Improvement of the contextual multi-armed bandit algorithm for persuasion profiling

Orekhov, V.

Award date:
2015

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

Improvement of the Contextual Multi-armed Bandit Algorithm for Persuasion Profiling

Master Thesis

Vladimir Orekhov

Supervisors:
dr. Mykola Pechenizkiy (TU Eindhoven)
dr. Maurits Kaptein (Webpower)
dr. Aristides Gionis (Aalto University)

Assessment committee:
dr. Mykola Pechenizkiy
dr. Maurits Kaptein
prof. dr. Paul De Bra
dr. ir. Irene Vanderfeesten

Eindhoven, August 2015

Abstract

Contextual multi-armed bandit (CMAB) problems are sequential decision problems, where on each step one need to choose one process from several alternatives and learn by interacting with it. These problems have a lot of applications in user profiling in the field of marketing automation. One of the popular application settings for user profiling is an e-commerce store where each user is treated on an individual level and the content on each page is personalized according to his profile. The delay between two interactions in this case may be very small. This setting introduces the restriction of fully online processing, which ensures that users do not experience any observable delays during the interactions with the website. In this case, the state-of-the-art algorithms for CMAB that have proven optimal regret bounds are computationally prohibitive and reasonable heuristics come in handy.

PersuasionAPI is a persuasion profiling service that solves CMAB problem on the individual user level under the restriction of fully online processing. This thesis describes the whole process of improvement of the PersuasionAPI core algorithm. The new version of the algorithm contains a reinforcement learning model and implements a heuristic that was designed within this research. Since the reinforcement learning model embeds static knowledge, the test of the new improved version of the PersuasionAPI core algorithm against the current version is based on the historical data from 8 major clients of Webpower. In the test the new version of the algorithm shows a relative improvement of approximately 0.4% less cumulative regret compared to the current algorithm and therefore represents a promising direction for further improvement. The next suggested step is to conduct field experiments to gain more empirical evidence of this improvement.

From the company perspective, this thesis not only delivers a concrete practical improvement, but also brings the prototyping environment that introduces a quick and cheap way to test and select the most promising hypotheses for the PersuasionAPI core algorithm improvement. Together with the prototyping environment, the data-driven improvement process is introduced. These two results structure and significantly simplify further activities on improving the PersuasionAPI algorithm within the company.

Acknowledgments

I express my sincere gratitude to my supervisor from the academia side, dr. Mykola Pechenizkiy, for his guidance and constructive criticism that challenged me in a good way and helped me to further improve as a software developer and as a researcher.

I was more than happy to work with my supervisor from the company side, dr. Maurits Kaptein. Thank you for your help and for your time, and especially for your positive attitude and support.

Mom, dad, Viola, you know that nothing would have been possible without your love and support.

Contents

Chapter 1

Introduction

1.1 Motivation

In modern marketing there is a variety of techniques aimed at validating the hypothesis that a certain change to the process of interaction with the customers will or will not increase/maximize the desired outcome. Such changes to the process can range from changing the greeting line on a website, the text of a notification in a mobile application to the time period between e-mailings or even the color of the button in the graphical user interface calling for action. The desired outcome can also be anything, as soon as it reflects the most important metric for the company, the representation of success of the communication, also referred to as a key result indicator (KRI).

When it comes to introducing such a change, there is typically a hypothesis that this change will have a positive effect on the desired outcome, as represented by a change in some target metric. Modern technologies and digital channels make the process of validating such hypotheses quick, easy, and relatively cheap. Hypotheses are rarely validated during closed-door meetings; rather, they are validated through interactions with users and learning from their reactions. A simple example of such method is two-sample hypothesis testing, or A/B testing. This method is put into practice by such companies as Google and Booking.com [1, 2], which are the industry leaders in their business domains.

In A/B testing, two versions of the communication process, which are identical except for the feature being tested, co-exist and are used randomly on a number of users (a test pool). Frequently, process A is the current communication process and process B represents a change to process A and needs to be tested. The purpose of the test is to discover whether or not the desired outcome metric is improved when following process B compared with process A. A more general case of A/B testing is multivariate testing, when more than one variable is being tested. In the literature [3, 4] this group of approaches is often referred to as

Randomized Controlled Trials (RCTs). In terms of RCTs, the goal is to estimate the causal effect of the change applied to the original process.

This approach has been proven to deliver meaningful results quite fast. For example, Booking.com states that they have been performing A/B tests for a decade already and consider it as one of their primary decision-making practices [2]. However, in this case, all the users are perceived as a homogeneous group regardless of the fact that the bigger the group is, the harder it is to divide it into only two homogeneous subgroups based on their behavior. Several questions arise. What if process A leads to an improved desired outcome for 60 percent of the users and process B leads to the same improvement for 30 percent? Should the latter average causal effect be sacrificed to follow the rule of the majority? Should the user base be treated as an inseparable mass?

One of the answers to these questions is adjusting the granularity of the user segments and applying persuasion techniques that fit best to each of the segments. Marketing segmentation in general uses a "divide-and-conquer" principle to transform one large segment that contains all the users to a number of particular segments that incorporate users with relatively similar behaviors. The task of market segmentation methods is to cluster users into a number of segments, where for each pair of users in a cluster, their behavior is much more similar than for a pair of users from different clusters. For instance, in the modern research of recommender systems clustering methods are applied both to the items space and the context space, where the users that have similar contexts are assumed to have a similar behavior (see Section 4.2.2) [5, 6, 7].

When it comes to e-commerce, one of the most beneficial market segmentation methods is behavioral or psychographic segmentation. With this method, segmentation can be performed based on such parameters as usage rate, readiness to buy, interests, values, general activity level and more [8]. In today's e-commerce industry, huge amounts of data about users' behaviors, actions, and reactions are being collected every day; therefore, market segmentation is frequently data-driven instead of preplanned. This, again, supports the hypothesis validation approach, and, in this case, the number of segments and rules specifying the division between those become such hypothesis.

Market segmentation is a great approach and is proven to work. For instance, in Amazon recommender system the similarity metric is defined for the customer space and dimensionality reduction is applied to this space in order to effectively group similar customers into clusters. According to [9], after implementing this approach "*both the click-through and conversion rates – two important measures of Web-based and email advertising effectiveness – vastly exceed those of untargeted content such as banner advertisements and top-seller lists*".

However, such segmentation still operates on a level of customer segments and is not precise enough to target the needs of individual customers. Distributed computing frameworks, non-relational random time access databases and general availability and low price of storage space enable the extreme case of market segmentation to the point where every user is a separate segment. There is a trade-off between the segmentation granularity and the quality of the analysis that can be conducted for such micro-segments compared with only a few big segments. In addition, certain restrictions on the complexity of the algorithms apply when it comes to reinforcement learning completely online, in real time. Nevertheless, marketing automation products that purport to consider users on an individual level gained a lot of traction in recent years, and one of them is PersuasionAPI[1], which is the subject of this study.

1.2 Introducing PersuasionAPI

PersuasionAPI is a marketing automation product designed to apply different sales strategies to users based on their individual behaviors. The fact that there is still some room for improvement in terms of its performance is the primary motivation for this thesis.

PersuasionAPI offers an extreme segmentation approach where each user represents a separate segment. There is a number of persuasive strategies that frequently include authority, scarcity, social proof, and baseline, which can be used in the communication with each individual user. In a nutshell, these strategies represent the following patterns of user behavior [10]:

- **Social proof**: "*People will do the things that others do*".
- **Scarcity**: "*People tend to value products or services higher when they are limited in time, quantity, or availability*".
- **Authority**: "*People will tend to obey authority figures*".

Initially, the user's reaction to each strategy is not known, and the algorithm learns it by interacting with the user. The high-level goal of the algorithm is to find the most beneficial strategy for each user through sequential interactions. In the literature, this problem is referred to as the multi-armed bandit problem [11, 12, 13]; in the case of PersuasionAPI, this problem is being solved for each user individually. The multi-armed bandit problems class will be discussed in detail in Chapter 2. PersuasionAPI as a service (application program interface or API) is a web marketing customization software that applies its reinforcement learning algorithm to an input stream of requests on an individual level and outputs the result in the form of a particular persuasion strategy that is the most valuable to be used next for that particular person at that particular

[1] http://www.webpower.eu/marketing-solutions/persuasion-profiling/

point in the communication with that person. PersuasionAPI can be flexibly adjusted to different cases, be it a real-time dynamic content generation on a website based on the user profile or a marketing e-mail campaign.

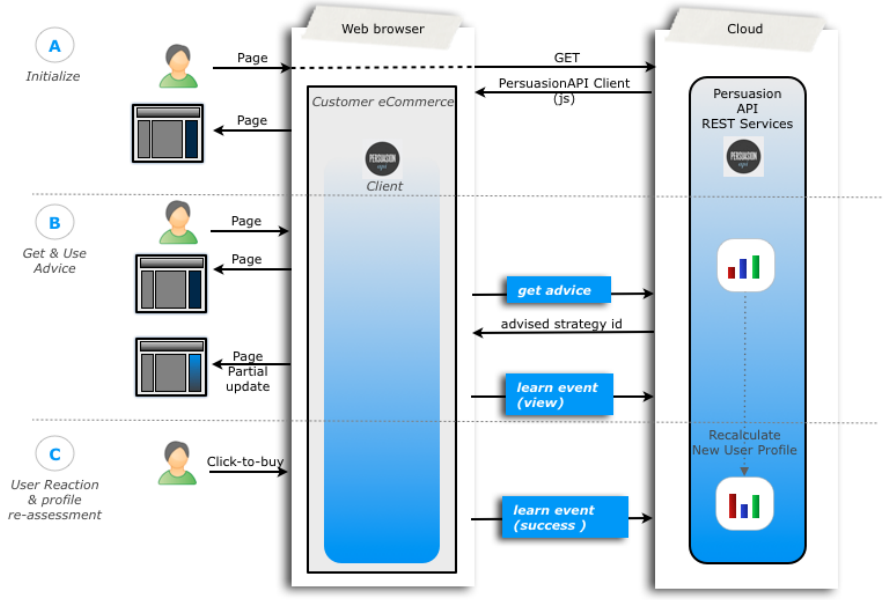

Figure 1.1: PersuasionAPI use case scenario [14]

Figures 1.1 and 1.2 describe the exact process of communication with the user within PersuasionAPI. In terms of the API calls there are three main calls. The first request sent from the user side is called *getAdvice* and requests an advice on which persuasion strategy to use in the next interaction with the current user. The core back-end algorithm of PersuasionAPI solves a multi-armed bandit problem, chooses strategy s and sends it as a response. After this strategy is presented to the user, the next request sent from the user side is called *learnEvent* and it is sent only in case of success, which is defined within a particular campaign. The core algorithm then observes user's response and updates the persuasion profile of the user accordingly.

1.3 Corporate context

This thesis is a separate project conducted at Webpower[2], a marketing automation company that offers a platform-as-a-service (PaaS) marketing automation solution and a variety of services based on it.

[2]http://www.webpower.eu/

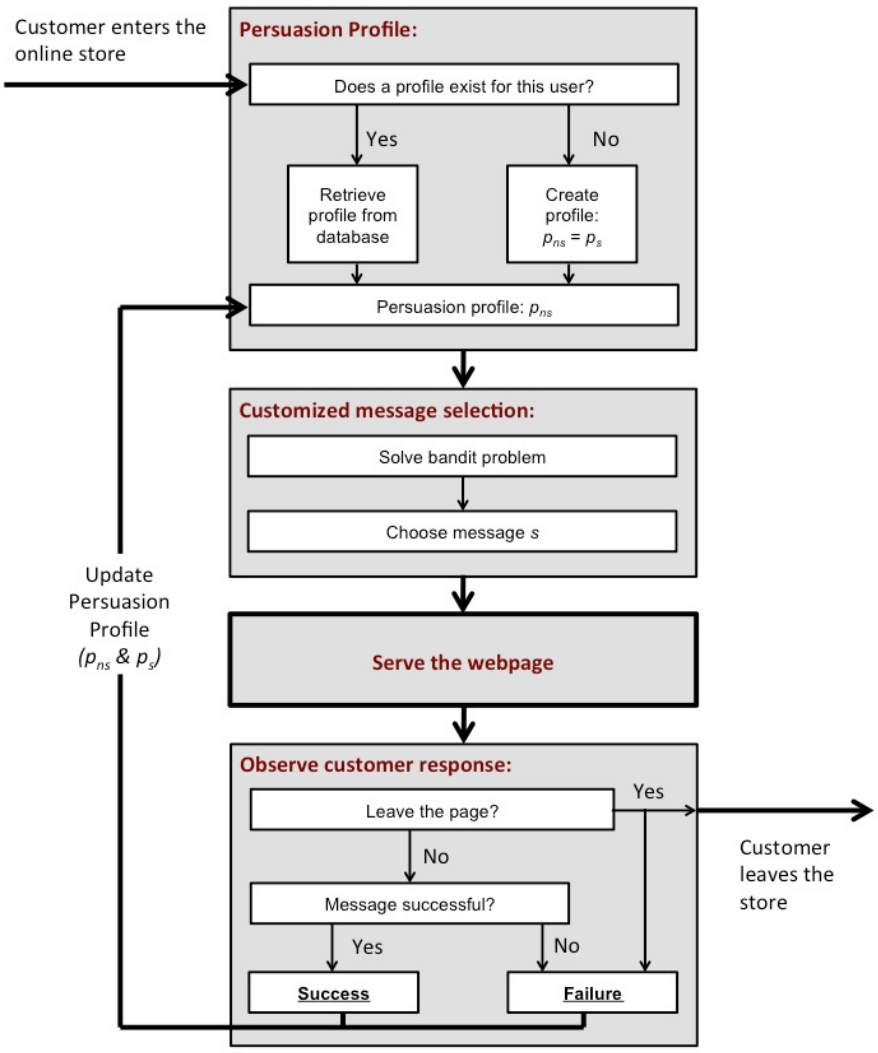

Figure 1.2: PersuasionAPI algorithm description [15]

Webpower's management plans to transform the current system architecture to have a distributed core data analytics platform that will process, store, and distribute enormous amounts of data in real-time and a number of loosely coupled services that will communicate with the platform. One of the main motivations behind this transformation is the need to enrich Webpower's offerings with new services. A promising category of services is based on the individual profiling and persuasion of customers. To gain expertise in persuasion profiling, Webpower acquired Science Rockstars company, the company that created the PersuasionAPI service.

Although the persuasion algorithm used in PersuasionAPI shows competitive performance compared to the other methods [15], there are still several patterns in user behavior that are not taken into consideration when a decision about which persuasion strategy to use is made, meaning that there is likely still some space to improve.

The overall goal of this thesis is to study different possibilities to improve the core contextual multi-armed bandit algorithm of the PersuasionAPI product and to come up with a practical contribution to the improvement of the existing version of the algorithm. This is the first step towards the new service offering under Webpower brand.

