CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Comparison of Arm Selection Policies for the Multi-Armed Bandit Problem

Master's thesis in Computer Science: Algorithms, Languages and Logic

FIFI JOHANSSON

MIRIAM MCHOME

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Comparison of Arm Selection Policies for the Multi-Armed Bandit Problem

Fifi Johansson and Miriam Mchome

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Comparison of Arm Selection Policies for the Multi-Armed Bandit Problem
Fifi Johansson and Miriam Mchome

Supervisor: Christos Dimitrakakis, Computer Science Department
Advisor: Magnus Petersson, Company Advisor
Examiner: Carl Seger, Computer Science Department

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Comparison of Arm Selection Policies for the Multi-Armed Bandit Problem
Fifi Johanssona and Miriam Mchome
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Web content optimization involves deciding what content to put on a web page, its layout and design. All of which involve selecting few options among many. With the advent of personalization, many companies seek to make this decision even on a per-user basis in order to improve customer experience and satisfaction. Contextual multi-armed bandit provides several strategies to address this online decision-making problem at a lower experimental cost than traditional A/B testing.

In this study, we compare three common Contextual Bandit strategies that exist in literature namely E-greedy, LinUCB and Thompson Sampling, and apply two of them, E-greedy and LinUCB, to three datasets. In doing so we offer further empirical evidence on the performance of these strategies and insights for practitioners on what strategy might work for them.

Our results suggest that both approaches, E-Greedy and LinUCB are effective in improving click-through rate compared to the random approach. The more sophisticated approach has better results with large datasets, and a quite unstable performance when the number of datapoints is small. On the other hand, we find that the more sophisticated approach is more sensitive to parameter tuning and can have significantly worse outcome when parameters are incorrect. Our study also finds that LinUCB can have higher data requirements when performing evaluation offline. Collectively the varying performance of these approaches across dataset signal the need for better tools and procedures to help practitioners decide on the appropriate approach.

Keywords: machine learning, multi-armed bandit, offline evaluation, contextual bandit.

Acknowledgements

Due to confidentiality reasons, the company name in this report will be substituted by "company A".
We would like to thank our company supervisor Magnus Petersson for his support and valuable inputs during our time working at company A. Another big thank you goes to our school supervisor Christos Dimitrakakis at Chalmers, who guided us through the thesis and provided remote support even through we are in two different cities. We also appreciate the patience from our examiner Carl Seger to read our reports as well as other people who are involved in the thesis.

Fifi Johansson and Miriam Mchome , Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

This chapter introduces a common online content optimization problem and shows how the multi-armed bandit problem can be used to model it. The chapter first presents some motivating background knowledge along with a few real world examples. It continues by discussing the critical challenges in Section 1.2, followed by the goal of the thesis in Section 1.3. The goal is presented from the practitioners' perspective using concrete examples and finally the chapter outlines the scope of the project in Section 1.4.

1.1 Background

Deciding on an optimal selection of content to show is a common optimization problem faced by companies that serve online content. An online news company, for example, may have forty news articles they could show at any given time but only a handful of slots available. Deciding which articles to show, as different users visit their site, is an important decision because it not only affects the user's experience but it also affects how quickly the company receives feedback on the articles. The crucial piece of the problem is that even after carefully screening and analysing new articles, the only way to know which articles are preferred is to try them on users. But since the number of available slots is limited, it is desirable to be very strategic about which articles to try and how often to try them.
This content optimization problem is applicable to many areas including advertising, recommender systems, e-commerce systems and others. The optimization parameters can also extend to how the content is ordered because different layout options may affect a user's experience on a site. The key unifying feature is having partial information because feedback is only received for what is shown. Consequently, this online content optimization problem belongs to a known set of problems called online partial information decision-making problems. Problems in this set require balancing exploitation to show the best content and exploration to gain insights into content performance. The multi-armed bandit (MAB) problem offers a good way to model and reason about this tradeoff.
Formally when modelling this problem a desirable user's response to what is being shown is called a reward. Rewards can be used as a performance measure when trying to gauge how well an optimization technique is doing. For example, we can compare technique A and B by looking at their total rewards after a certain time frame. In this case, the problem is set up to maximize reward. Similarly, if an optimal solution is known, the performance can be measured by comparing the difference

in rewards between solution A and a known optimal solution B. The difference here is called regret and the problem is set up to minimize regret. Another way to think of regret is that it measures the costs associated with either exploring content that turns out to be low performing, or not exploring enough and consequently missing out on content that would have been high performing.
In the multi-armed bandit problem [23] there is one player. The player is faced with many 'bandit' arms and has to decide which arm to pull in order to receive a reward.The goal of the player is to maximize reward. In the beginning, the player has no knowledge of what reward to expect from the arms. They can only observe rewards once they select an arm to pull. In the online content case, the arms represent the views. Selecting an arm represents selecting which content(s) to show to a user. A user's response to the content can be used to model the reward. The problem can also model incorporation of contextual data about the contents, users or the environment. This flavour of the problem is called contextual multi-armed bandit.
Since this problem is interactive in nature, evaluating solutions has its own challenges. Companies that wish to use multi-armed bandit algorithms need a way to evaluate and test their solutions before they can be exposed to real users. This evaluation is often performed on logged data, and when done as such it is called off-policy evaluation. Algorithms used to decide what the user sees are called policies. Unfortunately, logged data is often based on a different policy from that under evaluation resulting in a difference between content recommendation from new policy versus logged policy. These differences increase as the number of possible arms increase. In such cases, companies need to employ techniques in inferential statistics to make a correct estimate of the effect of a new policy.
Off-policy evaluation has several uses including, fine-tuning of algorithm parameters, estimation of algorithm effectiveness and best arm selection policy identification. Even though the contextual multi-armed bandit problem is well studied, its implementation still has several challenges. One of the most recent studies focusing on these challenges outlines three main areas that need to be addressed namely: how to best encode feedback, how to best infer knowledge in an off-policy setting, and what is the best policy for exploration [4]. In addition to utilizing some of the recommendations suggested in this study, our study provides a tool to help practitioners answer these questions for their own data sets. It also focuses specifically on the last question (What is the best policy for exploration) to offer further empirical insight into the trade-offs between different policies.

