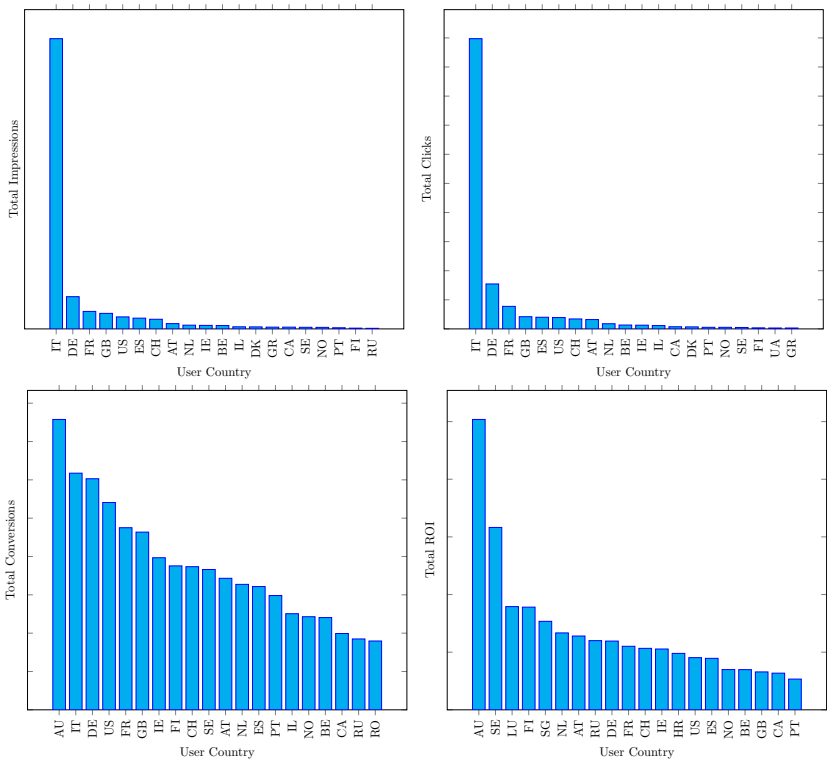

Figure 7.3: Comparing all types of global ranking for Italian hotels. (top left) Normalized and weighted total impressions. (top right) Normalized and weighted total clicks. (bottom left) Normalized and weighted total conversions. (bottom right) Weighted total ROI.

7.3.2 Performance Trend Monitoring

In Section 3.2.5, we explained why the ARL_0 and ARL_1 are two crucial performance measures for a Change Detection Test. The ARL_0 describes the expected amount of time between multiple false positives, under stationary conditions. On the other hand, the ARL_1 indicates the expected time delay for detecting an occurred change. In our scenario, there is absolutely zero feedback on our change detections: we cannot distinguish between false positives and true positives. For this reason, it is not possible to find a value of a threshold that satisfies a given ARL_0 or ARL_1 value. Due to this lack of *true labels*, we cannot even estimate the Detection Delay (DD), the False Positive Rate (FPR) and the False Negative Rate (FNR). Indeed, if we do not know exactly when a change happens, we cannot perform multiple simulations on finite sequences containing a change at a known location, in order to compute the DD, the FPR and the FNR.

For these reasons, we decided to use CUSUM and ADIWN as change detection tests for monitoring the performance trend of user-countries. Indeed, despite the impossibility of setting thresholds in a rigorous way, hyperparameter tuning turns out to be very easy to perform on these two methods.

Specifically, we had to fine tune four hyperparameters in total: three for the CUSUM test and one for ADWIN. The CUSUM hyperparameters are m, α and β. First, m indicates the number of observations needed, in stationary