This improvement in general aims at raising the quality of the persuasion profiling, or increasing the probability of the desired outcome by choosing the optimal way of interaction. To monitor the improvement, the specific performance metric is chosen and a set of experiments are conducted.

Thesis structure

The thesis has the following structure. **Chapter 2** presents the requirements analysis and introduces the general multi-armed and contextual multi-armed bandit classes of problems that PersuasionAPI focuses on. In **Chapter 3**, the PersuasionAPI version of CMAB problem is discussed and the implementation details of PersuasionAPI and specific design choices made along the way are described. **Chapter 4** presents the state-of-the-art improvement directions and focuses on the most promising direction. Then the process and the methods of the research that were used are discussed. The following **Chapters 5 and 6** focus on the analysis of two specific hypotheses and their application to the existing algorithm. **Chapter 7** covers the process of designing and prototyping the new improved version of PersuasionAPI. **Chapter 8** presents the process of data-driven improvement of PersuasionAPI core algorithm. **Chapter 9** concludes the thesis and describes the outcomes both from the research and the business perspectives.

Chapter 2

Problem statement

This chapter contains an overview of the problem being approached within the PersuasionAPI product. Firstly, the requirements are presented and analyzed. Secondly, the core multi-armed bandit problem is discussed from various angles including the more classic approach [16] and the newer reinforcement learning approach [17]. Some particular parts of the problem are discussed in detail and proper examples are provided to illustrate the problem.

2.1 Requirements analysis

In this section the requirements of the current version of PersuasionAPI are presented and critically discussed. Note that these requirements are reflected in the design choices described in Chapter 3.

2.1.1 No slow start

In PersuasionAPI there are few points of interaction with each user as the interaction itself occurs within a campaign that typically has a narrow time frame.

Therefore, the algorithm should be able to give relatively good estimations of user's response rates from the very beginning, in other words it should not suffer from a slow start. This is supported by the reinforcement learning survey, which states that "*many algorithms come with a provable guarantee of asymptotic convergence to optimal behavior. This is reassuring, but useless in practical terms. An agent that quickly reaches a plateau at 99 % of optimality may, in many applications, be preferable to an agent that has a guarantee of eventual optimality but a sluggish early learning rate*" [18].

Discussion

This requirement affects the choice of the estimation model of the success probability (discussed in detail in Section 3.2). Two simulation studies of Web

Persuader, the predecessor of PersuasionAPI is described in [15]. One of the studies covers the estimation part of the algorithm, where the four methods are compared, namely two naive strategies (Individual Mean and Grand Mean), Web Persuader itself and state-of-the-art Hierarchical Bayes Logistic model (HBL) method. An important criterion of comparison is not only the overall performance curve, but also the performance of the algorithm right from the start.

Evidence for the hypothesis of having few points of interaction is shown based on the historical data, where for a big number of client companies the average number of interactions with a user is 4.6 times (discussed in detail in Section 3.3). Therefore, it is empirically shown that the algorithm used in the PersuasionAPI should not suffer from a slow start since it can negatively affect the performance of the algorithm. In other words, there might be a much better estimation model for a general case, which requires comparatively many observations to start giving good estimation and it will have a worse performance than a simpler model just because there is no opportunity to facilitate this many observations.

2.1.2 Fully online processing

When content personalization is implemented in a digital form, it faces a well-known trade-off between performance and execution time, also known as a time-quality trade-off [19, 15, 20]. In one of the papers it is described follows: "*In providing personalized services, a website [...] can deliver an optimally personalized version of the content to the visitor, possibly with a long delay because of the computational effort needed, or it can deliver a suboptimal version of the content more quickly*" [19].

For the PersuasionAPI case the main priority is smooth user experience, which incorporates seamless interaction with the site. Therefore, PersuasionAPI should work completely online, in particular the time between the request for advice being sent for a particular user and the response being received should be seamless for this user, which usually stands for the whole round-trip performed in under a second.

Therefore, PersuasionAPI has a requirement of being completely online. This requirement imposes restrictions on the computational complexity of the algorithm, which are discussed in detail in Chapter 3.

Discussion

There are solutions that try to minimize the average waiting time for the users while accepting the computational complexity of good personalization models [19, 21]. It is possible to approach this problem by designing sophisticated scheduling mechanisms, which order the incoming requests for personalized content [19]. One of the papers describes several queueing approaches to

scheduling such incoming requests and more importantly a batching approach that determines the optimal batch length. An interesting experimental result of this work is the dependency of the profit from the batch length. This empirical chart shows that the profit decreases almost linearly as soon as the batch length is longer than 0.5 second. This gives additional support to the importance of seamless user experience and split-second waiting times.

Although several papers demonstrate an interesting approach to scheduling the incoming requests and processing them in batches, the optimal batch length can vary from user to user and from website to website. More importantly, such sophisticated queueing and batching mechanisms lack predictability and performance guarantees.

These two characteristics are crucial in order to keep the user experience level on a high mark in virtually any setting, and since user experience is chosen as a crucial factor in PersuasionAPI, prioritizing time over quality in this case is a reasonable choice.

2.2 Multi-armed bandit problem in general case

The core challenge of PersuasionAPI product is solving the multi-armed bandit problem in an effective and efficient way, and this problem in general case is introduced in the following section.

There are many definitions of the multi-armed bandit problems depending on the areas of its application (economics, statistics, medicine, biology, control, and more). The most general and applicable to the case being researched states that a bandit problem involves sequential selections from $k \geq 2$ stochastic (random) processes (a collection of random variables that represent the evolution of some system of random values over time) [16]. Time may be discrete or continuous; however, in most cases it is discrete. The objective in the bandit problems is to maximize the expected value of the payoff:

$$\sum_{m=1}^{\infty} \alpha^m Z_m, \tag{2.1}$$

where Z_m is the variable observed at stage m and α are non-negative numbers and α^m forms a discounted sequence. This objective is equal to minimizing the expected regret, which is defined as follows:

$$\sum_{m=1}^{\infty} \alpha^m (R(S^*) - R(S)), \tag{2.2}$$

where $R(S)$ is the reward function that takes strategy S as an input, S is the candidate strategy, and S^* is the optimal strategy that is not known.

For a finite time horizon the objective can be formulated as minimizing the undiscounted finite time expected regret given the time horizon T [16, 22]:

$$\sum_{t=1}^{T}(R(S^*) - R(S)), \tag{2.3}$$

The goal of solving a bandit problem is to design a strategy for choosing an arm (or a stochastic process) at each time t so that the objective stated above is satisfied. More formally, for a 2-armed bandit the strategy is defined as a "*function that assigns to each (partial) history of observations the integer 1 or 2 indicating the arm to be observed at the next stage*".

An important characteristic of a bandit problem system is that when one arm is played, there is an outcome or payoff that immediately follows. This information is crucially important for improving and updating the strategy as arms are played and payoffs received.

Exploration-exploitation trade-off

A usual setting for a multi-armed bandit problem is that there are k arms and their θ parameters (in the case of a Bernoulli distributed payoff function) are not known. This gives high uncertainty, which needs to be decreased over time. At the same time, the system should at some point start benefiting from the knowledge gained. To accomplish these two goals, one has two types of actions to perform at each time t, given the history of previous t - *1* rewards:

- **Exploration** is used for learning, or playing one of the alternative arms that are not optimal *at time t* and gaining knowledge about the payoff[1] of these arms and, ideally, finding an arm with a better payoff than the current one;
- **Exploitation** is used for earning, or playing the arm that has the best probability of reward *at time t* to increase the cumulative reward.

Exploitation is defined as a process of playing the arm with the highest estimated payoff at time t and exploration as a process of playing any other arm. Any strategy should embed a proper balancing mechanism between exploration and exploitation. Improper balancing mechanism may result in one of two corner cases: either the strategy will tend to over-exploit, remaining playing a suboptimal arm and never having a chance to discover the optimal one, or the strategy will tend to over-explore, gaining more and more knowledge about the arms' payoffs but not making use of this knowledge, and, hence, not playing the optimal arm that might be already explored.

[1] In the case of Bernoulli distribution gaining knowledge is just becoming more confident about the estimation of the θ parameter.

Strategy

Strategy describes the algorithm of playing one arm at each time t so that cumulative regret is minimized. Strategies consist of the two parts:

- Estimation of the expected reward;
- Choosing the arm to be played, which covers the exploration-exploitation trade-off given the model and the estimate.

Example

For simplification purposes, assume that there are two arms with reward functions described by Bernoulli distributions under parameters θ_1 and θ_2. Bernoulli distribution describes a random variable that takes value 1 with probability p and value 0 with probability $1 - p$, and its probability mass function is $f(k, p) = p^k(1 - p)^{1-k}$. If these parameters are known, then the optimal strategy would be to select the arm with the highest θ and then play it at each time t. This strategy will give 0 regret simply because it is impossible to find a strategy that will give a better cumulative payoff over time. More formally, the regret is defined as the difference between the cumulative payoff obtained by using optimal strategy and the one obtained by using the current strategy (see equation 2.3). During the simulations when the θs used to generate the test data are given the regret value can be computed for any time t.

Note that although in theory regret can not drop lower than 0 because there is no strategy that can give a better cumulative payoff than the optimal strategy on an infinite run, when it comes to simulations where samples are taken from particular distributions, it can happen that for a particular point in time drawing a sample from the optimal arm's distribution will give 0 and drawing a sample from a non-optimal arm will give 1. If it happens in the beginning, it will cause regret to temporarily drop below 0.

An example of a trivial strategy is playing one arm every time. Assuming that the choice between two arms in the beginning is completely random, this strategy has a 50% chance of choosing the optimal arm with the highest θ and therefore be optimal. This is clearly not the best strategy in half of the cases since it does not satisfy the objective of maximizing the expected payoff; however, at the same time, it is still a valid strategy that describes the action that needs to be taken at each time t.

2.3 Contextual multi-armed bandit problem

The contextual multi-armed bandit problem is a generalization of the simple bandit problem discussed above, which introduces environment or context as a new component of the system. In this case, at each point in time the decision

of choosing an arm to play depends not only on the previous history, but also on the current state of the environment. The payoff received is also dependent not only on the arm played, but also on the current state of the environment. Environments can be either non-reactive or reactive [23]; in latter case, the current choice of the arm can affect the environment in the future (i.e., the environment reacts to the actions taken by the agent).

Modern literature [17] relates contextual bandit problems to the class of reinforcement learning problems and introduces slightly different terminology and definitions as compared with the general bandit problem. The general definition [17] states that "*reinforcement learning involves [agent] learning while interacting with the environment*". The four elements of the reinforcement learning system are defined as follows:

- **Policy** (another name for strategy) "*defines the learning agent's way of behaving at a given time*". The term "policy" in this context means "*a mapping from perceived states of the environment to actions to be taken when in those states*" [17] and has an analogy with stimulus-response set of rules.
- **Reward function** is a "*[mapping from] each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state*". This function describes the reward for each possible combination of the states of the environment and of the actions taken. Cumulative reward is a subject to be maximized according to the definition given in Section 2.2.
- **Value function** is a predictive metric that describes the expected cumulative reward in future based on the state after the current action is taken.
- **Model** is an optional element and describes the behavior of the environment. Models are used for planning, "*any way of deciding on a course of action by considering possible future situations before they are actually experienced*".

The contextual multi-armed bandit as a reinforcement learning problem has a set of available actions $a_{i=1..n} \in A_t$ at time t (each action represents an arm) and the environment has its state C_t at time t. At each point in time t, the agent chooses one action based on its policy and gets a reward $R(a_{it}, C_t)$, which is dependent on both the action taken and the current state of the environment. The agent receives this reward and updates its policy accordingly.

The reinforcement learning definition of CMAB can be seen as an expanded version of the one discussed previously in Section 2.2. While the reinforcement learning definition describes the reward function as an exact mapping from state-action pairs of the environment to the payoffs, the more classic one [16]

describes arms as stochastic processes, which means that there can be no exact mapping to the outcomes of these processes and the whole idea is to discover the parameters of the distributions of random variables instead of mapping each state-action pair to the exact payoff. The latter definition can be seen as a generalization of the former one, where there is an exact mapping from each state-action pair to the expected payoff but the number of states is infinite, so there is no way to get to know this mapping function precisely and modeling it in terms of a stochastic process makes the problem approachable. For the particular problem discussed in the thesis the classic definition is used and user responses are modeled as stochastic processes.

2.4 Thesis objectives

Given a rather broad topic of improving the algorithm described in the previous section, it is necessary to define the main objectives as parts of a sequence of steps. Conceptually, it all begins with research of the current state-of-the-art and improvement directions and opportunities. After that, some particular hypotheses are tested. These tests not only illustrate the process of improvement, but are also very valuable from the business point of view as they answer the question of what particular improvement should be implemented in the next version of the algorithm. Next, the new improved prototypical version of the PersuasionAPI algorithm is designed, developed and tested to compare its performance against the current version.

More formally, the main objectives of this thesis can be formulated as follows:

- **Objective 1**: Exploration of possible directions and methods to improve the existing algorithm (Chapter 4).
- **Objective 2**: Testing two specific instances of these improvement directions, namely the effect of repetition on the user response rate and the correlation between response rate for different pairs of strategies (Chapters 5 and 6).
- **Objective 3**: Design and development of the improved version of the PersuasionAPI algorithm and its testing against the current version (Chapter 7).
- **Objective 4**: Design of a data-driven improvement process and implementation of the corresponding prototyping environment (Chapter 8).

The next chapter presents an overview of the implementation of the PersuasionAPI core algorithm and describes specific choices made along the way.

Chapter 3

PersuasionAPI CMAB algorithm design choices

This chapter maps the generic CMAB problem described in the previous chapter to a particular instance of it within PersuasionAPI product and describes the current existing version of PersuasionAPI algorithm on a relatively high level of abstraction, covering the main design choices made along the way.

3.1 PersuasionAPI version of CMAB problem

The PersuasionAPI product contains an algorithm and a surrounding infrastructure that aims to solve a particular instance of the contextual multi-armed bandit problem class – web marketing customization on an individual basis. Specifically, it implements a learning algorithm that maintains a policy of showing one or another persuasive message for each individual user. There is a fixed number of persuasive messages per marketing campaign, usually containing authority, scarcity, social proof, and baseline persuasive messages. The algorithm learns from interactions with the users and from user's responses to be able to estimate the success probabilities for each strategy.

The next several paragraphs map the main components of a contextual multi-armed bandit problem to a particular PersuasionAPI case.

Time-flow Time in the system is described as a discrete variable, because only the sequential order of events is necessary for the algorithm. The underlying implementation of the algorithm contains the timestamps of the events; however, for the purposes of solving the bandit problem, only the natural ordering of the timestamps is used.