1.2 Problem

The general problem is finding an optimal selection and layout of content on a website.The assumption is that the website has K number of top slots to place the content and M number of options to choose from. The study is conducted at a digital rights management company A which serves online content. The company A has multiple product areas where online content needs to be served to multiple users. The study intends to provide a way to evaluate different policies in order to inform how to get optimal content and layout for those users. Since the content

being served is large and can change frequently, the contextual MAB arms used can be very large. The intention is to devise a solution that can work well under these assumptions. The study also aims to find a solution that can generalize well to different products.
To evaluate the performance of a solution, one anonymized logged data set from the company A was used and an off-policy evaluation was performed. An artificially constructed data set and a publicly available referenced dataset were investigated to further supplement the analysis. The logged data set in question was from a webpage. The page had positions that could be filled with content a user might be interested in. The logged data consisted of user interaction events with the page, the options the user was presented with when the events occurred and their response to the recommended options. There was a fixed number of slots for the options and a large choice of values that these slots could occupy. These positions could be treated as an ordered list with the first position being a higher priority than others. Optimization implies selecting options that the user visiting the page would want to use. The data set had a fixed number of contextual information pertaining to the user and the page they were interacting with.
Available options of content were modelled as the arms. A users response to what is being shown was used to model reward. Given a new observation of context and user, each policy produced a ranking of the arms. Depending on the observed response from the user, each policy updated model parameters used in its ranking. Each example had its own set of top K rankings it was interested in. Offline evaluation was used to provide an estimate of expected reward from each policy and therefore a way to decide which policy would result in a higher expected reward.

1.3 Goal

The primary goal of the thesis is to investigate an optimal way of applying contextual multi-armed bandit algorithms to optimize some targeted content on a web page. It seeks to make it easy for practitioners to evaluate existing policy options and choose a strategy that best fits their need. For someone who wants to apply a bandit algorithm, the study aims to provide them with a list of the main factors they need to evaluate/consider and a library that they can use to make these evaluations. The study seeks to help practitioners answer questions like:

1. How much logged-data do they need to make useful estimates of effectiveness
2. What should the logged data comprise of
3. What policy is the most promising for a specific application
4. How confident can they be with their estimate

Since the list of possible algorithms/policies is large, only three common variations are compared. The study produces a python library which consists of three policy implementations for contextual multi-armed bandit and an offline evaluation script. These policies are then used to produce input for the evaluation script. The evaluation script and policies are separated to allow the possibility of using custom policies in the future. The evaluation script is used to compute an estimate of algorithm performance on logged data. The performance of these three algorithms is compared on a selection of datasets. Each dataset is divided into a training set and validation

set. The training set is used to finetune parameters and the validation set is used to assess algorithm performance. Finally, the study compares results for each of the three datasets, evaluates the tool and suggests parameters that should be explored when performing an analysis.

1.4 Limitations

The thesis evaluates two different arm selection policies whose efficiencies have been demonstrated in various fields. These are E-Greedy and contextual UCB. Kuleshov has shown that e-greedy in practice outperforms other algorithms in most settings [12]. Contextual UCB [13] has been applied to web page optimization problems and showed uplifts in conversion rate/click-through rate. The intention of the thesis is to analyse efficiencies of these arm selection policies on selected datasets and compare their pros and cons.
In order to compare the performances of the policies, the thesis also aims to provide a way of evaluating policies offline. Since the focus of the thesis is on different arm selection policies, only two policy evaluation methods are implemented. The direct method (DM) is implemented for evaluating the performance of the models on condition the logged data is uniformly distributed. The inverse propensity score (IPS) is used when the logged data is selected with another policy. The other offline evaluation methods such as doubly robust (DR) are left out for future studies.

2
Theory

The online content optimization problem has been researched extensively in both industry and education. Those studies have proposed several approaches to match different problem settings. In this chapter, these approaches are summarized together with the multi-armed bandit problem. The chapter first motivates where the bandit approach fits in the context of solutions to online content optimization. It then provides a detailed description of the multi-armed bandit problem, followed by three common solutions from literature. To provide further insights on the practical techniques used when implementing solutions, relevant evaluation techniques and evaluation metrics are described in Section 2.3 and 2.4.

2.1 Online content optimization

[13] describes personalization as a process of gathering and storing user attributes, managing and evaluating content based on past and present user behaviour, and, delivering the best content to the current user being served. The process of personalization is sought after as a method of improving user satisfaction. The goal is to tailor content to suit individual user needs [5]. To do so, users and content are modelled into features and various algorithms are used to match the users with their desired content. Since the number of users and content can be very large, the challenge is finding manageable commonalities for transferring knowledge, either between user features or content features.
Some of the approaches used in personalization include collaborative filtering, content-based filtering and hybrid approaches. Collaborative filtering [9] works by finding similarities in user's past behaviour. A common descriptive example is the phrase users who bought X also bought Y. [13] argue that this method works well when the content doesn't change often and when there is a lot of similarities in user behaviour. Content-based filtering, on the other hand, focuses on learning content similarities [16]. In doing so, it helps users find new content that is similar to the content they were interested in previously. Hybrid approaches combine these two approaches in some way, for example, offsetting collaborative filtering with content based for the recommendation of new content that other similar users haven't tried yet [13].
Unfortunately, these traditional content optimization techniques don't address two common scenarios namely, having a significant number of new users and having content that changes frequently. When a user is new to a system, their preferences are not known. This situation is called 'cold start' [18]. Being able to match preferences to appropriate content efficiently is crucial when the number of new users

is high. Similarly, if the content changes frequently over time or if its popularity changes frequently over time it becomes critical to have a way to efficiently model its performance before the contents value becomes obsolete. Finding the best way to match users and content requires some exploration phase. Exploration is not only expensive but it also comes at the risk of reducing user satisfaction. Multi-armed bandit algorithms are specifically designed to balance exploration and exploitation and systematically learn from the exploration in an online fashion.