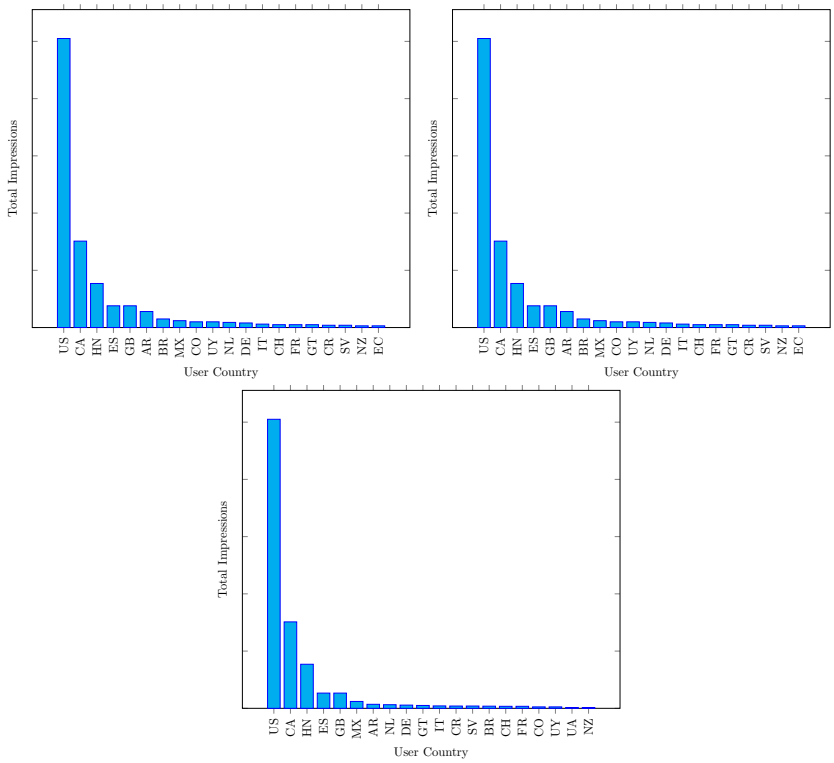

Figure 7.4: The three phases of Honduran global ranking built with the total impressions of every user-country. (top left) Not yet normalized total impressions. (top right) Normalized total impressions. (bottom) Normalized and weighted total impressions.

conditions, to build a good estimate of the current concept mean, which is a fundamental component of both CUSUM statistics g^+ and g^-. Note that, inside the window of m observations, we cannot detect any change because we are still trying to estimate the model. If a drift happens inside the window, we end up with a very bad estimate, which is built with observations coming from both concepts. Therefore, a crucial aspect is that the number of observations between any two drift should be higher than m. With this in mind, we set $m = 15$ because: (1) we think it would be incredibly rare to have two drifts happening in less than 15 days, and (2) 15 observations turned out to be enough to estimate the concept mean. Secondly, α defines the threshold h as a function of the concept mean. As soon as one of the statistics g^+ and g^- exceed h, a drift is detected. Over multiple tests, we found a value of α that satisfies our performance expectations: $\alpha = 4$. At last, β defines ϵ as a function of the concept mean. As we know, 3ϵ should be the minimum gap provided by the a change. We want the value of this minimum gap to be the same of the concept mean. Thus, we set $\beta = 3^{-1}$.

The only ADWIN hyperparameter η, defines the threshold ϵ as a function of the first window's mean. Even though, in general, ADWIN has lower performances with respect to CUSUM, we managed to reach similar performances by setting $\eta = 0.85$.