Arms Each strategy represents an arm with the Bernoulli distribution of the reward. The meaning behind it is that each user will either perform a target action (be it clicking or facilitating payment or anything that is considered a meaningful result by a customer) or he will not. A big assumption, which is supported by psychological and behavioral research, is that people's reactions to particular types of persuasion remain relatively constant over time; that is, the multi-armed bandit problem that models user interactions is stationary and does not change over time. Thus, it is possible to talk about a Bernoulli distribution with a constant θ parameter for each user for each persuasion strategy.

Environment/context Currently, context is limited to a very narrow scope and separate individuals are distinguished; thus, context can be seen as the user id.

3.2 Estimation of the expected reward

The policy of PersuasionAPI algorithm uses the Bayesian inference method, which describes how to update the probability of a hypothesis when evidence is acquired:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} = \frac{P(X|\theta)P(\theta)}{\int P(X|\theta')P(\theta')d\theta'}, \tag{3.1}$$

where θ is the hypothesis, or the estimation of the parameter of Bernoulli distribution, X is observation or evidence (0 or 1 in this case), $P(\theta)$ is the prior probability (the probability of θ before X is observed), $P(\theta|X)$ is the posterior probability (the probability of θ when X is observed), and $P(X|\theta)$ is the likelihood function (the probability of observing X given θ).

Getting the value of $P(X)$ is computationally demanding, so the approach taken for the PersuasionAPI case is to substitute the previous equation with the following one:

$$P(\theta|X) \propto P(X|\theta)P(\theta) \tag{3.2}$$

Applying the normalization procedure to this proportional equation by adding a multiplicative factor or a normalizing constant will make that function a probability distribution (the integral over the entire range is 1).

The likelihood function is modelled by the Bernoulli distribution with parameter θ, which is conjugate to the Beta distribution. In terms of the Bayesian inference model, Beta distribution is a conjugate prior to Bernoulli distribution; thus, Beta chosen as a prior distribution will give the same Beta family of distributions for the posterior:

$$Beta(\alpha_1, \beta_1) = Bernoulli(\theta) \times Beta(\alpha_0, \beta_0) \tag{3.3}$$

Parameters α and β are called hyperparameters to distinguish them from the underlying parameter θ, and can be chosen to represent successes and failures, respectively. In this case, the whole algorithm of updating the model based on the outcome of the interaction boils down to updating the parameters of Beta distribution that represent the probability distribution of the θ parameter for each individual and for each persuasive strategy. In the case of success, $\alpha_t = \alpha_{t-1} + 1$ and $\beta_t = \beta_{t-1}$, in the case of failure $\alpha_t = \alpha_{t-1}$ and $\beta_t = \beta_{t-1} + 1$.

As a result, the two parameters α and β are stored for each strategy for every user and are updated based on the result of the interaction with the user. Beta distribution with these parameters characterizes the likelihood of a particular strategy to succeed for a particular user. Initially, $\alpha_t = 1$ and $\beta_t = 1$ represent a uniform distribution. This conveys the idea that there is no prior knowledge about how users will respond to persuasive strategies and, therefore, θ parameter that specifies the probability of success is initially distributed uniformly. At the same time, when sufficient data is collected for multiple users it might be valuable to embed these data into the initial parameters of Beta distribution (when there was no prior interaction with this user using this particular persuasive strategy). The approach to this problem is described in the next section.

Discussion

For the case of MAB with stochastic reward function that has a Bernoulli distribution with unknown parameter it is very common to use a Bayesian approach and to represent the probability distribution as a conjugate Beta distribution [24, 13]. It is a simple and effective way to estimate the expected reward.

Another approach can be building a full Hierarchical Bayes model, which is computationally prohibitive. For example, *BayesGeneral* scheme requires solving a two-dimensional convex, non-differentiable minimization problem for two parameters. This approach leads to a long computation time and an approximation for it is called *Bayes2x2*.

It is empirically shown that PersuasionAPI method outperforms both Individual Mean and Grand Mean methods and has a worse performance compared to the state-of-the-art Bayesian Hierarchical Logistic (HBL) model [15]. HBL is a very computationally demanding method since it estimates the success probability based on all the history points, therefore it does not fit into the requirement of fully online processing.

3.3 Handling insufficient data on an individual level

The average number of interactions with one user is comparatively small, with the average number of interactions per one user across 20 client companies being 12.2. Moreover, Table 1 presented in Appendix A shows two distinct cluster of clients. One cluster represents campaigns with a relatively longer history of interactions (31.0 interactions on average) and the other bigger cluster contains clients that tend to have very few interactions with the user (4.6 interactions on average). The first cluster represents a typical situation of custom content generation, when persuasive messages are embedded into a list of products shown to the user simultaneously. This means that the average number of interactions in the case of the first group of campaigns is comparable with the number for the second cluster. The algorithm should be able to deliver relatively good predictions based on as few as 4.6 interactions.

This question is related to the choice of the initial values for the number of interactions and the probability of success. One way is to set the initial values of these parameters to the static values each time a new user is encountered: a realistic approximation for the success probability and a small number representing the number of interactions so that the approximated success probability can converge to the real success probability quickly. However, in the latter case, information about the mean users' success probability is not used. In general, information about users accumulates over time and becomes an increasingly accurate representation of the initial assumptions about the new user than any static values. It also represents the current mean for a particular experiment at each run, not just a static approximation.

In the current PersuasionAPI version the following ad hoc heuristic is used to compute the estimated probability based on the individual and the average overall probabilities to obtain the approximate hierarchical model using shrinkage factor [25, 26]:

$$\hat{p}_{is} = B_{is}P_s + (1 - B_{is})p_{is}, \tag{3.4}$$

where P_s is the average probability of success over all the users for the particular strategy s, p_{is} is the probability of success for the user i for the particular strategy s, and B_{is} is the shrinkage factor that gives a weight to both average and individual success probabilities based on the number of observation for the individual:

$$B_{is} = \frac{1}{\sqrt{n_{is}}} \tag{3.5}$$

3.4 Persuasive strategy choice

Every time an advice request comes into the system, meaning that there is an upcoming communication with a particular user within a marketing campaign planned, the algorithm should deliver a response containing a strategy that will be applied to the next communication session. The implementation has to address the the exploration-exploitation trade-off.

Thompson sampling is one of the oldest heuristics to address the exploration-exploitation trade-off [27] and is implemented in PersuasionAPI. It embodies a comparison strategy that takes into consideration not only the most probable value of the success probability from its distribution (the maximal value from the distribution), but also the degree of uncertainty. It is sufficient to "*to draw a random sample θ^* from the posterior at each round and select the action with the greatest expected reward according to the current draw*" [15]. Based on these random samples, the strategy with the maximal sample is chosen.

Discussion

There are several methods that address the exploration-exploitation trade-off. In this section they are briefly introduced and critically discussed with respect to the applicability to PersuasionAPI case. Thompson sampling is shown to still be a valid choice given the constraints of fully online processing.

The ϵ-greedy method [28] is often called the simplest yet most used method in the literature [11, 12, 29, 30]. It selects a random arm with static probability ϵ and chooses the arm with the highest empirical mean reward with probability $1 - \epsilon$. The bound of the expected regret in this case is only linear, which shows poor performance of the method.

Well-known adjustments of this method are ϵ-decreasing, when probability ϵ is decreasing over time, and ϵ-first, when the exploration is done all at once by choosing each of the arms and after that only the arm with the highest empirical mean reward is chosen. ϵ-decreasing variant has proven poly-logarithmic bounds.

Another simple method is *SoftMax* or *Boltzmann exploration*, which picks each arm with a probability proportional to its average reward. This algorithm has a parameter τ and poly-logarithmic bounds are proven for the case of decreasing τ [11].

Another popular method is called Gittins indices [31]. In this work it is proven that the MAB problem is indexable and a formula to compute such indices is introduced. In particular, this paper introduces Bellman equation and solves it for a known payoff to obtain particular values (indices). The formula is only

dependent on α and β parameters of Beta-distribution and discount d in case of a discounted infinite sequence. On each step, the arm with the highest index is chosen [24]. This method has an explicit guarantee of being optimal [32, 33].

However, this method is more computationally demanding than Thompson sampling. Thompson sampling is also proven to be asymptotically optimal [34]. At the same time, it is less computationally complex than the Gittins indices method.

There is a number of more complex methods that have optimal regret bounds, such as Exponential weights algorithm for Exploration and Exploitation with Experts (*EXP4*). However, they are very computationally heavy and in literature are frequently referred to as computationally prohibitive. For example, in one of the papers it is stated that "*schemes with optimal regret bounds may have poor empirical performance due to inappropriate assumptions and large constants associated with regret bounds (explore-exploit schemes)*" and in another one it is mentioned that computing the optimal estimation is often infeasible and approximations are necessary [22].

When copmared to the other methods, Thompson sampling has the best performance given the restriction of fully online processing and therefore is a reasonable choice for the PersuasionAPI.

Chapter 4

Research of the improvement opportunities

4.1 Application settings

Currently there are two main application areas for PersuasionAPI. One of them is personalized content generation on websites, which in practice is implemented in a form of embedding persuasive banners into the images of some selected items in the list of items for e-commerce platforms.

Another case is e-mail marketing. Strictly speaking, this is not a perfect use case for PersuasionAPI since there is no need in the fully online processing. In e-mail marketing the time gap between the interactions with the users is much bigger than compared to the pager reload time in case of the interaction within an e-commerce store, which allows to go for more complicated algorithms as soon as the computational complexity is no longer a bottle-neck. Nevertheless, it is still a valid setting for PersuasionAPI and it had been successfully adjusted to this setting.

In general, persuasion profiling is performed for multiple clients and each client can have multiple campaigns. Due to the private data usage laws in Europe it is prohibited to aggregate user data across campaigns and across clients.

4.2 Improvement directions

This section presents the state-of-the-art of the CMAB problems and the common improvement directions. These directions are critically assessed in terms of their applicability to the PersuasionAPI case and one particular direction is chosen based on this assessment.

4.2.1 Large number of arms

One of the popular improvement directions are CMAB with (infinitely) many arms or strategies [35, 36, 37, 38, 39]. This is an extreme case of a well known CMAB problem when "*one needs to assume extra structure in order to make the problem tractable*" and is a subject of investigation nowadays due to its high practical applications such as online auctions and web advertisement.

The crucial part of most of the models that tackle this problem is the similarity metric space, which has one important property: if two arms are close in this metric space they have similar payoffs. The in-depth discussion of such space and its relation with Lipschitz-continuous maps and contractions lies beyond the scope of this thesis.

Several papers research a general MAB problem, when a metric space describes arms. One of the papers discusses the case when this metric space is implicit and defined by a tree-based structure that represents a classification of arms, but not known numerically [37]. Other papers discuss the case when the metric space is both implicit and defined numerically [38, 39].

Another paper discusses the contextual MAB problem and defines a similarity space containing distances between the context-arm pairs [35]. In this paper adaptive partitioning is used instead of the uniform partitioning of the similarity metric space, which leads to a finer partitioning in high-payoff regions and in popular regions of context space.

Applicability to PersuasionAPI

This improvement direction is irrelevant for the PersuasionAPI case since there are currently only four main persuasion strategies and even the clients that want to get several different messages per strategy hardly go above 15 substrategies.

4.2.2 Context enrichment

Several other papers tackle the problem of the multi-armed bandit context enrichment within recommender systems [5, 6, 7, 40]. In one of the papers the context space and the corresponding similarity metric is introduced in addition to an item space and its similarity metric, which is fairly common for recommender systems [5]. In this work authors implement an item cluster tree that efficiently partitions the item space into K clusters. These clusters represent sets of items that are similar to each other. The recommender system at each time t selects a cluster based on the current context and the history, which is a collection of past contexts, cluster selections and payoff observations. It then recommends a random item from the selected cluster.

The other paper related to the context enrichment within recommender systems introduces an improvement to the *ε-greedy* algorithm that integrates case base reasoning [6]. In this paper a case is described as a pair of a situation occurring while a user is browsing on his mobile device and user preferences in this situation. The suggested two-step approach is to (1) find situations that are similar to the current one and (2) solve the exploration-exploitation problem of CMAB. The context discussed in this paper is limited to location, time and social connection.

A common approach to the context enrichment that can be seen from the papers discussed above is collecting historical data and performing computations on it when the next recommendation needs to be given. One of the papers describes the approach as follows: "*to integrate CBR into each iteration: before choosing the document, the algorithm computes the similarity between the present situation and each one in the situation base; if there is a situation that can be re-used, the algorithm retrieves it, and then applies an exploration/exploitation strategy*" [6]. The other one has a comparable approach: "*given a user's context, our algorithm aggregates its past history over a ball centered on the user's context*" [5].

Applicability to PersuasionAPI

The approaches described above base their estimations on the historical data, which obviously brings more computational complexity. More formally, the time complexity of such algorithms is at least linear *O(N)*, where N is the number of historical events. Unfortunately, no explicit discussion on the exact implementation and its time/space complexity is available to give more information about it, but it is reasonable to assume that such solutions introduce at least linear time complexity, while PersuasionAPI aims at fully online processing and constant *O(1)* time complexity.

The current version of PersuasionAPI algorithm uses the context in a very narrow scope. More formally, the context in terms of the contextual multi-armed bandits problem currently contains only the user identifier, which enables persuasion on an individual level. The context describes what information is used to come up with an estimation of response rates for all the persuasion strategies. In future, additional information about the users can be used to come up with better estimations of the response rates.

Enriching the context of PersuasionAPI can help to go beyond the binary result of the interaction and answer the question of why a user positively responded to the strategy given the context.

However, this contextual data is not (yet) collected and is not available in the historical data set. Any improvements would be purely theoretical and would

be based on simulations using generated data. This would not help to discover trends in user behavior based on the large existing historical data set that is available.

4.2.3 Refinements for specific cases

There is a big number of articles related to the specific application case, for which CMAB is adjusted and refined.

One of the papers shows that bandit algorithms are attractive alternatives to current adaptive treatment allocation strategies in clinical trials [11].

It is also possible to describe CMAB in the setting of online advertising, where ads have limited lifetime [41]. A common assumption in this case is that in such constantly changing environment bandit arms born and die regularly. In [41] the algorithm for both deterministic and stochastic reward functions is presented and the main adjustment is the reduced exploration phase.

MAB problems are also discussed in application to learning a ranking of documents. This task is different from the usual MAB problem when a single best result needs to be computed because a ranking of documents needs to be computed instead [42]. Paper [42] questions the independence between documents for ranking and assumes that the utility of documents is not independent, therefore similarities are taken into account.

Another article tackles the dependencies among arms and presents a framework for exploiting these dependencies in multi-armed bandit problems when the dependencies are in the form of a generative model on clusters of arms [22].