2.2 Multi-armed bandit problem

The concept of multi-armed bandit problem was first introduced in early 1930s [23] and 1950s [20]. The problem describes the decision making challenge in choosing among a set of possible choices with unknown rewards for the purpose of achieving maximum profit in multiple tries. One classic example of a one-armed bandit problem that is similar to the multi-armed bandit problem is when a gambler stands in front of several slot machines and needs to make decisions of which machine to play, how many times each machine should be played, whether to try a new machine or choose the machine which has the most reward so far, etc. In this example, the gambler explores during the time when new machines have been tried and exploits by choosing the best performed machine. The problem of balancing the exploration of new knowledge and the exploitation of existing knowledge is also known as the exploration-exploitation tradeoff in reinforcement learning.
In a multi-armed bandit problem, several arm selection policies have been established to solve the exploration-exploitation tradeoff. Three of these are presented in the following sections, namely E-Greedy (2.2.1), UCB (2.2.2) and Thompson sampling (2.2.3).
There are several variants of the multi-armed bandit problem which are proposed from empirical studies with the aim of modeling different problem settings in real-world situations. Contextual bandit describes the multi-armed bandit problem as its expected payoff of a chosen action is dependent on both the chosen action (arm) and a context vector. The context vector is represented by an n-dimensional feature vector which contains additional information of either the chosen action or the user. The section 2.2.4 presents some contextual bandit studies which have conducted in the recent years.

2.2.1 E-Greedy

Epsilon-Greedy algorithms are the most common approach for the multi-armed bandit problem. Perhaps this is explained by their simplicity and generalizability to different scenarios. The key features of an E-Greedy are outlined below:

1. ϵ percent of the time, select an arm randomly.
2. The rest of the time select the best arm, for example, the arm with the highest mean reward.

Epsilon can be configured to remain constant or decay over time. It can also be configured to work in phases, for example, exploration first followed by pure exploitation phase. If the value of epsilon is configured to remain constant throughout

the algorithm then only a linear bound on regret can be achieved [12]. An empirical evaluation, however, by [25], found no significant advantage of configuring a decaying epsilon over having a constant one.
The biggest criticism of E-Greedy algorithms is the wastage of resources. In most variations exploration is performed at the same rate across all arms. Algorithms that are designed to improve on E-Greedy seek to be smarter about when to explore and when to stop. Despite their clear shortcomings, E-Greedy algorithms have shown surprising success rates in empirical studies when compared to other advanced strategies [25] and [12].

2.2.2 UCB

UCB stands for upper confidence bound. Different versions of the algorithm exist and they are considered to be a more advanced exploration technique than e-greedy. The common strategy for UCB algorithms is selecting an arm with the highest 'confidence bound'. Different versions differ in how they define this confidence bound. A common factor is having a bound that increases over time and whose size fluctuates depending on the performance of the arm. UCB's popularity can be attributed to having a provable logarithmic bound on regret [2]. Consequently, it has many applications including independent arms [6] and continuous arms [22].
Empirical studies, however, report different performance when compared to e-Greedy. [2] for example, perform synthetic experiments which suggest that while e-Greedy policies outperform UCB policies in most experimental cases, UCB'S performance is more consistent across different experimentation parameters. Their experiment suggests that UCB is more stable to high variance in rewards across different arms. Its performance does not change when the number of arms increases. On the other hand, [13] shows higher uplift in click-through rate for news recommendation when using a tweaked version of UCB as compared to Greedy approaches suggesting that UCB is better than greedy.
In most flavours of UCB, the level of exploration also takes into account the confidence in the performance of each arm. The simplest variant is called UCB1 [2]. Here the algorithm ensures all the arms are tried at least once. Afterwards, the algorithm keeps track of the number of times each arm is chosen and it sequentially selects the arm that maximizes the following formula:

$$\mu_{a,t} + \sqrt{\frac{2 \ln t}{n_a}}$$

$\mu_{a,t}$ is the mean reward of the arm at time t, and n_a is the number of times that arm was chosen. Therefore if n_a is small, the algorithm would favour higher exploration of this arm, unless the mean reward from this arm relatively low.

2.2.3 Thompson sampling

Thompson sampling is another common arm selection policy. It was first introduced in 1933 on a two-armed bandit problem analysing treatment efforts in clinical