In Figure 7.5, we show the results of the CUSUM and ADWIN tests on

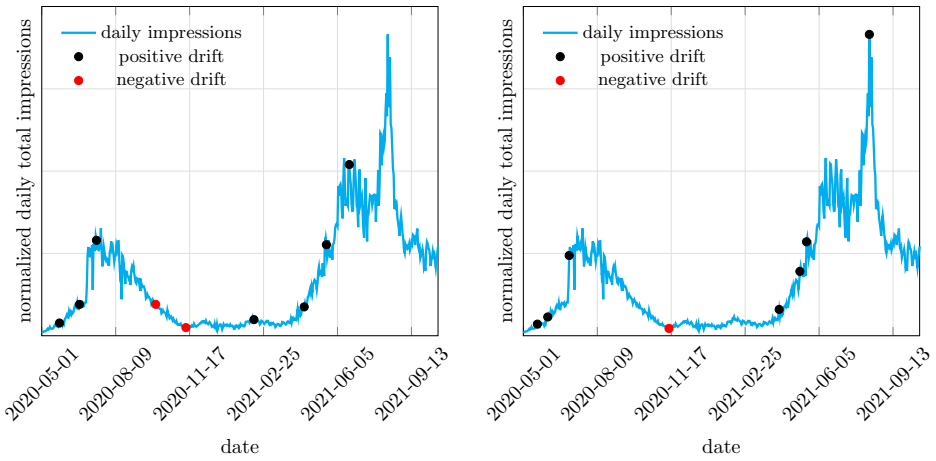

Figure 7.5: Comparing the performance of CUSUM and ADWIN tests, both executed on the whole history of normalized daily total impressions generated by Italian hotels on Italian customers. (left) CUSUM test. (right) ADWIN test.

the same datastream, composed of the whole history of normalized daily total impressions, generated by Italian hotels on Italian customers. The whole datastream spans from 1 May 2020 to 19 October 2021. We performed the test on the whole history of data just for display purposes. Note that, due to NDAs with hotels, the actual impression amounts are not displayed. As you can see, through several trials of hyperparameter tuning, we managed to achieve similar performances. However, CUSUM seems to be more reactive to changes, detecting them a little bit earlier. Moreover, CUSUM has significantly lower computational complexity compared to ADWIN. For these reasons, we decided to adopt the CUSUM test as CDT for our *Performance Trend Monitoring* shown in Algorithm 6.

Even though we cannot evaluate the performance of our CUSUM in a rigorous way, we can still look at the history of detected concept drifts, and see whether or not there are strong motivations behind them. In Figure 7.6, we show the complete history of normalized daily total impressions, generated by Italian hotels on Italian customers. Moreover, black dots represent the moment in which a positive concept drift has been detected by our algorithm. Instead, red ones represent the moment of detection for a negative concept drift. A positive drift indicates an increase in the distribution mean of the new concept, with respect to the old one. Conversely, a negative drift indicates a decrease in the distribution mean of the new concept, with respect to the old one. Now we analyze and explain the sequence of changes detected by our algorithm. The whole datastream spans from 1 May 2020 to 19 October 2021. As we all know, due to the COVID-19 pandemic, a traveling ban was introduced in Italy on 9 March 2020. These restrictions remained in force for almost three months, after which traveling across Italy was allowed again on 3 June 2020. The relaxation of these restrictions are reflected by our datastream and detections. As a matter of fact, we can see that the normalized daily total impressions are very low at

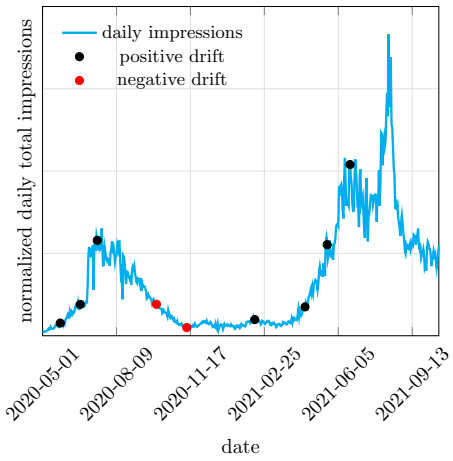

Figure 7.6: CUSUM change detection test on normalized daily total impressions generated by Italian hotels on Italian customers, over the whole history of available data.

the beginning of the stream, indicating a low interest in traveling, as it was still not allowed. Then, on 25 May 2020, we detected a positive change. We think this may be caused by the government's communication of its intention to allow traveling again a week later. From this point on, we see an increase in the generated impressions, as people started booking hotels once more. Moreover, we detect two positive drifts on 21 June 2020 and 14 July 2020: this is undoubtedly caused by summer holiday bookings. The first negative change is detected on 2 October 2020, probably caused by the end of the summer holidays. For most of the period in between 3 November 2020 and 22 April 2021, traveling between regions was forbidden again. In this interval of time, traveling was allowed only between regions having low COVID-19 cases. For this reason, impressions further decreased and a new negative change was detected on 12 November 2020. Then, on 12 February 2020, a positive change has been detected. This may have been cased by the fact that a lot of regions had low COVID-19 cases at the same time, however this may be a false detection. On 22 April 2021, traveling across Italy has been allowed again and no further restrictions have been applied up to today. For this reason, combined with the incumbent summer holidays, the impressions started increasing in April and continued to do so until August. This has been captured by our algorithm with three positive drifts on 21 April, 21 May and 21 June. Therefore, we can say that there is a precise reason behind the detection we provided.