4.2.4 Reinforcement learning model

Planning in reinforcement learning is defined as "*any way of deciding on a course of action by considering possible future situations before they are actually experienced*" [17]. The other paper states that "*planning in reinforcement learning refers to the use of models of the environment to compute value functions and thereby to optimize or improve policies*" [43]. The part that is responsible for planning is the reinforcement learning model that describes the behavior of the environment and is considered optional within the reinforcement learning.

Planning is an important part of modern CMAB research, however to the best of author's knowledge it is not explicitly discussed within the papers and in most cases is taken as granted. Papers researching the problem of infinitely large number of arms [37, 38, 39] assume that the metric space is implicit and defined by a tree-based structure, which can also be defined numerically. This metric space embeds the static information about the degree of similarity

between arms and is a reinforcement learning model that is used for planning. The question of obtaining such a model is not discussed in these papers.

When tackling MAB problems with dependent arms, it is common to assume that all the arms are clustered into K clusters and the dependencies among arms in a cluster are described by a generative model, the form of which is known [22]. This cluster set along with the known dependency model is a static information that is embedded into the reinforcement learning model, however the process of obtaining this information is again not discussed in the paper.

The question of obtaining the reinforcement learning model is an interesting area that most of the times is not explicitly discussed, especially for the constraint that is derived from the requirements, namely low computational complexity. Therefore, building a reinforcement learning model is a valid research topic and is interesting from a research point of view.

Applicability to PersuasionAPI

Currently, the PersuasionAPI algorithm works as a pure reinforcement learning system in the sense that it learns by interacting with the environment and no historical data is used in the system. There are certain regulations that restrict private data usage, in the case of PersuasionAPI, these forbid to accumulation of user data across clients based on private data, such as e-mail address or driving license number. For instance, it is not allowed to accumulate historical data representing users' response behaviors to different persuasion strategies within one campaign/experiment and then use this data in another campaign/experiment, drawing a conclusion the two users from the two experiments represent one person based on the private data stored in their profiles that can identify them. All in all, strict European policies related to data privacy mean that any improvements that involve historical data usage should be carefully checked against these policies.

That being said, there are still opportunities to improve the algorithm by building a reinforcement learning model. One of the improvements proposed in this thesis is to build a model that embodies the static knowledge that is proven to be true independently from the information obtained from a particular experiment or campaign.

Example There is a hypothesis stating that the more times a user is presented with the same persuasive strategy consecutively, the more his response rate drops for this particular strategy. In simple terms, the user expects the items mentioned in the persuasive message (authority, scarcity, social proof, and other strategies) to be rare, and when they occur one after another he becomes increasingly suspicious and the probability of optimal behavior of this

user drops. A model based on this knowledge will somehow decrease the number of consecutive advises with the same strategy given to the same user. In the simplest case, such a model can contain a rather straight forward rule that forbids choosing the same strategy that was chosen for the previous interaction. Note that this most probably would not be the best strategy as it does not perform well in the case where the response rate for one strategy is much better than for all the others and the algorithm will still advise the clearly optimal strategy only 50% of the times based on the inflexible rule in the model.

4.2.5 Improvement direction choice and motivation

After several state-of-the-art directions of the improvement of CMAB methods were examined and critically assessed in the sections above, building a reinforcement learning model was chosen as the most perspective direction due to several factors:

- **Research aspect**: this is not a commonly addressed problem in the literature and in many cases static reinforcement learning model is taken as a prerequisite [22, 37, 38, 39].
- **Business aspect**: embedding a static reinforcement learning model is more of an incremental innovation rather than a disruptive one and does not require the whole algorithm to be re-implemented from scratch. This means faster results to the business with minimal investments.
- Embedding static information that enables better planning fits well into all the requirements, especially the fully online processing requirement and does not have a negative effect on computation complexity.

4.3 Process

The process of identifying new and promising ways to improve the existing PersuasionAPI algorithm is proposed according to the following steps:

1. Build a hypothesis based on initial knowledge, psychological and marketing sources of knowledge, and expert opinions.
2. Collect and clean the appropriate historical data from PersuasionAPI infrastructure and transform it to serve the purpose of validating a particular hypothesis.
3. Mine the data gathered during the previous step to discover the trends that will validate or invalidate the hypothesis.
4. In the case where the hypothesis is valid, suggest on the PersuasionAPI algorithm improvement that incorporates the phenomenon proven to exist by the hypothesis' validation.

The process described above is part of the overall data-driven improvement process described in Chapter 8 and was applied to validate the two hypotheses of this thesis, which both fall into the category of building a reinforcement learning model. This area is considered to be both challenging from a research point view and valuable from the business perspective, in that it answers the question of whether or not it is a promising direction for further algorithm improvement in future.

4.4 Methods

The method used to test the validity of the hypotheses is data-driven validation, which requires the use of real data that need to be retrieved and mined before the actual validation can be performed. This section describes the data retrieval and mining steps, which were similar or identical for all the hypothesis validations.

PersuasionAPI data is stored in a MongoDB database, and has two logical parts. One part contains the most up-to-date information for each user and is accessed whenever it is necessary to get an advice. The other part contains all the historical data and has the role of an event log. This event log contains the information about the so-called "random" user group for which the next strategy is chosen randomly. The latter is needed to test the hypothesis. The data used for this purpose are fused from 20 clients and contain almost 82 million interactions with 6.7 million users. The process of data retrieval is described in the next subsection.

User ID	Strategy ID	Success
"u34211"	0	0
"u44566"	3	1
"u01032"	2	0
"u34211"	2	0
...		
"u44112"	1	1

Table 4.1: The initial snapshot of historical data

Data retrieval

Data retrieval was coded in Java. With the help of MongoDB Java driver, all the necessary data was retrieved from the database in a raw format, converted into "comma separated values" (CSV) format and stored as a file (see Table 4.1). This approach ensures the following:

- Unlike querying a live database, there is a single source of data used for

all the mining procedures for different hypotheses and applying the same stable algorithm to this data source will always give the same result.

- All the mining procedures are independent from the network connection with MongoDB server.
- There is no need to perform the same computations again to build a base data set.

Chapter 5

Hypothesis 1: repetition effect

5.1 Motivation

This hypothesis has its roots in the field of human psychology. There are certain differences in the recognition of the novel things as opposed to the familiar ones. More formally, the "novelty/encoding hypothesis" and the experiments related to it show that "*accuracy of explicit (episodic) recognition was higher for novel than for familiar words*" [44, 45].

In the case of PersuasionAPI, this results in a lower accuracy of recognition for familiar persuasive sentences representing the same strategy, which may result in a lower response rate. Given PersuasionAPI's context, there are usually several interactions within one session. This means that the same strategy used in two or more consecutive interactions may result in a response rate decrease.

This is being tested within the first hypothesis already described as an example in the previous chapter and, in short, states the following: the more times a user is presented the same persuasive strategy consecutively the more his response rate drops for this particular strategy.

5.2 General response rates comparison

Firstly, data was retrieved, collected, and mined as described in the previous section so that it is ready for the specific tests.

Based on the data stored as a file, it is possible to compute various statistics that can validate or invalidate the initial hypothesis. During the validation, two statistics need to be compared: the average decrease in response rate when one

strategy is presented to a user on two consecutive interactions and the average decrease in response rate when two different strategies are presented to a user one after the other. Note that due to the low average number of interactions per user and the necessity of the exploration phase the case of the same strategy chosen consecutively three or more times was much smaller and therefore was left out of the tests.

Another useful statistic that was computed prior to the actual validation was the total number of interactions per contact, starting from the first contact with the user. It turned out that the largest number of interactions occur during the first and second contacts with the user. This means that the whole validation can be based on the first two interactions and can represent the overall trend as it is more reliable because there are more users participating, which gives more reliable averages of the response rate.

The next step was to compute the averages for the first and the second interactions in the case of "paired" strategies (meaning that the same strategy was used consecutively) and "not paired" ones across all users. This was done with a Java program, and the results are presented in Table 5.1.

Response rate (1^{st} interaction)	Response rate (2^{nd} interaction)	Sample size
Paired		
0.0616329987	0.0432507665	152321
Not paired		
0.0615868408	0.0438910146	454740

Table 5.1: Results of the experiment of comparing paired and not paired strategies used consecutively during the 1^{st} and the 2^{nd} interactions over all users

Based on these results, it is possible to compare the proportions of the decreases of the response rates for paired versus not paired strategies:

$$\hat{P}_p = p_{p2} - p_{p1} \approx 0.01838 \tag{5.1}$$

$$\hat{P}_{np} = p_{np2} - p_{np1} \approx 0.01770 \tag{5.2}$$

To do that, a hypothesis test was chosen with the two-proportion z-test procedure. The null hypothesis is that the two proportions are equal:

$$H_0 : P_p = P_{np} \tag{5.3}$$

The alternative hypothesis represents the alternative that needs to be (in)validated and is chosen to state that the two proportions are not equal:

$$H_A : P_p \neq P_{np} \tag{5.4}$$

Following the two-proportion z-test steps, the proportion of successes in two samples combined was computed as well as its squared error:

$$\hat{P} = \frac{\hat{P}_p * N_p + \hat{P}_{np} * N_{np}}{N_p + N_{np}} = 0.01787 \tag{5.5}$$

$$SE = \sqrt{\hat{P} * (1 - \hat{P}) * \left(\frac{1}{N_p} + \frac{1}{N_{np}}\right)} = 0.000000129^{1} \tag{5.6}$$

The test statistic (z-score) was computed based on the previous products:

$$z = \frac{\hat{P}_p - \hat{P}_{np}}{SE} \approx 5271.3178 \tag{5.7}$$

Formally, at this point the P-value needs to be computed for both inequality situations:

$$P = P(Z \leq -5271.3178) + P(Z \geq 5271.3178) \tag{5.8}$$

The z-score that was obtained in this comparison is large and far beyond the usual $[-4, 4]$ interval. Given the fact that even a z-score 4 would fit into 0.01 significance value, it can be stated that the difference between P_p and P_{np} is significant.

However, this test rejects the idea that the two probabilities are exactly the same, which does not necessarily mean that the difference is worth a closer look. The reason for rejecting this hypothesis is the comparatively large number of observations N (order of 10^5). This case is described in [46], which states that when N is large, "*virtually any parsimonious parametric model [..] will be strongly rejected by any standard hypothesis test using the usual confidence intervals. Virtually all specific null hypotheses will be rejected using present standards*".

In this case, the results of this experiment show that there is a significant difference between the case when the same strategy is used consecutively and the case when two different strategies are used consecutively. However, this is a far lower level of difference than was expected. Since the result obtained after this experiment lacks certainty, it is reasonable to repeat the same experiment for the individual strategy level.

[1] Strictly speaking, with the chosen precision of 5 digits after the decimal point $SE = 0.00000$.

5.3 Individual strategy level response rates comparison

Despite the discussion in the previous section there is still a question of whether or not the high-level results obtained in the previous comparison can characterize what is going on at the individual strategy level. Since all 20 clients used the same four distinctive persuasive strategies, it is possible to run the same experiment on the strategy level to see whether or not the same trend that was clear during the previous experiment is proven for each strategy.

In this case, the response rates for paired strategies were computed for each strategy separately and after that this decrease was compared to the decrease for not paired strategies computed in the previous comparison. The results of the next experiment are presented in Table 5.2. Note that the same experiment was repeated for all the pairs of interactions where the same strategy was chosen repeatedly and it shows the same trends as the ones in Table 5.2.

RR_1[2]	RR_2	Sample size	Z-score	Sign.[3] 0.05	Sign. 0.01
Strategy 1					
0.063141237	0.0412765396	37915	5.8633	Pass	Pass
Strategy 2					
0.0634088825	0.0478710861	38165	-3.0857	Pass	Pass
Strategy 3					
0.0610477398	0.0411121673	37872	3.1613	Pass	Pass
Strategy 4					
0.0589707493	0.0427290265	38369	-2.0809	Pass	Fail

Table 5.2: Results of the experiment of comparing paired and not paired strategies used consecutively during the 1^{st} and the 2^{nd} interactions for each strategy

5.4 Conclusion

The results of both experiments show several trends. First of all, there is an overall trend of decreased response rate on the next interaction compared with the previous one, and the small overall the difference between the decrease rates for paired and not paired strategies.

In the second experiment, the same metrics were computed for each strategy separately, and this granularity level adjustment gave a more detailed view of

[2] A shortcut for the Response Rate (first interaction).
[3] A shortcut for significance level.

the data set with respect to the hypothesis being tested. Table 5.2 shows that the relative decrease varies from strategy to strategy and even changes its sign: for strategies 1 and 3 there is an increase of the response rate for the paired case compared with the not paired case. This fact proves the hypothesis of the negative effect of consecutively used strategies on the response rate to be wrong at least for strategies 1 and 3. In general,Table 5.2 shows a positive effect for some strategies and a negative effect for the other strategies, which makes it hard to predict whether or not there is a negative effect for the new strategy a new client may decide to implement.

Overall, the effect discussed above is neither stable nor predictable for future strategies, which makes it impossible to embed any static rule into a reinforcement learning model, as was planned. The hypothesis in its initial form is invalidated.

Chapter 6

Hypothesis 2: correlations between strategies

6.1 Motivation

The need for cognition, or the tendency for the individual to engage in and enjoy thinking, is first mentioned in [47]. This study showed that people can be clustered into two groups, with need for cognition being high or low. The corresponding research revealed an important experimental result that is relevant for the field of persuasion profiling. In their previous publications authors described the persuasion process "*as either one in which message recipients actively process the arguments presented in a communication or one in which the message arguments are virtually irrelevant to persuasion, since attitude change results from various noncontent cues in the situation*". In [47] they called these two processes as the central and peripheral routes: the latter opens opportunities for persuasive communication where the persuasion setting (such as speed of speech or expertise) predominates, whereas the former operates when the message content is processed rationally.

These authors stated that the extent to which recipients are motivated by their need for cognition to think about the issues they confront may determine the route (either central or peripheral) – in other words their individual vulnerability to persuasive messages [47, 48]. This statement leads to a hypothesis that people with low need for cognition will be more vulnerable to any persuasive message. This can be validated by checking correlations between the success probabilities for each pair of strategies.

Formally, the second hypothesis as well as the first one aims at building a reinforcement learning model and focuses on the mutual dependencies between the success probabilities of different strategies. As before, the data were retrieved, collected, and mined in preparation for the specific tests.

The next step was to compute the correlation coefficients between different strategies. There are several possible approaches to do that and two of them (Pearson's correlation coefficients and contingency table) are discussed in this chapter.