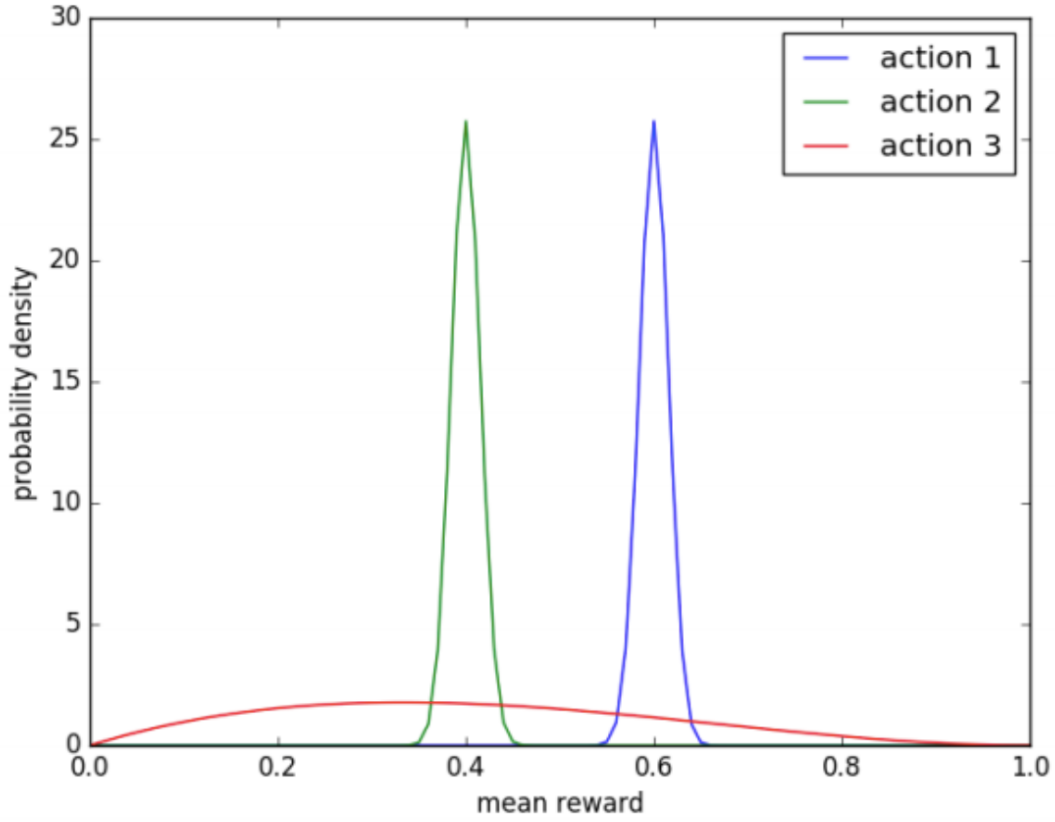
trials [23]. It has recently become more popular in problems such as advertising agarwal2014laser, recommender systems [11] and search [10]. This increase in the algorithm popularity has been attributed to empirical success over UCB demonstrated in studies such as [6] who have shown lower regret on advertising and news recommendation data.

The algorithm itself is best described in contrast to the e-greedy algorithm. Initially, we assume that every arm produces a reward according to some likelihood function P conditioned on an initially unknown parameter θ, context x, and rewards r. While e-greedy selects an arm with the maximum expected reward, Thompson sampling policy randomly samples θ from the posterior distribution and selects the arm with the highest reward given the sampled θ. General steps for this algorithm from [21] are outlined below.

Greedy (X, p,r)	Thompson(X,p,r)
for t = 1, 2, . . . do	for t = 1, 2, . . . do
#estimate model $\boldsymbol{\theta}^* \leftarrow E_p[\boldsymbol{\theta}]$	#sample model Sample $\boldsymbol{\theta}^* \sim p$
#select and apply action $x_t \leftarrow \text{argmax}_{x \in X} E[r(y_t) \mid x_t = x)]$ Apply x_t and observe y_t	#select and apply action $x_t \leftarrow \text{argmax}_{x \in X} E[r(y_t) \mid x_t = x)]$ Apply x_t and observe y_t
#update distribution $p \leftarrow P_p(\boldsymbol{\theta} \mid x_t, y_t)$	#update distribution $p \leftarrow P_p(\boldsymbol{\theta} \mid x_t, y_t)$
end for	end for

For illustration, consider an example from [21]. Figure 1 represents the probability density functions of 3 actions 1, 2 and 3. Assume that Action 1 and 2 have been tried 1000 times each, while action 3 is new and has only been tried 10 times with their success and failure rates as outlined below. If a greedy approach has a fixed exploration rate, it will explore each arm at the same rate. If the exploration rate is small, it will favour action 1 because the average success rate of arm 2 and 3 are lower. On the other hand, with a beta distribution as the posterior, Thompson sampling policy will sample from action 1, 2 and 3 with a probability of 0.82, 0 and 0.18 respectively. These are probabilities that a random estimate from each action is greater than a random estimate from other actions. In this case, the rate of exploration not only proportional to expected performance but it is also targeted toward arms with higher uncertainty and potential to have a higher reward.

Action 1 (succes, failure) = (600, 400)
Action 2 (succes, failure) = (400, 600)
Action 3 (succes, failure) = (4, 6)



Despite its popularity,[21] outline several criteria to be considered when applying this algorithm. Firstly, careful consideration of a prior distribution for the arms as it can negatively affect overall performance. Secondly, some systems have non-stationary and therefore exploration needs to account for the fact that the target is changing over time unlike in the standard Thompson sampling approach. And finally, [21] argue that Thompson sampling algorithms are not good for time-sensitive problems requiring little active exploration or sparse-linear reward models that require complex assessment of rewards with interaction effects between different arms.

2.2.4 Contextual bandit

The contextual bandit setting has been studied extensively in recent years. It is mainly used in the situation where additional information is provided in a multi-armed bandit problem and thus influence which arm to pull. Such situation is very common in online content recommendation problems.
An example can be seen in the paper "Contextual Multi-Armed Bandits" [14], the contextual bandit setting is used to model an ad recommendation problem on a search engine taking user's search query as the context. The solution which is proposed from the paper describes a simple algorithm known as query-ad-clustering. The solution steps can be briefly explained by first clustering the contexts (search query) based on their similarities, and then run a multi-armed bandit algorithm separately on each cluster. However, this solution may not work if the contexts consist of numerical data, which is the case for the public dataset (3.1.2. Public

dataset).

Another example that is similar to the approach taken in this thesis is described in the paper "A Contextual-Bandit Approach to Personalized News Article Recommendation" [13]. Here the contextual bandit setting is addressed as a news recommendation problem where each unique news id represents one arm. Two different types of contexts (user contexts and news contexts) are used and together they compose a context vector x. The paper introduces a new algorithm LinUCB which its expected payoff of each arm is calculated by a linear function of the context vector x and an unknown coefficient vector gamma. LinUCB has proved to be computationally efficient when it applies to sparse data as well as when the number of available arms is large.

Despite the similarities in implementations, the contextual bandit approach in this thesis differs from Li's study [13] in two ways. First, three datasets are tested to validate the usefulness of bandit algorithms, where each of these simulates a different bandit problem setting. Second, an online evaluation is applied to the generated dataset to confirm results from the offline evaluation. The difference between online evaluation and offline evaluation is further discussed in section 2.3.