Chapter 8

Conclusion and Future Directions

Online advertising campaign optimization is a challenging problem composed of multiple tasks that cannot be efficiently addressed by humans. The aid of automatic mechanisms is needed to cope with its complexity. In this work, we applied state-of-the-art AI technologies in a real-world online advertising scenario for the company AdsHotel, which is responsible for the optimization of the advertising campaigns of multiple hotels all around the world.

We designed a real-world system built on top of the algorithm proposed by Spadaro et al. [45] to perform safe bid optimization for online hotel campaigns. This task consists of choosing the bid values of a set of sub-campaigns to maximize, in an online fashion, the revenue while satisfying return-on-investment (ROI) and daily budget constraints with high probability. Therefore, the system we built is in charge of interfacing the before-mentioned algorithm with AdsHotel's real-world environment and implementing all the specific features required by such an environment. Specifically, we adopted a sliding window approach to deal with the non-stationary nature of the environment. Moreover, at the request of AdsHotel, we defined some policies, limiting our algorithm to suggest bid values not differing too much from the current ones. Then, we designed the *DynamicROIOpt* algorithm, which solves the bid optimization problem while satisfying the highest possible ROI value constraint, exploiting the optimization algorithm introduced by Spadaro et al. [45] in a binary search fashion. Unfortunately, no hotel has been applying our suggestions consistently over time yet, and thus we could not show and evaluate the performance of our algorithm. However, we still displayed the bid values that our system would suggest for a real-world hotel campaign.

A very important block of our system is in charge of evaluating whether or not the observations generated by sub-campaigns are good enough to build accurate model estimates. To solve the task, we defined two different algorithms, namely *DynamicCountryAggregation* and *SingletonAggregation*. We compared their performance on real-world data, showing that *DynamicCountryAggregation* is the best alternative.

At last, we defined the *Global Rankings* and *Performance Trend Monitoring*

algorithms, providing suggestions on how to expand an ongoing campaign by selecting the most promising user-countries a hotel might open. Then, we display the results of both algorithms run on multiple real-world hotel advertising campaign data. Specifically, we show that the concept drifts detected by the *Performance Trend Monitoring* algorithm are well-founded, according to field experts.

Our work opens up several interesting directions. First of all, a brand new CDT algorithm could be designed to actively detect concept drifts and adapt the model to the non-stationary environment. This task could be tackled by defining strategies to monitor the distributions of the GPs. Moreover, our system could be expanded, considering the Google sub-campaigns at their finest granularity during the optimization procedure. To do so, a new safe Gaussian Combinatorial Multi-Armed Bandit could be designed, having the various multipliers as arms instead of the bid values. Last but not least, it could be interesting to directly analyze the functions estimated by the GPs, rather than analyzing raw data, to check whether or not multiple sub-campaigns can be aggregated.

Bibliography

[1] Accabi, G. M., Trovo, F., Nuara, A., Gatti, N., and Restelli, M. When gaussian processes meet combinatorial bandits: Gcb. In *European Workshop on Reinforcement Learning (EWRL)* (2018), pp. 1–11.

[2] AdsHotel. Adshotel's website. https://www.adshotel.com.

[3] Alippi, C., Boracchi, G., and Roveri, M. Just-in-time classifiers for recurrent concepts. *IEEE transactions on neural networks and learning systems 24*, 4 (2013), 620–634.

[4] Alippi, C., Boracchi, G., and Roveri, M. Hierarchical change-detection tests. *IEEE transactions on neural networks and learning systems 28*, 2 (2016), 246–258.

[5] Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning 47*, 2 (2002), 235–256.