6.2 Correlation: Pearson's correlation coefficients

The first step was to rebuild the data set from a set of event records into a set of user records with the success probabilities of four strategies related to each user. Simply put, the outcome of the transformation was a data frame: each row represented a user and each column represented a strategy. The actual transformation can be conceptually represented like this:

$$(U_i, S_{ij}, scs_{ij}) \Longrightarrow (U_i, [\hat{p}_{i1}..\hat{p}_{iN}]), \tag{6.1}$$

where i identifies a specific user, j identifies a specific event record, S stands for strategy, scs represents success or failure of the interaction and p_{ik} is the average success probability of strategy k for user i.

R language was chosen during previous steps because of its simplicity and implementation speed given the task of transforming one data frame into another and performing some additional computations along the way. However, R implementation yielded a very poor performance, which resulted in hours of computations to transform a data set of 426 Megabytes (MB). The initial goal was to choose an environment that would allow rapid prototyping, and R was suitable for prototyping statistical and learning algorithms; however, as it turned out, it is not a good environment for performing heavy computations on big data sets.

Besides, this transformation has a high parallelization coefficient, which means that every event record can be processed independently. Spark technology using Python language was chosen as a more suitable environment for the purpose of fast parallel computation. Python, in this case, is used as a functional programming language and the whole transformation process is implemented in a chain of operations on Resilient Distributed Datasets (RDDs).

The method used for computing correlation comprised two main steps. The first step was to estimate success probabilities for each strategy and each individual, averaged over all interactions. This gives a matrix where each row represents an individual user and each column represents a strategy, and each cell contains an estimation of success probability. The second step was to compute correlations using Pearson's coefficients between each pair of matrix columns.

	S_1	S_2	S_3	S_4
S_1	1.0	0.139886313658	0.139617173946	0.139346015849
S_2	0.139886313658	1.0	0.14002336345	0.140130992547
S_3	0.139617173946	0.14002336345	1.0	0.13843020816
S_4	0.139346015849	0.140130992547	0.13843020816	1.0

Table 6.1: Correlation coefficient between different strategies (version 1)

Both steps were implemented in Spark and Python in its functional flavor. Pearson's coefficients computation was implemented from scratch in a distributed environment (the main equation used was equation 6.2) and then basic statistics library *MLib* was used to validate the custom-made computation method. These results were delivered amazingly fast, in under 2 minutes.

$$R_{xy} = \frac{N\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{N\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}\sqrt{N\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}} \tag{6.2}$$

The results are presented in Table 6.1 and show a positive correlation between all the pairs of strategies.

Although these coefficients show some correlation, it is not as large as expected. To understand whether or not the implementation of the method affected the outcomes, several checks and tests were performed.

Since the functionality that computed the correlation coefficients was checked against MLib implementation and proven to work correctly by returning identical results, the second test focused on the correct data vectors formed for the correlation computation. It turned out that it was not completely correct as soon as in the implementation there was no difference between 0.0 as an average response rate and 0.0 as a signal of no interactions for this strategy-user pair. For the purposes of handling missing data, pairwise deletion was used [49]. After that, the correlation coefficients were recomputed, which yielded a higher correlation for all pairs of strategies (see Table 6.2). In Figure 6.1, correlations are visualized by plotting the success probabilities for each pair of strategies on scatter plots. The complete Python script for Spark is presented in Appendix C.

The final step after computing Pearson's correlation coefficients is to formulate the null and the alternative hypotheses and to perform significance testing:

$$H_0 : R_{xy} = 0 \tag{6.3}$$

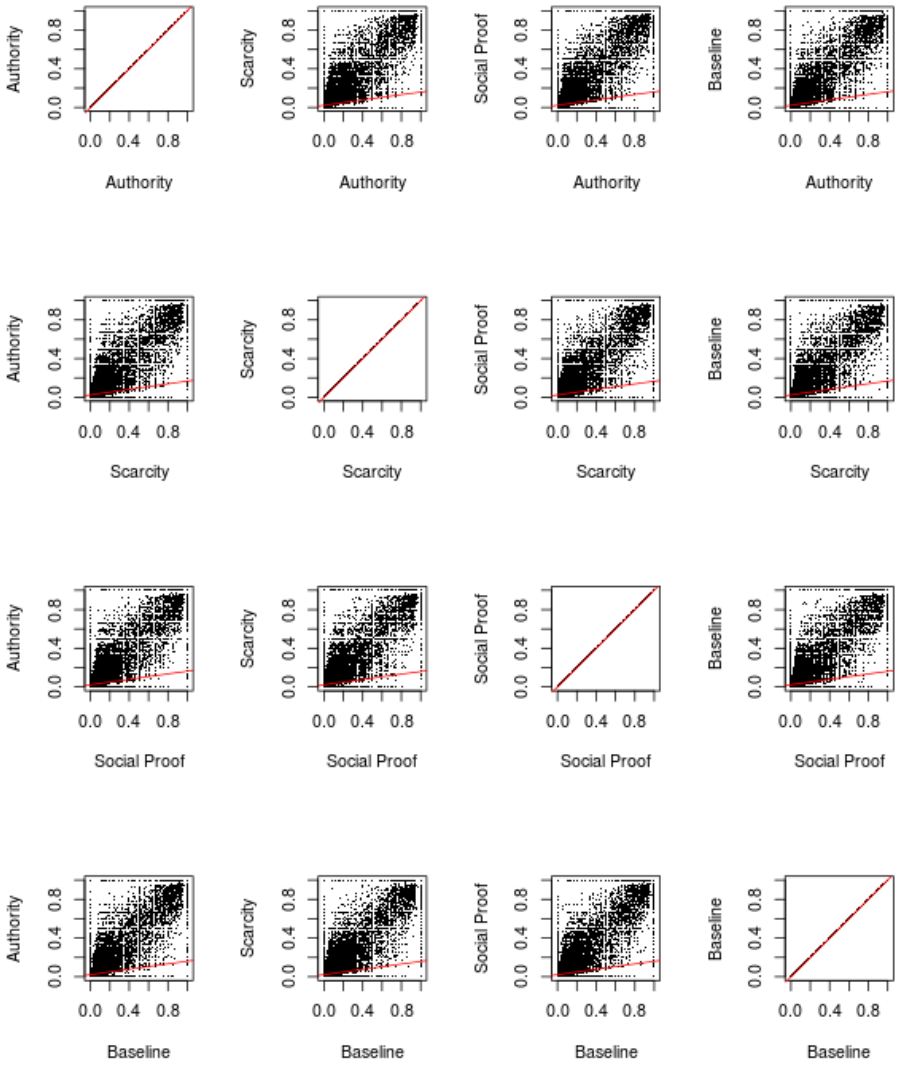

Figure 6.1: Scatter plots of the correlations between the success probabilities for each pair of strategies

	S_1	S_2	S_3	S_4
S_1	1.0	0.247503734044	0.251313654705	0.250928335387
S_2	0.247503734044	1.0	0.251798737338	0.251542743559
S_3	0.251313654705	0.251798737338	1.0	0.250587845746
S_4	0.250928335387	0.251542743559	0.250587845746	1.0

Table 6.2: Correlation coefficient between different strategies (version 1)

	S_1	S_2	S_3	S_4
S_1	X	324042	323265	323696
S_2	324042	X	323837	324090
S_3	323265	323837	X	323563
S_4	323696	324090	323563	X

Table 6.3: Numbers of observations for each pair of strategies

	S_1	S_2	S_3	S_4
S_1	X	140.8751	142.8716	142.7476
S_2	140.8751	X	143.2739	143.1842
S_3	142.8716	143.2739	X	142.5247
S_4	142.7476	143.1842	142.5247	X

Table 6.4: T-values corresponding to the correlation coefficients

$$H_A : R_{xy} \neq 0 \tag{6.4}$$

The corresponding test statistic is presented in equation 6.5.

$$t = R_{xy} \cdot \frac{\sqrt{N-2}}{\sqrt{1-R_{xy}^2}} \tag{6.5}$$

The numbers of observations for each pair of strategies are presented in Table 6.3 and the corresponding t-values are presented in Table 6.4. Such big t-values correspond to very small P-values, which means that the null hypothesis is invalidated. Now it can be stated that there is a significant correlation between each pair of strategies.

6.3 Correlation: contingency table

Contingency tables and methods of its analysis present another option for computing the correlation between strategies. A contingency table is a matrix-format table that shows the frequency distribution of the variables. In the case of PersuasionAPI, it is a table for two binary variables where each represents a strategy and can have a true/false value (see Table 6.5 for the observed values).

	$S_i = True$	$S_i = False$	S_i total
$S_j = True$	N_{11}	N_{10}	$N_{1\bullet}$
$S_j = False$	N_{01}	N_{00}	$N_{0\bullet}$
S_j total	$N_{\bullet 1}$	$N_{\bullet 0}$	N

Table 6.5: Contingency table for two strategies

Chi-square statistics											
	Non-significant								Significant		
SL	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83

Table 6.6: Chi-square distribution table

The next step is to compute Chi-square statistics, which in a nutshell shows how far the actual observed values lie from the expected case when the success probabilities are equal (see 6.6).

$$\chi^2 = \frac{N(N_{11} * N_{00} - N_{10} * N_{01})^2}{(N_{\bullet 1} + N_{\bullet 0} + N_{0\bullet} + N_{1\bullet})} \tag{6.6}$$

The last step is to use the Chi-square distribution table to determine the significance of computed values. Since the purpose is to find the correlation between two variables, there is one degree of freedom and, in this case, the relevant part of the Chi-square distribution table is presented in Table 6.6.

The problem with method is its complexity when applied to the case of PersuasionAPI. In the situation where there are multiple interactions per user, this method should be changed: for each user all pairs of the responses for the two strategies should be retrieved and inserted into the contingency table. This method is shown as an alternative to the one performed in the previous section.

6.4 Conclusion

Overall, the correlation coefficients presented in Table 6.2 show that all the strategies are not independent and there is a positive correlation between each pair of strategies. The fact that there are correlations between each pair of strategies is also mentioned in [50], where the authors drew this conclusion based on data from several experiments. This means that it is possible to give a better prediction of user response rate for strategy i if there is information about the user response rate for strategy j.

Chapter 7

Prototyping and implementation

In the previous two chapters, two particular hypotheses have been analyzed and discussed. The hypothesis about the negative effect of repetition was invalidated. The hypothesis about the correlation between strategies was validated. This chapter describes the actual implementation of the improved algorithm that follows hypotheses' validation.

7.1 Preparing the data

For validation and testing purposes, historical data needed to be transformed from the general data snapshot retrieved in Section 4.4 (see Table 4.1) into a suitable data frame that contained for each event/interaction the user id, the average success probabilities for each strategy (precomputed from historical data and not known in real life), draws from each of the Bernoulli distributions specified by these probabilities and the index of optimal strategy in terms of multi-armed bandit problems (see Table 7.1). This is the index of the strategy with the maximal average success probability, which is obviously not known in real life and is used to compute the cumulative regret metrics.

User ID	p_1	p_2	p_3	p_4	o_1	o_2	o_3	o_4	Best (index)
"u34211"	0.00	0.10	0.31	0.02	0	0	0	0	2
"u44566"	0.30	0.00	0.10	0.00	1	0	0	0	0
"u01032"	0.10	0.11	0.01	0.23	0	0	0	0	3
"u34211"	0.00	0.10	0.31	0.02	0	0	1	0	2
				...					
"u44112"	0.00	0.20	0.00	0.01	0	0	0	0	1

Table 7.1: The data frame for algorithm validation and testing

Draws from the distributions are needed to simulate user behavior and give a response (either success or failure) for each particular strategy and for each particular interaction. Note that embedding these draws into the data frame is crucial since it makes all the tests based on this data independent from the random sampling factor. In other words, any algorithm that will be tested on these data and that will advise strategy S to user U during interaction i will receive the same simulated response from this user.

This data frame was constructed via a Python script executed on Spark, which is an improved version of the script from the second hypothesis validation (Chapter 6). The statistics package *Stat* from the *MLLib* library was used for sampling from the distributions.

One of the features worth mentioning that was implemented in the script is filtering based on the number of interactions. The task was to exclude the users with more than N interactions from the data frame. The first step was to get all the users that have more than N interactions:

```
outliers = logData.map(lambda x: x.split(',')).map(lambda x: (x[0],
    1)).reduceByKey(lambda a, b: a + b).filter(lambda (k, v): v >
    5).cache()
```

The second step was to filter out these users from the main data frame:

```
reducedLogData = logData.map(lambda x : x.split(',')).map(lambda x:
    (x[0], x)).subtractByKey(outliers).cache()
```

Finally, it was necessary to transform the data frame into a one with composite keys (user id and strategy id):

```
reducedLogData = reducedLogData.map(lambda (k, x) : (x[0] + ',' + x[2],
    x[1])).cache()
```

7.2 Implementation of the current version

The current implementation of the core PersuasionAPI algorithm is in Java and is distributed across several classes. It is not flexible and is not suitable for rapid prototyping and monitoring the results, which is crucial when it comes to the algorithm's improvement. Therefore, there was a strong need for an environment that would enable rapid prototyping and monitoring of the results. Due to lack of documentation and redundant Java code, it is far from self-explanatory and this task was not a straight forward porting of the algorithm into a prototyping environment; rather, the algorithm was built from scratch based on the main design choices described in Chapter 3.

The current version of the algorithm has been implemented within the research and the main class is presented in Appendix B. *Apache Commons Math3* library was used for implementing the Thompson sampling.

The next step was to embed the knowledge obtained from the validation of the hypothesis into the new version of the algorithm.

7.3 Success probability estimation method redesign

This section describes the prototyping of the new version of the PersuasionAPI algorithm based on the hypotheses validation and analysis performed in Chapters 5 and 6 and illustrates the possible ways of bringing these results to life by embedding them into the next version of the PersuasionAPI algorithm.

Since the first hypothesis is invalidated it gives a "no-go" to possible extensions of the algorithm that aim at re-ordering sequences of the same strategies presented consecutively.

The second hypothesis was validated and is in the implementation phase. Since the current version of PersuasionAPI algorithm is already implemented and tested in the prototyping environment, the next step is to embed the knowledge about correlation between pairs of strategies into the algorithm and see how it performs against the current version. There are two ways to implement this: one is a full hierarchical model and the other one is a reasonable heuristic.

The full hierarchical model [51] measures hierarchical effects that occur "*when predictor variables are measured at more than one level*" [52]. Previously hierarchical effects were covered by the Stein's shrinkage estimator heuristic (equation 3.5). The full hierarchical model is computationally very demanding as compared with the heuristic approach and since one of the restrictions for PersuasionAPI implementation is giving out the advice completely online and avoiding computational complexity, implementing the full hierarchical model is not recommended in this case. Instead, one can think of another heuristic that can model the mutual correlation between the strategies. This section focuses on possible heuristics that can improve the existing algorithm and embed the proven fact of mutual correlation between strategies.