2.3 Evaluation

In real-world applications, it is often very costly to put online algorithms into deployment. Any "bad" choice which accidentally occurs in an online experiment can possibly lower user satisfaction and therefore leads to decreased profit. Online experiments also require a long time before actual user feedback can be received. In theory, these constraints can be eliminated by taking an alternative approach, i.e. offline evaluation.

In the setup of offline evaluation, only historical datasets are provided. The algorithm iterates through all logged data points in the historical dataset and predicts an arm at each iteration based on the information provided. In many cases, the predicted arm doesn't necessarily match the logged arm, i.e. the reward of the predicted arm is not revealed due to the mismatch. This addresses the main challenge of the offline evaluation of the situation where only a partial truth is revealed in the reality. On the contrary, an online evaluation can overcome the challenge easily by simply showing users the predicted content.

Some discussions around the differences between the online experiment and the offline experiment can be found in the studies such as [3] and [8]. Beel [3] argued in the context of recommendation system studied that results from offline evaluations often contradicted to what was shown in online evaluations. The reason cited being human factors, which had major impact on the results, were usually not considered in offline evaluations. In spite of criticism in empirical studies, offline evaluation is still the most common evaluation method in recommender systems. It has demonstrated accuracy in various fields, [13] and [19].

2.4 Loss Encoding and Evaluation Metrics

For the purpose of measuring whether the user is satisfied with a predicted content (reward), there are several evaluation metrics that can be used. An immediate indicator is CTR, click through rate. It is a binary variable determining whether the user has clicked on a predicted content (1) or not clicked (0). This variable can be encoded with 0/1 or it can be encoded as -1/1. In this setup, each user visit is treated independently, meaning that every user visit is treated as it comes from a new user. Hence the user satisfaction is only measured greedily for each user visit. CTR is very efficient when encoding the loss in a short term, but it does not show effects on users in the long term.
Another evaluation metric that is used in this master thesis is conversion rate. It is calculated by dividing the number of conversions over the number of user visits. The conversion rate shows the potential financial impact on the predicted content. However, it does not indicate the performance of the predicted content because other factors may have strong effects when the conversion rate increases, i.e., weekday and people who visit sites. Also, all user visits are not intended to convert. For instance, the user can open the site accidentally without knowing its content.

3
Methods

This chapter describes three datasets that are used in the thesis and explains how the multi-armed bandit solutions are implemented on each of those datasets. A detailed explanation of the evaluation setup can be found in section 3.4 of this chapter.

3.1 Data

3.1.1 Generated dataset

A generated dataset using the procedure described by [15] was used to simulate a news recommendation problem. This dataset simulates choosing one out of three articles to show to users as they visit a site. The data also simulates having some contextual information about the users. The dataset contains three different arms. Each arm represents showing one article to a user. The distribution of the logged arms was uniform, i.e. in each line of logged data the arm shown was selected with an equal probability of 1/3. The reward for the arms was modeled by a variable called click. The value for click was 1 if a user clicked or 0 if they did not click. A context vector was used for identifying whether a user had clicked sports articles or political articles in the past. The vector consisted of two binary variables, i.e. clicked_sports, and clicked_politics.
In the generated dataset, the clicks where determined by a variable called $click\_factor$ which was calculated from the following formula:

$$\begin{aligned} click\_factor = arm\_baseline + sports\_coef \times clicked\_sports \\ + politics\_coef \times clicked\_politics \end{aligned} \tag{3.1}$$

The variable click_factor is fraction determining whether a user clicked on a certain article given a certain context vector. Whenever this fraction was greater than a random variable called rand_draw, click value was set to 1 and it was set to zero otherwise. Table 3.1 shows the values used as coefficients in the formula for each arm. When these coefficients are substituted in Equation 3.1 we can predetermine that arm 1 is best when the user has only clicked on sports articles or articles in both categories. Arm 2 is best if the user hasn't clicked on anything in the past. Arm 3 is best when the user has clicked on political articles. The hope is that the chosen algorithms will eventually learn and converge to these arms. These predefined values are summarized according to context group and are presented in Table 3.2.

Table 3.1: Coefficients used to generate clicks

Arm	arm_baseline	sports_coef	politics_coef
1	0.1	0.5	0.1
2	0.2	0.1	0.1
3	0.1	0.1	0.4

Table 3.2: Pre-defined best arm

BestArm	clicked_sports	clicked_politics
1	1	1
1	1	0
3	0	1
2	0	0

3.1.2 Public dataset

The public dataset that was used was produced by the Today module on Yahoo front page [26]. The dataset recorded 45,811,883 user visits to the Today module in May 2009 and was stored in 10 separate files. Each file contains user visits in one weekday. The amount of user clicks varies slightly between different weekdays, for example there are more user visits on a Friday than a Sunday.
The public dataset has the same structure as the generated dataset. Each line of the file represents a recorded user visit which consists of the timestamp at the moment, the article id being presented, the recorded user click and lists of relevant user/article features (see an example line in Table 3.3). The lists of article features at the end of the line represents available articles for the recommendation and their associated contextual features. The presented article was chosen uniformly at random from the list of available articles. The number of available articles is 48 in maximum, meaning that the probability for each arm being chosen was 1/48 in the minimum. Each user and article are associated with six anonymous contextual features. Unlike the generated dataset, the values of contextual features in the public dataset are float numbers. Feature 1 was constantly equal to 1. Features 2-6 represent the 5 membership features which were constructed via conjoint analysis with a bilinear model [26].

Table 3.3: An example of one line of data from Yahoo front page

Field	Values
$TimeStamp$	1241160900
$Displayedarticleid$	109513
$Userclick$	0
$Userfeatures$	|user 2:0.000012 3:0.000000 4:0.000006 5:0.000023 ...
$Articlefeatures$	|109498 2:0.306008 3:0.000450 4:0.077048 5:0.230439 ...