[6] Bach, S. H., and Maloof, M. A. Paired learners for concept drift. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 23–32.

[7] Baena-Garcia, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., and Morales-Bueno, R. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams* (2006), vol. 6, pp. 77–86.

[8] Bernasconi de Luca, M., Vittori, E., Trovo, F., and Restelli, M. Conservative online convex optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2021), Springer, pp. 19–34.

[9] Bifet, A., and Gavalda, R. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (2007), SIAM, pp. 443–448.

[10] Chen, W., Wang, Y., and Yuan, Y. Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning* (2013), PMLR, pp. 151–159.

[11] Chu, C.-S. J., Stinchcombe, M., and White, H. Monitoring structural change. *Econometrica 64*, 5 (1996), 1045–1065.

[12] Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., and Kipersztok, O. Real-time data mining of non-stationary data streams from sensor networks. *Information Fusion 9*, 3 (2008), 344–353.

[13] Department, S. R. Digital advertising spending worldwide 2018-2023. https://www.statista.com/statistics/237974/online-advertising-spending-worldwide/, May 2021.

[14] Department, S. R. Online advertising revenue in the u.s. 2000-2020. https://www.statista.com/statistics/183816/us-online-advertising-revenue-since-2000/, Apr 2021.

[15] Ding, W., Qin, T., Zhang, X.-D., and Liu, T.-Y. Multi-armed bandit with budget constraint and variable costs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence* (2013).

[16] Edelman, B., Ostrovsky, M., and Schwarz, M. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review 97*, 1 (March 2007), 242–259.

[17] Elwell, R., and Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks 22*, 10 (2011), 1517–1531.

[18] Gama, J., Medas, P., Castillo, G., and Rodrigues, P. Learning with drift detection. In *Brazilian symposium on artificial intelligence* (2004), Springer, pp. 286–295.

[19] Garivier, A., and Moulines, E. On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory* (2011), Springer, pp. 174–188.

[20] Gatti, N., Lazaric, A., Rocco, M., and Trovò, F. Truthful learning mechanisms for multi-slot sponsored search auctions with externalities. *Artificial Intelligence 227* (2015), 93–139.

[21] Golrezaei, N., Lobel, I., and Paes Leme, R. Auction design for roi-constrained buyers. In *Proceedings of the Web Conference 2021* (2021), pp. 3941–3952.

[22] Google. Google hotel ads's campaign. https://support.google.com/google-ads/answer/9243943.

[23] Google. Google hotel ads's payment schemes. https://support.google.com/google-ads/answer/9244120.

[24] Google. Google hotel ads's website. https://ads.google.com/intl/it_ALL/hotels/.

[25] Hawkins, D. M., Qiu, P., and Kang, C. W. The changepoint model for statistical process control. *Journal of quality technology 35*, 4 (2003), 355–366.

[26] Hinkley, D. V. Inference about the change-point from cumulative sum tests. *Biometrika 58*, 3 (1971), 509–523.

[27] Hulten, G., Spencer, L., and Domingos, P. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), pp. 97–106.

[28] Kaufmann, E., Korda, N., and Munos, R. Thompson sampling: An asymptotically optimal finite-time analysis. In *International conference on algorithmic learning theory* (2012), Springer, pp. 199–213.

[29] Kolter, J. Z., and Maloof, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research 8* (2007), 2755–2790.

[30] Lai, T. L., and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics 6*, 1 (1985), 4–22.

[31] Liu, F., Lee, J., and Shroff, N. A change-detection based framework for piecewise-stationary multi-armed bandit problem. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018), vol. 32.

[32] Mas-Colell, A., Whinston, M. D., Green, J. R., et al. *Microeconomic theory*, vol. 1. Oxford university press New York, 1995.

[33] Minku, L. L., and Yao, X. Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering 24*, 4 (2011), 619–633.

[34] Nishida, K., and Yamauchi, K. Detecting concept drift using statistical testing. In *International conference on discovery science* (2007), Springer, pp. 264–269.