Although the exact production model and implementation require deeper research, during this research several heuristics were offered to cover the mutual correlation effect. One of the important assumptions embedded in these heuristics is that the correlation between each pair of strategies remains constant over time and, thus, can be modelled with the constant coefficients Cor_{ij}. One of the first heuristics that was tested was the one presented in equation 7.1

$$\hat{p}_{is} = \sum_{j=1}^{S} \frac{n_j}{N} Cor_{js}(B_{ij}P_j + (1 - B_{ij})p_{ij}), \tag{7.1}$$

where P_j is the average probability of success over all the users for the particular strategy j, p_{ij} is the probability of success for the user i for the particular strategy j, B_{is} is the shrinkage factor introduced earlier, Cor_{js} is the correlation coefficient, and $\frac{n_j}{N}$ is a normalization coefficient.

For a simple two-strategy case, equation 7.1 will look like this:

$$\hat{p}_{i1} = \frac{n_1}{N} Cor_{11}(B_{i1}P_1 + (1 - B_{i1})p_{i1}) + \frac{n_2}{N} Cor_{12}(B_{i2}P_2 + (1 - B_{i2})p_{i2}), \tag{7.2}$$

where Cor_{11} as well as any Cor_{ii} are equal to 1.

However, two problems were spotted in this heuristics. Firstly, it is not properly balanced. Consider a case of two strategies with four and six observations in total for each strategy. Assuming that both probabilities $\hat{p}_{is}$ are estimated as 1.0 and $Cor_{12} = 0.24$:

$$\hat{p}_{i1} = \frac{n_1}{N} Cor_{11}\hat{p}_{i1} + \frac{n_2}{N} Cor_{12}\hat{p}_{i2} = 0.4 \cdot 1.0 \cdot 1.0 + 0.6 \cdot 0.24 \cdot 1.0 = 0.544 \tag{7.3}$$

This approximation is far from the expected 1.0 and thus is poor. This happens due to improper balancing since the chosen normalization coefficients do not add up to 1. The proper normalization coefficients should add up to 1 and are shown in equation 7.4.

$$\hat{p}_{is} = \sum_{j=1}^{S} \frac{n_j}{N} \cdot Cor_{js} \cdot \frac{1}{T}(B_{ij}P_j + (1 - B_{ij})p_{ij}) \tag{7.4}$$

T is the normalization sum that ensures proper normalization (see equation 7.5).

$$T = \sum_{j=1}^{S} \frac{n_j}{N} \cdot Cor_{js} \tag{7.5}$$

Secondly, the normalized coefficients do not prioritize the observed current strategy for which the success probability is being estimated based on some measure of certainty. In other words, if the number of observations for strategy x is much bigger than the same number for strategy y, then the approximation of the success probability for client i for strategy y will be heavily biased by strategy y regardless of the fact that there are significant data collected about strategy x. Consider a case of two strategies with 10 and 990 observations in total for each strategy. Assuming that $\hat{p}_{i1} = 0.9$, $\hat{p}_{i2} = 0.1$ and $Cor_{12} = 0.24$:

$$\hat{p}_{i1} = \frac{n_1}{N} \cdot 1 \cdot \frac{1}{T} \cdot \hat{p}_{i1} + \frac{n_2}{N} \cdot 0.24 \cdot \frac{1}{T} \cdot \hat{p}_{i2} = 0.036 + 0.096 = 0.132 \tag{7.6}$$

This approximation is heavily biased toward the estimated success probability of strategy 2, although there have been significantly many interactions within strategy 1 already.

Based on the design choices described in Chapter 3, it is reasonable to replace a weight coefficient $\frac{n_j}{N}$ with a second level shrinkage estimator, which changes equation 7.1 to the following equation:

$$\hat{p}_{is} = \left(1 - \frac{1}{\sqrt{n_s}}\right) \cdot \frac{1}{T} \cdot (B_{is}P_s + (1 - B_{is})p_{is}) + \tag{7.7}$$
$$\frac{1}{\sqrt{n_s}} \sum_{j=1, j!=s}^{S} Cor_{js} \cdot \frac{1}{T} \cdot (B_{ij}P_j + (1 - B_{ij})p_{ij})$$

7.4 Prototyping the improved version

The heuristics discussed above cover the method of probability estimation and do not impact the other parts of the algorithm. Therefore, all the code that covers this method was encapsulated in the *Estimation* interface implementations (see listing 7.1). The current implementation is given in listing 7.3.

Listing 7.1: Success probability estimation interface

```
1  package papi;
2
3  public interface Estimation {
4    public double getEstimation(Double[] currentP, Integer[] currentN,
         double[] P, int[] N, int strategyID);
5  }
```

Listing 7.2: Current implementation of the success probability estimation

```
1  package papi;
2
3  public class BasicEstimation implements Estimation {
4
5    @Override
6    public double getEstimation(Double[] p, Integer[] n, double[] P, int[]
         N, int strategyID) {
7      double shrinkage = 1 / Math.sqrt(n[strategyID]);
8      return(shrinkage * P[strategyID] + (1 - shrinkage) * p[strategyID]);
9    }
10 }
```

Listing 7.3: The new improved implementation of the success probability estimation

```
package papi;

public class CorrelationEstimation implements Estimation {

    @Override
    public double getEstimation(Double[] p, Integer[] n, double[] P,
        int[] N, int strategyID) {
      int strategiesNumber = p.length;
      double[][]cor = {{1.0, 0.247503734044, 0.251313654705,
          0.250928335387 },
         {0.247503734044, 1.0, 0.251798737338, 0.251542743559},
         {0.251313654705, 0.251798737338, 1.0, 0.250587845746},
         {0.250928335387, 0.251542743559, 0.250587845746, 1.0}};

      double shrinkage = 1 / Math.sqrt(n[strategyID]);
      double shrinkage2level = 1 / Math.sqrt(N[strategyID]);

      double sum = (1 - shrinkage2level);

      for (int h = 0; h < strategiesNumber; h++) {
        if (strategyID != h) {
          sum += shrinkage2level * coef * cor[h][strategyID];
        }
      }

      double estimatedP =
          (1 - shrinkage2level) / sum * (shrinkage * P[strategyID] + (1 -
              shrinkage) * p[strategyID]);

      for (int h = 0; h < strategiesNumber; h++) {
        if (strategyID != h) {
          double shrinkage2 = 1 / Math.sqrt(n[h]);
          estimatedP += shrinkage2level * coef * cor[h][strategyID] / sum
              * (shrinkage2 * P[h] + (1 - shrinkage2) * p[h]);
        }
      }
      return estimatedP;
    }
}
```

7.5 Testing

After the new prototypical version of the PersuasionAPI algorithm is implemented, it is time to test it against the current version. Before doing that, it is also useful to compare both methods with other methods, such as Individual

Mean and Grand Mean. Note that this comparison was already performed on test data [15], but never on a large volume of historical data. For this purpose, both Individual Mean and Grand Mean methods were implemented in the prototypical Java environment described in the previous sections.

The main comparison metric is cumulative regret (see equation 7.8). In the case of PersuasionAPI, it is simply the sum of all the differences between the binary outcomes of playing the optimal strategy and the strategy suggested by the algorithm:

$$\sum_{n=1}^{t}(R(S^*) - R(S)) \tag{7.8}$$

In the code, this metric is computed at each step and then stored in a list, one entry per step (see listing 7.4).

Listing 7.4: Cumulative regret computation

```
1 List<Integer> cumulativeRegret = new LinkedList<Integer>();
2 for (Event e : events) {
3     [...]
4     int base = cumulativeRegret.size() > 0 ?
          cumulativeRegret.get(cumulativeRegret.size() - 1) : 0;
5     cumulativeRegret.add(base + idealOutcome - outcome);
```

Such cumulative regret vectors were produced for the current PersuasionAPI algorithm, Grand Mean, and Individual Mean and then plotted on the same figure with a log scale on x axis (see Figure 7.1).

The results of this comparison repeat the outcomes described in [15]; that is the Grand Mean performs worse than the other two methods and PersuasionAPI has the best performance of the three methods, until it is eventually outperformed by the Individual Mean method, when the number of observations grows.

During the next experiment the new improved version of the algorithm was executed against the current one and Individual Mean and Grand Mean methods base on the data for 8 clients. Each method was executed 10 times and then the average performance was taken into account for each client. This data was once again averaged across clients and is presented in Figure 7.2. It shows how the new version of PersuasionAPI algorithm outperforms the current one on the range (4000, 30000) and is performing virtually identical to the current version on the range (1, 4000).

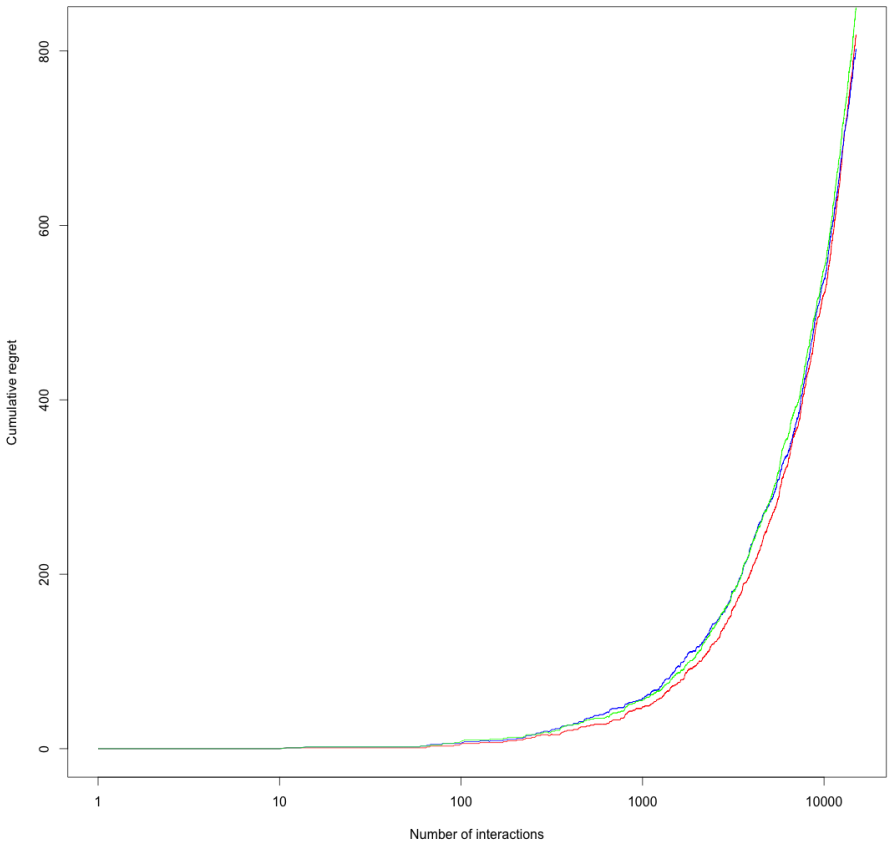

Figure 7.1: Comparison of the PersuasionAPI algorithm with Individual Mean and Grand Mean

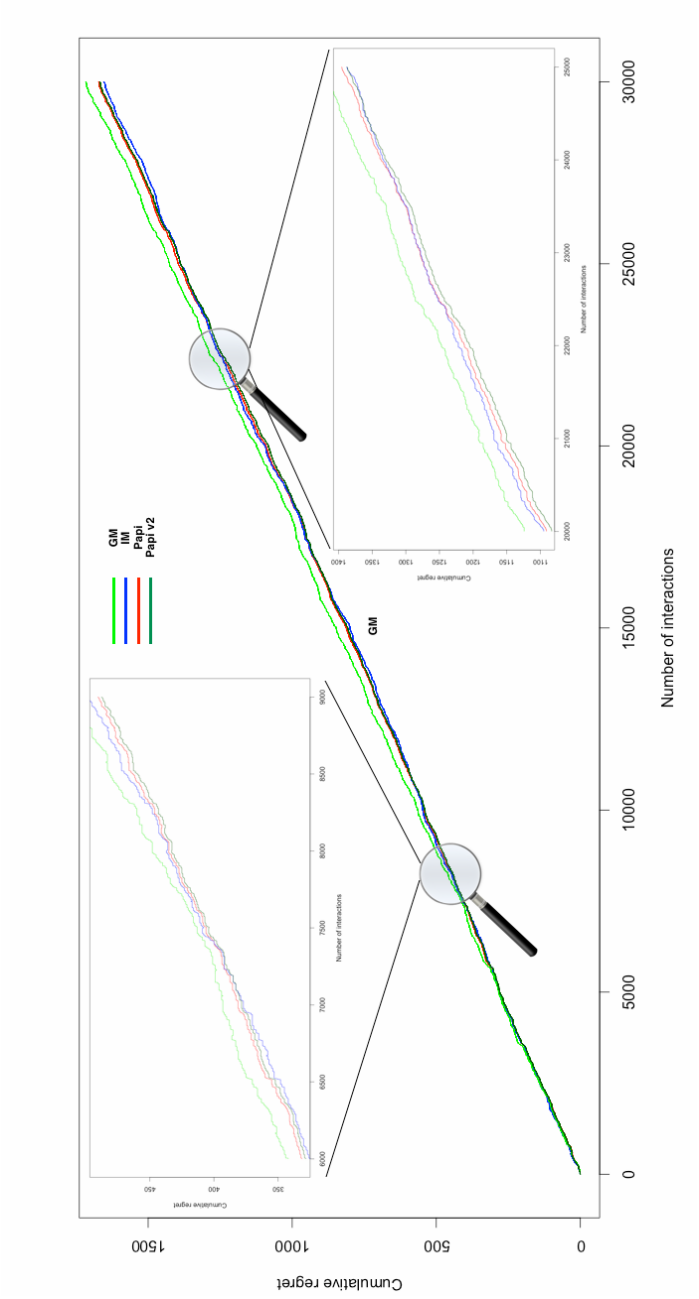

Figure 7.2: Comparison of the PersuasionAPI algorithm with the new improved version, Individual Mean and Grand Mean

7.6 Complexity analysis

The new improved version of the algorithm is able to perform two types of actions – giving an advice on which strategy to use and updating the user profile based on the received information. For the complexity analysis it is necessary to define the main variables: number of strategies S, number of users U and number of interactions for a particular user till time t I_{ut}. In terms of Big-O notation, the time complexity of the method that gives the advice on which strategy to use depends only on the number of strategies and has a complexity of ***O(S)*** and, since the number of strategies is constant, the complexity is ***O(1)***. The method that updates the profile has a complexity of ***O(1)***.

The state-of-the-art methods that were discussed earlier deliver better estimations and have theoretically proven regret bounds, but this quality comes at a price of increased computational complexity. The state-of-the-art methods [53, 54, 55] that guarantee optimal regret bounds have polynomial and even exponential complexity and often introduce complex distributions, sampling from which is very computationally demanding.