3.1.3 Company dataset

The company dataset came from interaction logs of user visits to one of the company A's website. It contains 10,641 data points, two categorical feature variables with five main categories each. The website had two slots to optimize and each log data shows what the user saw on those two slots together with the probability of them seeing the content in those slots. The data also shows users final feedback, represented by one reward variable together with which option user preferred when the reward was positive. The variables for this dataset are summarized below:
Number of content to choose from: 3
Number of the combination of slot positions: 6
Number of content features: 2
Number of reward features: 1
Number of data points: 10,641

3.2 Model

The general contextual bandit setting can be described as consisting of t = 1,2,3 ... discrete trials. At each trial t, these three basic steps are followed:

1. The world presents a context as a feature vector x_t.
2. The learning algorithm chooses an arm a_t from a set of m possible arms.
3. The world presents a reward $r_{t,a}$ for the action chosen at step 2.

We chose to model the reward and contextual information using ridge - regression [24]. This is a linear model that we use to fit feature information in order to predict average reward for each action. We use the same contextual bandit formulation as that described in [13]. In this formulation, the ridge regression model stores information about a set of arms A together with their observed context and rewards thus far. The ridge regression model uses the context to make predictions for the expected payoff from each arm. Given this prediction, different arm selection policies/algorithms decide on an arm to present to a user. Afterwards, the ridge regression model observes a reward $r_{t,a}$, based on how the user responded to the displayed arm. We model the average/expected reward per arm as a function of context and some unknown coefficient vector $\boldsymbol{\theta}_{\boldsymbol{a}}^{*}$ as summarized by the formula below:

$$E[r_t|x_{t,a}] = x_{t,a}^{T}\boldsymbol{\theta}_{\boldsymbol{a}}^{*} \tag{3.2}$$

With this formulation, our model stores different coefficients $\boldsymbol{\theta}$ for each arm and learns about the arms in a disjoint manner. We can think of $\boldsymbol{\theta}$ as initially unknown weights that determine the importance of a feature in predicting the reward. These weights are updated every time a model receives feedback about a prediction. Additionally, since the model is updated from logged data, the arm picked by our model is not always the same as the one that a user at time t observed. When this is the case, we ignore this logged observation. Hence our model only learns from the cases where the arm picked by the model match with what the user saw. This means that if data from logging policy has a high bias towards one arm, this bias will also affect

our model. To offset the potential impact of this bias, we use Inverse Propensity Scoring in our evaluation especially when the logging policy was not displaying arms uniformly at random. The details of this method are described in Section 3.4.
As described on [13], the coefficient vector $\boldsymbol{\theta_a}$ for each arm is derived from ridge-regression model where $\boldsymbol{\theta_a}$ is estimated using the formula below:

$$\boldsymbol{\theta_a} = (D_a^\intercal D_a + I_d)^{-1} D_a^\intercal c_a \tag{3.3}$$

In the above equation D_a is an $m \times d$ dimensional matrix with d number of features for m observations. The symbol c_a represents a response vector with m responses to being shown arm a.
In our setup, every arm stores a covariate matrix $\boldsymbol{A}$ and a weight vector $\boldsymbol{b}$. These are the two components needed to compute the coefficient vector $\boldsymbol{\theta_a}$. A is initially set to identity matrix of size d, the number of features used, and b is a zero vector of size d. If a reward is observed for that arm then A and b are updated as shown below:

$$A_{a,t} = A_{a,t} + x_{t,a} \times x_{t,a}^T \tag{3.4}$$

$$b_{a,t} = b_{a,t} + r_t \times x_{t,a} \tag{3.5}$$

Using matrix A from equation 3.4 and vector b from Equation 3.5, $\boldsymbol{\theta_a}$ is computed from the following formula:

$$\boldsymbol{\theta_a} = A_{a,t}^{-1} \times b_{a,t} \tag{3.6}$$

Different arm selection policies use the same model to calculate an expected payoff for each arm. They differ slightly in how they handle prediction and arm suggestion. We outline the differences on Section 3.3.
The chosen model makes several assumptions about the problem to simplify the analysis. Although some of these assumptions could be avoided with more existing approaches, we decided that a more general solution across several datasets would be better. These are outlined below:

1. There are no interaction effects between different arms
2. The sequence of events (user visits) are independent and uniformly distributed
3. There is a linear relationship between context features and observed reward

3.3 Policies

3.3.1 E-Greedy

With an Epsilon-Greedy arm selection policy, the algorithm follows the steps outlined below at each trial t:

1. The world presents a context x_t
2. The algorithm uses a ridge regression model to predict reward r_a for each arm
3. Epsilon percentage of time the algorithm presents a random arm otherwise it presents the best arm
4. The algorithm observes a reward r_a and learns from it.

Epsilon in our implementation is a configurable parameter describing the percentage of time the policy should make a random choice. Hence it is the exploration rate. In our setup, the 'best arm' is the arm that has the highest expected reward at time t according to the formula:

$$E[r_t|x_{t,a}] = x_{t,a}^T \boldsymbol{\theta}_{\boldsymbol{a}}^* \tag{3.7}$$

If there is more than one arm that has the highest expected reward, the policy breaks the tie randomly. Our implementation keeps track of when the choice was random and when the choice was based on the highest expected reward. We feed these choices into the evaluator as probabilities of the algorithm to suggest this arm. Our IPS estimated evaluator uses these probabilities to estimate the impact of the policy.