[35] Nuara, A., Sosio, N., Trovò, F., Zaccardi, M. C., Gatti, N., and Restelli, M. Dealing with interdependencies and uncertainty in multi-channel advertising campaigns optimization. In *The World Wide Web Conference* (2019), pp. 1376–1386.

[36] Nuara, A., Trovò, F., Crippa, D., Gatti, N., and Restelli, M. Driving exploration by maximum distribution in gaussian process bandits. In *International Conference on Autonomous Agents and MultiAgent Systems* (2020), pp. 948–956.

[37] Nuara, A., Trovo, F., Gatti, N., and Restelli, M. A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[38] Nuara, A., Trovò, F., Gatti, N., and Restelli, M. Online joint bid/daily budget optimization of internet advertising campaigns. *CoRR abs/2003.01452* (2020).

[39] Paladino, S., Trovo, F., Restelli, M., and Gatti, N. Unimodal thompson sampling for graph-structured arms. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2017), vol. 31.

[40] Re, G., Chiusano, F., Trovò, F., Carrera, D., Boracchi, G., and Restelli, M. Exploiting history data for nonstationary multi-armed bandit. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2021), Springer, pp. 51–66.

[41] Ross, G. J., Adams, N. M., Tasoulis, D. K., and Hand, D. J. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters 33*, 2 (2012), 191–198.

[42] Ross, G. J., Tasoulis, D. K., and Adams, N. M. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics 53*, 4 (2011), 379–389.

[43] Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. Coalition structure generation with worst case guarantees. *Artificial intelligence 111*, 1-2 (1999), 209–238.

[44] Sinha, P., and Zoltners, A. A. The multiple-choice knapsack problem. *Operations Research 27*, 3 (1979), 503–515.

[45] Spadaro, G. Online bid optimization with return-on-investment constraints. http://hdl.handle.net/10589/170793.

[46] Street, W. N., and Kim, Y. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), pp. 377–382.

[47] Strong, E. K. *The psychology of selling and advertising*. McGraw-Hill book Company, Incorporated, 1925.

[48] Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika 25*, 3/4 (1933), 285–294.

[49] Tripadvisor. Tripadvisor's website. https://tripadvisormediamanager.com.

[50] Trovò, F., Paladino, S., Restelli, M., and Gatti, N. Budgeted multi–armed bandit in continuous action space. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence* (2016), pp. 560–568.

[51] Trovò, F., Paladino, S., Restelli, M., and Gatti, N. Improving multi-armed bandit algorithms in online pricing settings. *International Journal of Approximate Reasoning 98* (2018), 196–235.

[52] Trovo, F., Paladino, S., Restelli, M., and Gatti, N. Sliding-window thompson sampling for non-stationary settings. *Journal of Artificial Intelligence Research 68* (2020), 311–364.

[53] Trovò, F., Paladino, S., Simone, P., Restelli, M., and Gatti, N. Risk-averse trees for learning from logged bandit feedback. In *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), IEEE, pp. 976–983.

[54] UNWTO. Tourism and covid-19 – unprecedented economic impacts. https://www.unwto.org/tourism-and-covid-19-unprecedented-economic-impacts.

[55] Vittori, E., de Luca, M. B., Trovò, F., and Restelli, M. Dealing with transaction costs in portfolio optimization: Online gradient descent with momentum. In *ACM International Conference on AI in Finance* (2020), pp. 1–8.

[56] Škare, M., Soriano, D. R., and Porada-Rochoń, M. Impact of covid-19 on the travel and tourism industry. *Technological Forecasting and Social Change 163* (2021), 120469.

[57] Williams, C. K., and Rasmussen, C. E. *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[58] Xia, Y., Li, H., Qin, T., Yu, N., and Liu, T.-Y. Thompson sampling for budgeted multi-armed bandits. In *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[59] Ye, Y., Squartini, S., and Piazza, F. Online sequential extreme learning machine in nonstationary environments. *Neurocomputing 116* (2013), 94–101.

[60] Zhang, W., Zhang, Y., Gao, B., Yu, Y., Yuan, X., and Liu, T.-Y. Joint optimization of bid and budget allocation in sponsored search. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), pp. 1177–1185.