Strictly speaking, there is no proven theoretical guarantee of the regret bound for the proposed algorithm as well as for the original PersuasionAPI algorithm, and this proof is a valid follow-up research question. Nevertheless, the empirical evidence is shown for a log-linear regret bound for the original algorithm in [15] and the proposed improved algorithm showed a better performance during the simulations and therefore it is reasonable to assume that the same regret bound holds for the new improved version.

7.7 Conclusion

During the prototyping and implementation phases of the research the initial data frame was transformed into one suitable for testing purposes and several heuristics for the mutual correlation between strategies were suggested and critically evaluated. After that, the part of the algorithm that needed to be changed was located and encapsulated into a separate interface. The new improved version of the algorithm was implemented as well as the more orthodox methods (Individual Mean and Grand Mean). After that, these methods were tested against each other using cumulative regret as the primary performance metric. The new version of the algorithm outperformed the current version on a wide range of observations.

Chapter 8

The data-driven improvement process of PersuasionAPI

Apart from testing two particular hypotheses and implementing the second one into a new improved version of the PersuasionAPI algorithm, a more general outcome was achieved as a result of the thesis, which is the tested process of data-driven improvement of PersuasionAPI. This chapter introduces the data-driven improvement process, followed in this research and according to the following steps

- Getting and cleaning historical data;
- Mining the data to validate or invalidate the hypothesis being tested;
- Implementation (in the case where the hypothesis is validated);
- Testing against the previous version (both on generated and historical data).

8.1 Getting and cleaning the data

During this step, data need to be retrieved from data storage, formatted, and cleaned. These data should be a static snapshot of the historical data available so that the changes that happen to the live data storage do not affect the results. The results of this step are cleaned and pre-formatted snapshots of data ready for mining.

For both of the hypotheses, the same basic snapshot of data was used, which consists of the historical events that occurred within 20 experiments, one event

per line. For the second hypothesis, a more complex data frame was built, that contained the average success probabilities of each strategy for each user, one user per line. The set of most up-to-date snapshots is available for future hypotheses validation and is one of the results of this thesis.

The basic snapshot of data has already been retrieved from the production database such that all the new data frames can be built on top of it if and when necessary without interacting with the production environment. More than that, a more complex data frame used in the second hypothesis is available together with the transformation code. However, it is possible that one will need a different data frame to check other hypotheses, in which case it is highly suggested to use the existing base data snapshot and to reuse the existing code to minimize the time needed for getting the data ready.

8.2 Data mining

This is the core step in hypothesis validation and is an iterative process. After each iteration, results have to be carefully analyzed and, quite frequently, new methods should be applied during the next iteration to either support or invalidate the results of the previous one. This process can be significantly different for various hypotheses.

Data mining was performed for the two hypotheses within this thesis. For the first hypothesis, data mining was performed on both general and strategy levels. In each case, the first two interactions were taken into consideration and the average response rates for the first and second interactions were then compared. For the second hypothesis, correlation was computed based on Pearson's coefficients both with the incomplete pairs data and without it (using pairwise exclusion). Another method, namely contingency tables, was described in detail and suggested for such classes of hypotheses.

The data mining step was completed for the two hypotheses and, since this step is highly individual for every hypothesis, it is unlikely to have a high code reuse for this step in future. However, Spark, together with a functional programming, simplified and significantly sped up data mining in the case of the second hypothesis and is suggested for cases where heavy computations with a high parallelism coefficient are performed. Java and Python code for both hypotheses is available on demand and represents the one of the results of this thesis.

8.3 Prototyping and implementation

There is an environment that consists of the current version of the PersuasionAPI algorithm and test data generation scripts and metrics measurement

scripts ready for tests. The improved version of the algorithm is implemented, thereby embedding the knowledge obtained from the validation of the hypothesis, and is tested against the current version and both Individual Mean and Grand Mean methods.

The current version of the PersuasionAPI algorithm was implemented and tested in a prototyping environment (see Appendix B). Together with the algorithm, test data generation methods were implemented in R environment to perform testing. All together, these environments serve for (1) prototyping the improved version of the algorithm, and (2) testing it against the current version.

8.4 Testing

Testing the new version of the algorithm against the current one is the way to check if there is any improvement and whether or not it is significant enough to implement the new version in production. This comparison includes two runs, in both of which algorithm's performance is monitored for both versions. The first run is based on the generated data in which the hypothesis is embedded. This is a rather straight forward approach that helps to test whether or not the algorithm is implemented correctly and is able to capture patterns in data that form a hypothesis. However, the second run is required to check the improvement on real historical data instead of artificially generated data.

There are several test data generation methods: the current version of PersuasionAPI algorithm, the snapshots of historical data, the performance metric and the new improved version of the algorithm in place, which together form a prototyping and testing environment.

Note that this process is not novel in the research field. It is common to design an offline simulation framework and then conducted evaluations with real online event log data [6], which is very similar to the process described above. However, this is a valuable outcome from the business perspective as it delivers a ready and tested process of the data-driven improvement of PersuasionAPI that is applicable for further improvements and is accompanied by the prototyping infrastructure.

Chapter 9

Conclusion

This chapter presents the results of the thesis, together with the main challenges met along the way, and describes the main contributions from both the research and the business perspective. It concludes with a discussion of the applicability of this research in a broader sense.

9.1 Results

This section describes the results of the research, focusing on the objectives that were set up in Section 2.4.

Objective 1

After several state-of-the-art directions of the improvement of CMAB methods were examined and critically assessed in Chapter 4, building a reinforcement learning model was chosen as the most perspective direction as a more attractive both from the research and the business perspective and due to its fit into the requirements (see section 4.2.5). For this improvement direction, two hypotheses were formed based on psychological and marketing sources of knowledge and expert opinions.

Objective 2

The two hypotheses chosen for validation were the effect of repetition on user's response rate and the correlation between response rate for different pairs of strategies. They were tested, and the first hypothesis was invalidated while the second one had a strong empirical support through all the performed tests.

Objective 3

After the hypotheses were tested, the improved version of PersuasionAPI algorithm was designed, implemented and tested.

Firstly, the current version of PersuasionAPI algorithm was implemented in a prototyping environment. Secondly, the proper heuristic that embedded the knowledge gained from the second hypothesis validation was built. Thirdly, the new improved version of the algorithm was implemented together with two other orthodox methods (for comparison purposes). Lastly, the key testing metric was defined and the new version of the algorithm was tested against the current version and showed performance improvements.

Objective 4

The data-driven improvement process was designed and tested specifically for the PersuasionAPI case. The prototyping/testing environment was developed and tested and includes the following parts:

- Historical data snapshots together with the data retrieval scripts in Java;
- Several scripts in Spark/Python enabling fast and efficient data mining;
- Prototype of the current PersuasionAPI algorithm that allows rapid development and improvement (as opposed to the production code);
- Prototype of the new improved version of the PersuasionAPI algorithm;
- Testing environment including the performance metric, the two other methods to compare with and the testing metric monitoring functionality.

Challenges

There were several challenges along the way worth mentioning as they can give a more complete picture of the research process described in this thesis.

One of the characteristics of this research was the large amount of historical data available, which at some point forced both the methods and the tools of the research to be changed.

Regarding the methods, one of the more complex steps was the interpretation of the hypotheses testing results. For the first hypothesis, a general significance test was applied to understand whether or not the difference between the increases of the response rate was significant. Although this test showed that the difference was significant, it was clear that from the business perspective that such an incremental change is not worth improving the algorithm. The answer given in [46] explains that in the case of a large number of observations (nowadays referred to as "big data"), traditional methods like significance testing do not work, as they perceive virtually any variables that are not equal as significantly different. When the same method was applied at the individual strategy level, it showed that the decrease is also not stable, making it very hard to draw patterns from the historical data.

Traditional tools, such as R language, in some cases were not able to handle the amount of data and computations. For instance, data mining for the second hypothesis was extremely slow when implemented in R and the technology stack was changed to Spark and Python in its functional programming variant.

When it came to implementing the core PersuasionAPI algorithm, its production implementation lacked comments and structure and was far from self-explanatory. It was risky to try to port of the algorithm into a prototyping environment without a proper understanding of what was going on. At some point, it was decided to understand all the design choices of the PersuasionAPI algorithm and then implement the algorithm from scratch.

9.2 Contributions

9.2.1 Research side

The main high-level research result is the fact that in the case of PersuasionAPI there are some static factors that affect users response rates and embedding some of them into the algorithm (more formally – building a reinforcement learning model) is beneficial and rises the quality of PersuasionAPI profiling, meaning the increased response rates.

This result is backed up by the strong empirical evidence for the second hypothesis and by the initial performance benchmark of the new version of the algorithm that clearly shows the potential of the new improved version of the algorithm. There are certain static patterns in user behavior, one of which is that their responses to different persuasive strategies are not completely independent, which means that embedding these patterns and therefore building a reinforcement learning model instead of starting every experiment from scratch is the primary suggested direction for the improvement of the algorithm. For instance, in this research the correlation coefficients between the success probabilities of each pair of strategies were computed and later embedded into a static reinforcement learning model of the new version of the PersuasionAPI.

9.2.2 Business side

There are two results of the thesis that are the most valuable for the business side:

1. The prototype of the new version of the algorithm with a better performance, which is a very practical and applicable result.

2. The data-driven improvement process together with the prototyping environment that were designed and built during this thesis. This two elements are not novel in terms of the research world, but for the company

they structure and significantly simplify further activities on improving the PersuasionAPI algorithm.

9.3 Discussion

9.3.1 Webpower company

From company perspective, this thesis in a broader sense is applicable to the whole process of the PersuasionAPI core CMAB algorithm improvement. It contains a precise description of practical steps that need to be taken in order to further improve the PesuasionAPI algorithm and the prototyping environment that introduces a quick and cheap way to test and select the most promising hypotheses for the PersuasionAPI core algorithm improvement. Note that the prototyping environment and the process description is already available to the data team.

9.3.2 Research body

The outcomes of the thesis have a broader area of application than just a particular PersuasionAPI case. This section discusses the applicability of this thesis to the CMAB research body.

The novel part of this thesis is the improvement of a CMAB algorithm by building a reinforcement learning model under two constraints that are underpinned by the requirements, namely persuasion on an individual level and fully online processing. In this thesis a CMAB algorithm was improved by embedding a reinforcement learning model that contains the correlation data yet maintaining a constant ***O(1)*** time complexity for CMAB *getAdvice* and *learnEvent* methods. The state-of-the-art methods deliver better estimations and have theoretically proven regret bounds, but this quality comes at a price of increased computational complexity. The state-of-the-art methods [53, 54, 55] that guarantee optimal regret bounds have polynomial and even exponential complexity and often introduce complex distributions, sampling from which is very computationally demanding. Other methods that group users into clusters have at least linear complexity ***O(N)*** [5, 6] and do not facilitate persuasion on an individual level.

The heuristic introduced in Section 7.3 uses the information about the correlations between different arms of a CMAB together with the Spark code that efficiently computes the correlation coefficients can be reused in more or less any CMAB problem setting as long as (1) the number of arms is relatively small and (2) there is at least some correlation between pairs of arms. This setting is most commonly seen in but not limited to persuasion profiling.

Another high-level outcome that contributes to the research body of social science is the empirical support of the hypothesis about the need for cognition applied to the persuasion techniques [47, 48], which in short states that people with low need for cognition will be more vulnerable to any persuasive message (see Section 6.1). This research shows strong empirical support for this hypothesis and describes a promising way of improving virtually any persuasion profiling approach by considering this fact.

Bibliography

[1] Susan Moskwa. *Website testing & Google search*. 2012. URL: http://googlewebmastercentral.blogspot.in/2012/08/website-testing-google-search.html (visited on 06/15/2015).

[2] Erin Weigel. *A/B Testing - Concept != Execution*. 2015. URL: http://blog.booking.com/concept-dne-execution.html (visited on 06/15/2015).

[3] Mark J. van der Laan and Sherri Rose. *Targeted Learning: Causal Inference for Observational and Experimental Data*. Springer, 2011.

[4] Thomas C Chalmers et al. "A method for assessing the quality of a randomized control trial". In: *Controlled clinical trials* 2.1 (1981), pp. 31–49.

[5] Linqi Song, Cem Tekin, and Mihaela van der Schaar. "Clustering based online learning in recommender systems: a bandit approach". In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE. 2014, pp. 4528–4532.

[6] Djallel Bouneffouf, Amel Bouzeghoub, and Alda Lopes Gançarski. "A contextual-bandit algorithm for mobile context-aware recommender system". In: *Neural Information Processing*. Springer. 2012, pp. 324–331.

[7] Djallel Bouneffouf. "The Impact of Situation Clustering in Contextual-Bandit Algorithm for Context-Aware Recommender Systems". In: *arXiv preprint arXiv:1304.3845* (2013).

[8] Sulekha Goyat. "The basis of market segmentation: a critical review of literature". In: *European Journal of Business and Management* 3.9 (2011).

[9] Greg Linden, Brent Smith, and Jeremy York. "Amazon.com recommendations: Item-to-item collaborative filtering". In: *Internet Computing, IEEE* 7.1 (2003), pp. 76–80.

[10] Webpower. *Creating Persuasive Content*. 2014. URL: https://docs.webpower.io/guides/creating-persuasive-content.html (visited on 06/15/2015).

[11] Volodymyr Kuleshov and Doina Precup. "Algorithms for multi-armed bandit problems". In: *arXiv preprint arXiv:1402.6028* (2014).

[12] Joannes Vermorel and Mehryar Mohri. "Multi-armed bandit algorithms and empirical evaluation". In: *Machine Learning: ECML 2005*. Springer, 2005, pp. 437–448.

[13] Steven L Scott. "A modern Bayesian look at the multi-armed bandit". In: *Applied Stochastic Models in Business and Industry* 26.6 (2010), pp. 639–658.

[14] Webpower. *PersuasionAPI REST API*. 2014. URL: `https://docs.webpower.io/references/rest-api.html` (visited on 06/15/2015).

[15] Maurits Kaptein. "Web Persuader: Customized Persuasive Messages on the Web." In: Pending review (2013).

[16] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*. Springer, 1985.

[17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* (1996), pp. 237–285.

[19] Dengpan Liu, Sumit Sarkar, and Chelliah Sriskandarajah. "Resource allocation policies for personalization in content delivery sites". In: *Information Systems Research* 21.2 (2010), pp. 227–248.

[20] Dongsong Zhang. "Delivery of personalized and adaptive content to mobile devices: a framework and enabling technology". In: *Communications of the Association for Information Systems* 12.1 (2003), p. 13.

[21] Leiguang Gong. "Can web-based recommendation systems afford deep models: a context-based approach for efficient model-based reasoning". In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM. 2004, pp. 89–93.

[22] Sandeep Pandey, Deepayan Chakrabarti, and Deepak Agarwal. "Multi-armed Bandit Problems with Dependent Arms". In: ICML '07 (2007), pp. 721–728.