3.3.2 LinUCB

We are using the LinUCB algorithm from [13]. This version of UCB selects the arm that has the highest 'confidence bound'. The creators of the algorithm motivate their confidence estimate as a sum of the expected reward and the standard deviation of the reward for each arm from the ridge-regression model. They argue that a ridge-regression model can have a Bayesian interpretation where, the posterior distribution of the coefficient vector p($\boldsymbol{\theta}_{\boldsymbol{a}}$) is Gaussian with mean $\boldsymbol{\theta}_{\boldsymbol{a}}$ and covariance $A_{a,t}^{-1}$. Hence when applied to the context, the predictive mean becomes $x_{t,a}^\intercal \theta_a$ and predictive variance becomes $x_{t,a}^\intercal \theta_a x_{t,a}$. This algorithm chooses an arm that maximizes the tradeoff between increasing payoff from a higher performing arm(from the mean) and reducing the uncertainty of different arms' performance (from the standard deviation) with the formula shown below:

$$x_{t,a}^\intercal \theta_a + \alpha\sqrt{x_{t,a}^\intercal A^{-1} x_{t,a}} \tag{3.8}$$

where α is an input parameter that needs to be optimized . Hence the general outline of steps for the LinUCB policy becomes:

1. The world presents a context x_t
2. The algorithm uses a ridge regression model to predict reward r for each arm
3. The algorithm chooses the arm with the highest, predicted average $reward + \alpha(variance)$
4. The algorithm observes a reward r_a and learns from it

3.4 Evaluation Setup

In order to judge the performance of our model and arm selection policies (e-greedy and LinUCB), we use logged data to estimate an expected average reward for each policy. For each logged dataset, we know what the user saw and how they responded. As each policy suggests a new arm, our model only learns from this logged data if the suggested arm is the same as what the user saw. In essence, our model is only sampling from a subset of logs where policy arm suggestions are the same as logged observations. This offline setting creates two potential challenges namely, model

bias stemming from bias from the logging policy and large data requirements since most of the logged lines will be ignored.
The size of data challenge is especially critical when evaluating combinatorial bandits. These are a class of problems where evaluations are performed according to slates, where a slate is a combination of arms. If the size of a slate is large, the probability of a full slate match between a new and logging policy is reduced. In such cases, [1] argues that more advanced techniques for estimating the impact of a policy become necessary. For this study we explore high number of arm choices but the number of selected arms at each round is still small. Hence we only use two techniques to estimate reward differences between policies. The first estimation technique is called Direct Method (DM) or naive. The second method is called Inverse Propensity Scoring(IPS). The proceeding subsections, 3.4.1 and 3.4.2, explain how the evaluation is carried out and how the estimators work. The value of a new policy will be noted by V_π and the value of a logging policy will be denoted by V_μ.

3.4.1 Direct Estimator

This estimator uses a model to estimate the expected reward conditioned upon the context x and arm a. Using the formulation described in [7], we treat the model as a function g(x) that is learning to predict reward r_a when given a dataset S consisting of tuples $\{(x, h, a, r_a)\}$, where:

x is the context,
h is the history of previous observations,
a is the arm chosen by the policy,
r_a is the reward for the chosen arm.

Since the policy is choosing an arm according to the probability p(a | x, h) then the direct estimator is summarized by the formula below:

$$V_{DM} = \frac{1}{|S|} \sum_{x \in S} g(x) \tag{3.9}$$

This estimator is only as good as the model. If g(x) is a good estimator of the expected reward then V_{DM} is also good. Its drawback is that it does not take into account the biases that may exist in the logging policy. Therefore it should only be used when the logging policy was selecting arms uniformly at random.

3.4.2 Inverse propensity score

This estimation method re-weights observed reward according to a probability distribution of the arms from logging policy μ and the new target policy π. We are using a formula from [7]. This method reduces variance and biases from logging policy. The requirement is that every arm considered in the logging policy should have a probability greater than zero for every context group considered.

$$V_{IPS} = \frac{1}{t} \sum_{i=1}^{t} r_t \times \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)} \tag{3.10}$$

3.4.3 Evaluation of Generated dataset

The unique feature of this dataset is that we know the true optimal arm for each context. These are summarised on Table 3.2. It also has a low number of arms and the context features are categorical hence we can easily drill down the performance at a context level and see if each algorithm converges at the true best arm. Consequently, in addition to looking at the total expected reward per policy, we asses if the algorithm converges to the true best arm for each context. We also analyze how this convergence relates to the overall performance of the policy and asses the stability of convergence across multiple runs. Since the simulated arm shown is randomly distributed we only use direct estimator on this dataset during the validation phase.

To assess the validity of our estimators we perform a small online simulation. Here we use coefficients from generated dataset (see Table 3.1) together with some degree of noise to simulate online interaction with our policies. The simulation follows the following steps:

1. The simulator pre generates T tuples of context according to the following steps:

   $$clicked\_sports = sample\_no\_replace(choice = [0, 1], probability = (0.6, 0.4))$$
   $$clicked\_politics = sample\_no\_replace(choice = [0, 1], probability = (0.7, 0.3))$$
   $$X = concat(clicked\_sports, clicked\_politics)$$

2. The simulator feeds tuples from X one at a time to a policy π as x_t
3. The policy decides on an arm a_t to send to the simulator according to a probability $\pi(a_t|x_t)$
4. The simulator sends a click response r_t equal to clicked whenever the $click\_factor$ is greater than a random variable and not clicked otherwize. The variable $click\_factor$ and r_t are calculated as shown below:

   $$rand\_draw = sample\_normal\_distibution()$$
   $$click\_factor = arm\_baseline(a_t) + sports\_coef(a_t) \times clicked\_sports(x_t)$$
   $$+ politics\_coef(a_t) \times clicked\_politics(x_t)$$
   $$r_t = (rand\_draw_t < click\_factor?1 : 0)$$

5. The policy π observes r_t and updates its ridge-regression model according to formula Equations 3.4 and 3.5.

Above steps ensure that the probability of having a click is conditioned on arm shown a_t and context x_t, i.e probability $(r_t = 1|a_t, x_t)$. We run the simulation for T rounds on e-greedy and LinUCb policy and compare the results from our offline estimators. Since the same linear model that was used to generate the data is used for the online simulation, the difference between online estimation and offline estimation should tell us the impact of using offline evaluation.