[23] Daniela Pucci de Farias and Nimrod Megiddo. *Exploration-Exploitation Tradeoffs for Experts Algorithms in Reactive Environments*. URL: `http://papers.nips.cc/paper/2734-exploration-exploitation-tradeoffs-for-experts-algorithms-in-reactive-environments.pdf`.

[24] John R Hauser et al. "Website morphing". In: *Marketing Science* 28.2 (2009), pp. 202–223.

[25] Charles Stein. "Inadmissibility of the Usual Estimator for the Mean of a Multivariate Normal Distribution". In: (1956), pp. 197–206.

[26] Bradley Efron and Carl N Morris. *Stein's paradox in statistics*. WH Freeman, 1977.

[27] Olivier Chapelle and Lihong Li. "An Empirical Evaluation of Thompson Sampling". In: (2011). Ed. by J. Shawe-Taylor et al., pp. 2249–2257. URL: http://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling.pdf.

[28] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

[29] John Langford and Tong Zhang. "The epoch-greedy algorithm for multi-armed bandits with side information". In: *Advances in neural information processing systems*. 2008, pp. 817–824.

[30] Deepak Agarwal, Bee-Chung Chen, and Pradheep Ela. "Explore/exploit schemes for web content optimization". In: *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE. 2009, pp. 1–10.

[31] John C Gittins. "Bandit processes and dynamic allocation indices". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1979), pp. 148–177.

[32] Glen L Urban et al. "Morph the Web to build empathy, trust and sales". In: *MIT Sloan Management Review* 50.4 (2009), pp. 53–61.

[33] Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. "Thompson sampling: An asymptotically optimal finite-time analysis". In: *Algorithmic Learning Theory*. Springer. 2012, pp. 199–213.

[34] Shipra Agrawal and Navin Goyal. "Analysis of Thompson sampling for the multi-armed bandit problem". In: *arXiv preprint arXiv:1111.1797* (2011).

[35] Aleksandrs Slivkins. "Contextual bandits with similarity information". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 2533–2568.

[36] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. "Multi-armed bandits in metric spaces". In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 2008, pp. 681–690.

[37] Aleksandrs Slivkins. "Multi-armed bandits on implicit metric spaces". In: *Advances in Neural Information Processing Systems*. 2011, pp. 1602–1610.

[38] Sébastien Bubeck et al. "Online optimization in X-armed bandits". In: *Advances in Neural Information Processing Systems*. 2009, pp. 201–208.

[39] Robert D Kleinberg. "Nearly tight bounds for the continuum-armed bandit problem". In: *Advances in Neural Information Processing Systems*. 2004, pp. 697–704.

[40] Gamal Hussein. "Enhanced K-means-Based Mobile Recommender System". In: *International journal of information studies* 2.2 (2010).

[41] Deepayan Chakrabarti et al. "Mortal multi-armed bandits". In: *Advances in Neural Information Processing Systems*. 2009, pp. 273–280.

[42] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. "Ranked bandits in metric spaces: learning diverse rankings over large document collections". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 399–436.

[43] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1 (1999), pp. 181–211.

[44] J Poppenk, S Köhler, and M Moscovitch. "Revisiting the novelty effect: when familiarity, not novelty, enhances memory." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 36.5 (2010), p. 1321.

[45] Endel Tulving and Neal Kroll. "Novelty assessment in the brain and long-term memory encoding". In: *Psychonomic Bulletin & Review* 2.3 (1995), pp. 387–390.

[46] Clive William John Granger. "Extracting information from mega-panels and high-frequency data". In: *Statistica Neerlandica* 52.3 (1998), pp. 258–272.

[47] John T Cacioppo and Richard E Petty. "The need for cognition." In: *Journal of personality and social psychology* 42.1 (1982), p. 116.

[48] Curtis P Haugtvedt, Richard E Petty, and John T Cacioppo. "Need for cognition and advertising: Understanding the role of personality variables in consumer behavior". In: *Journal of Consumer Psychology* 1.3 (1992), pp. 239–260.

[49] John W Graham. "Missing data analysis: Making it work in the real world". In: *Annual review of psychology* 60 (2009), pp. 549–576.

[50] Maurits Kaptein and Dean Eckles. "Heterogeneity in the effects of online persuasion". In: *Journal of Interactive Marketing* 26.3 (2012), pp. 176–188.

[51] David Draper. *Bayesian hierarchical modeling*. 2000.

[52] G David Garson. "Fundamentals of hierarchical linear and multilevel modeling". In: *Hierarchical linear modeling: guide and applications. Sage Publications Inc* (2013), pp. 3–25.

[53] Varsha Dani, Sham M Kakade, and Thomas P Hayes. "The price of bandit information for online optimization". In: *Advances in Neural Information Processing Systems*. 2007, pp. 345–352.

[54] Jacob Abernethy, Elad Hazan, and Alexander Rakhlin. "Competing in the Dark: An Efficient Algorithm for Bandit Linear Optimization." In: *COLT*. 2008, pp. 263–274.

[55] Lihong Li et al. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web*. ACM. 2010, pp. 661–670.

A Detailed statistics of the experiments included into the historical data set

Interactions	Users	Interactions per user
21766199	741876	29.34
788834	261188	3.02
6749	358	18.85
5184774	711098	7.29
71588	51562	1.39
40437	33566	1.20
14828355	408974	36.26
344208	127989	2.69
134883	71889	1.88
1120619	249346	4.49
16787	2775	6.05
86430	57932	1.49
686485	251718	2.73
44776	17189	2.60
167491	44740	3.74
94826	74648	1.27
13732138	603027	22.77
13274340	2811189	4.72
31795	19114	1.66
9496780	173677	54.68

Table 1: Total numbers of users and interactions for 20 clients included into the historical data set

B PersuasionAPI core algorithm prototype implementation

Listing 1: PersuasionAPI core algorithm implementation in R

```
package papi;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import org.apache.commons.math3.distribution.BetaDistribution;

public class Papi {

  /**
   * Number of strategies used in the dataset
   */
  private int strategiesNumber;

  /**
   * An interface that represents the estimation method used
   */
  private Estimation estimationMethod;

  public Papi(Estimation method, int strategiesNumber) {
    this.estimationMethod = method;
    this.strategiesNumber = strategiesNumber;
  }

  /**
   * @author vladimir Stores an interaction event, one per line in a
       dataset
   */
  public static class Event {

    /**
     * User's identifier
     */
    private String id;

    /**
     * Average success probabilities for each strategy (not known),
         counted previously based on the
```

```
43       * historical "random" data
44       */
45      private double[] probabilities;
46
47      /**
48       * Draws from the Bernoulli distributions of probabilities (embedded
             into the dataset to make
49       * sure
50       */
51      private boolean[] outcomes;
52
53      /**
54       * Index of the optimal strategy (not known)
55       */
56      private int indexBest;
57
58      /**
59       * Number of strategies used in the particular event (note that this
             number should be always
60       * equal to the number of strategies of the whole dataset, this is
             taken care of during the
61       * dataset generation)
62       */
63      private static int strategiesNumber = 0;
64
65      public Event(String rawStr) {
66
67        // Parsing raw String into an object
68        String[] elements = rawStr.split(",");
69        strategiesNumber = (elements.length - 2) / 2;
70
71        probabilities = new double[strategiesNumber];
72        outcomes = new boolean[strategiesNumber];
73
74        id = elements[0];
75
76        for (int i = 0; i < strategiesNumber; i++) {
77          probabilities[i] = Double.valueOf(elements[i + 1]);
78          outcomes[i] = elements[strategiesNumber + i + 1].equals("0") ?
                false : true;
79        }
80
81        indexBest = Integer.valueOf(elements[strategiesNumber * 2 + 1]);
82      }
83    }
84
85    public List<Integer> sequentialProcessing(List<Event> events) throws
          IOException {
86      strategiesNumber = Event.strategiesNumber == 0 ? 4 :
            Event.strategiesNumber;
```

```
87      double[] P = new double[strategiesNumber];
88      int[] N = new int[strategiesNumber];
89
90      for (int i = 0; i < strategiesNumber; i++) {
91        P[i] = 0.5;
92        N[i] = 2;
93      }
94
95      // A list that stores cumulative regret for each point in time
            (discrete)
96      List<Integer> cumulativeRegret = new LinkedList<Integer>();
97
98      Map<String, Double[]> p = new HashMap<String, Double[]>();
99      Map<String, Integer[]> n = new HashMap<String, Integer[]>();
100
101     for (Event e : events) {
102
103       // Initialization of individual level N and P in case no
              observations were processed so far
104       // for this user id
105       if (!p.containsKey(e.id)) {
106         Integer[] initialN = new Integer[strategiesNumber];
107         Double[] initialP = new Double[strategiesNumber];
108         for (int i = 0; i < strategiesNumber; i++) {
109           initialP[i] = P[i];
110           initialN[i] = 2;
111         }
112
113         p.put(e.id, initialP);
114         n.put(e.id, initialN);
115       }
116
117       Double[] currentP = p.get(e.id);
118       Integer[] currentN = n.get(e.id);
119
120       int winningIndex = -1;
121       double winningValue = -1;
122
123       // Thompson sampling
124       for (int i = 0; i < strategiesNumber; i++) {
125
126         // Getting the estimation of P (delegated to the Estimation
                interface implementation)
127         double estimatedP = estimationMethod.getEstimation(currentP,
                currentN, P, N, i);
128
129         double alpha = estimatedP * currentN[i];
130         double beta = (1 - estimatedP) * currentN[i];
131
132         BetaDistribution distr = new BetaDistribution(alpha, beta);
```

```
133         double sample = distr.sample();
134
135         if (sample > winningValue) {
136           winningIndex = i;
137           winningValue = sample;
138         }
139       }
140
141       int outcome = e.outcomes[winningIndex] ? 1 : 0;
142       int idealOutcome = e.outcomes[e.indexBest] ? 1 : 0;
143
144       // Updating the number of observations (both overall and
              individual)
145       currentN[winningIndex]++;
146       N[winningIndex]++;
147
148       // Updating the probability (both overall and individual)
149       currentP[winningIndex] += (outcome - currentP[winningIndex]) /
              currentN[winningIndex];
150       P[winningIndex] += (outcome - P[winningIndex]) / N[winningIndex];
151
152       // Computing the cumulative regret metrics
153       int base =
154           cumulativeRegret.size() > 0 ?
                cumulativeRegret.get(cumulativeRegret.size() - 1) : 0;
155       cumulativeRegret.add(base + idealOutcome - outcome);
156     }
157
158     FileWriter writer = new FileWriter("cumRegretPapi.csv");
159
160     for (int X : cumulativeRegret) {
161       writer.append(X + "\n");
162     }
163
164     writer.flush();
165     writer.close();
166
167     return cumulativeRegret;
168   }
169
170   // An example on how sequential processing should be initiated
171   public static void main(String[] args) throws IOException {
172     List<Event> events = new LinkedList<Event>();
173
174     String csvFile = "{INPUT_PATH}/algo-dataset.txt";
175     BufferedReader br = null;
176     String line = "";
177
178     br = new BufferedReader(new FileReader(csvFile));
179     while ((line = br.readLine()) != null) {
```

```
180       events.add(new Event(line));
181     }
182     br.close();
183 
184     Papi papi = new Papi(new BasicEstimation(), 4);
185 
186     papi.sequentialProcessing(events);
187   }
188 }
```

C Correlation check implementation

Listing 2: Correlation check implemented in Spark and Python

```
from pyspark import SparkContext
import math
from pyspark.mllib.stat import Statistics

logFile = "{PATH_TO_LOGS}/randomUsers.csv"
sc = SparkContext("local[*]", "Correlation")
logData = sc.textFile(logFile).repartition(8)

print(logData.count())

# simple and straight-forward conversion
def func(a):
    if(a == "true"):
        return(1.0)
    else:
        return(0.0)

# A function to fulfill the probabilities that were not counted
def simplify(y):
    probabilities = 5*[-1.0]
    for i in range(0, 4):
        found = False
        for entry in y:
            if int(entry[0]) == i:
                probabilities[i] = float(entry[1])
                found = True

    return probabilities

def getFakeKey(v, var1, var2):
    if cmp(v[var1], -1.0) == 0 or cmp(v[var2], -1.0) == 0:
        return 0
    else:
        return 1


def countCorrelationScore(data, var1, var2):
    # counting the number of records - straight forward transformation
        into (0, 1) key-value pairs,
    # then reducing by 0 key
    data = data.filter(lambda (k, v): getFakeKey(v, var1, var2) == 1)

    n = data.map(lambda x: (0, 1)).reduceByKey(lambda a, b: a +
        b).first()[1]
```

```
    sum1 = data.map(lambda (k, v): (1, v[var1])).reduceByKey(lambda a,
        b: a + b).first()[1]
    sum2 = data.map(lambda (k, v): (1, v[var2])).reduceByKey(lambda a,
        b: a + b).first()[1]
    pow1 = data.map(lambda (k, v): (1, math.pow(v[var1],
        2))).reduceByKey(lambda a, b: a + b).first()[1]
    pow2 = data.map(lambda (k, v): (1, math.pow(v[var2],
        2))).reduceByKey(lambda a, b: a + b).first()[1]

    multi = data.map(lambda (k, v): (1, v[var1] *
        v[var2])).reduceByKey(lambda a, b: a + b).first()[1]

    res = (n*multi - sum1*sum2)/(math.sqrt(n*pow1 - math.pow(sum1,
        2))*math.sqrt(n*pow2 - math.pow(sum2, 2)))
    return res

base = logData.map(lambda x : x.split(',')).map(lambda x : (x[0] + ',' +
    x[2], x[1])).cache()

# number of successes per user per strategy
part1 = base.map(lambda (k, a): (k, func(a))).reduceByKey(lambda a, b: a
    + b).cache()

# number of events stored per user per strategy
part2 = base.map(lambda (k, a): (k, 1)).reduceByKey(lambda a, b: a +
    b).cache()

part3 = part1.join(part2).map(lambda (k, (a,b)): (k, a/b)).map(lambda
    (k, a) : (k.split(",")[0], (k.split(",")[1], a)))\
    .groupByKey().map(lambda x : (x[0], sorted(list(x[1])))).map(lambda
        (x, y): (x, simplify(y))).cache()
part3.repartition(1).saveAsTextFile("{OUTPUT_PATH}/output.txt")
part3 = part3.filter(lambda (k, v): getFakeKey(v, i, j) == 1)

f = open('correlation', 'w')

for i in range(0, 4):
    for j in range(0, 4):
        #MLib implementation
        f.write(str(Statistics.corr(part3.map(lambda (k, v) : v[i]),
            part3.map(lambda (k, v) : v[j]), method="pearson")))

        #Custom-made method
        #f.write(str(countCorrelationScore(part3, i, j)) + " ")
    f.write("\n")
f.close()
```