3.4.4 Evaluation of Public dataset

For this dataset, the unique feature is the number of arms. In our setup this data contains a large number of arms and non categorical features. In effect we needed more data for learning and evaluation. To facilitate this, we use batch processing and use the performance per batch as the step size in our performance graph. The data has 10 days worth of data but due to time constraint we could only use two days worth of data. We selected one day for algorithm tuning and the other day for validation. Since the logging policy showed all the arms uniformly at random, we only use the direct estimator for the validation as it is still statistically valid.

3.4.5 Evaluation of Company dataset

The company dataset had few option of content to choose from. In this case however, we were interested in the effect of having different combination of layout. Therefore each the bandit arm is the layout of those three content. Hence we have a total of 6 arms. The logging policy from which the data came from did not show the arms uniformly at random, therefore we employ IPS in our evaluation. We make a comparison of the total average reward under each arm selection policy to assess performance.

4
Results

In this section we describe the outcome of our study. The section is divided into three parts that correspond to the datasets that were examined. A comparison of the results across the datasets will be done in Chapter 5.

4.1 Generated dataset results

The overall performance of the generated dataset for two arm selection policies, E-greedy and LinUCB, indicate that both policies improved click-through rate when compared to a random policy. The mean regret over time during training are shown on Figure 4.1 and during validation are shown on Figure 4.2. The regret for both policies is lower than the regret when using a random policy. Drilling down to the contextual groups, Figure 4.3, Figure 4.4 and Figure 4.4 show how often each policy recommended a certain arm to a particular context group. When the most chosen arm per context is compared to the predefined best arm from Table 3.2, the graphs indicate that both policies converge to the correct arm in most runs. E-greedy policy sometimes converged to the wrong arm when the context was a user who clicked both sports and politics, see Figure 4.4.
The main differences observed between the policies after tuning the algorithm parameters is a slightly more efficient learning rate by LinUCB policy as seen in Figure 4.1 and a more consistent convergence to the correct arm cross multiple runs. The experiments revealed that when the true average rewards are close, E-Greedy sometimes converges to the wrong arm, see Figure 4.3 where E-Greedy converged to Arm 3 which has a true average reward 0.6 instead of Arm 1 which has a true average reward of 0.7 when the context is clicked both sport and politics.

This data also showed consistent result between online and offline evaluation which suggests that when the assumed conditions in the offline evaluation are similar to conditions in the online setting, the result of the offline evaluation are valid. The conditions are similar because the linear model used to generate the offline data was also used to simulate the online interaction. Figure 4.5 and Figure 4.6 summarize the result from this online interaction where the arm each context group converged to is the same as the offline arm Figure 4.2 and Figure 4.4. The average reward for converged arm per context is also similar.
In more detail, from Figure 4.1 we observe that both E-greedy and LinUCB policies obtain an average reward around 0.18 when they have converged. In the first 4500 attempts, the mean regret of both policies are fairly unstable. It can be seen as the

learning phase of our model where the model learns continuously from the logged data by updating the weight of the respective arm. After the learning phase, the average rewards become comparatively stable and therefore we can declare that our models have converged. The time it takes for the greedy model to converge depended on the exploration rate (epsilon) which we specifies in the setting. In this specific example which we have shown in Figure 4.1, the exploration rate is 10 for the greedy policy.

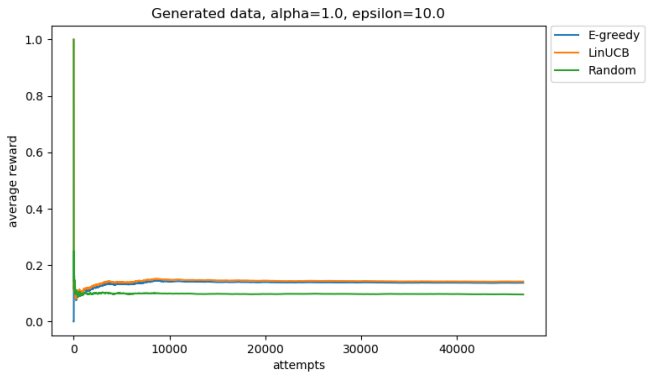


Figure 4.1: Learning of the generated dataset using naive estimator (average over 10 experiments)

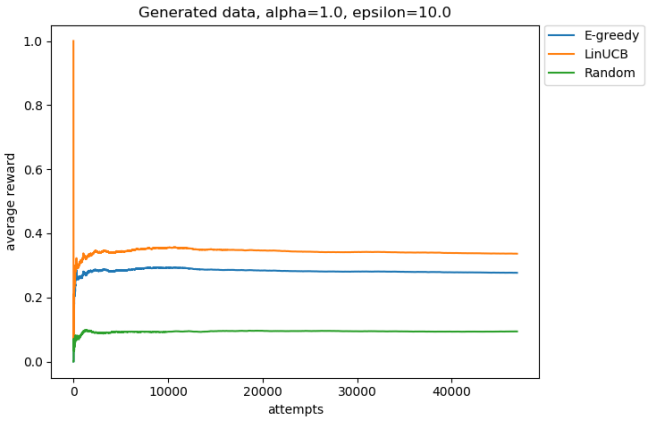


Figure 4.2: Validation of the generated dataset using naive estimator (average over 10 experiments)

Usually, a high exploration rate can shorten the total convergence time while it also worsens the average click through rate. For the LinUCB policy, we use a control variable alpha to control the standard deviation interval of the predicted average reward. We notice that the average reward for LinUCB is optimal only when alpha is between 0 and 1. A high control variable results in high standard deviation which can make the impact of the predicted payoff diminishes. In Figure 4.1, we are using 1 as alpha for the LinUCB policy.

We use direct estimator to estimate user feedback when the model is updated. In this setting, the script iterates through all data points and only updates for those logs with the same arms as the predicted arms. Therefore a big part of the dataset is ignored. In Figure 4.1, we use 47000 data points in total. Although the actual attempts for updating the model are around 15000.

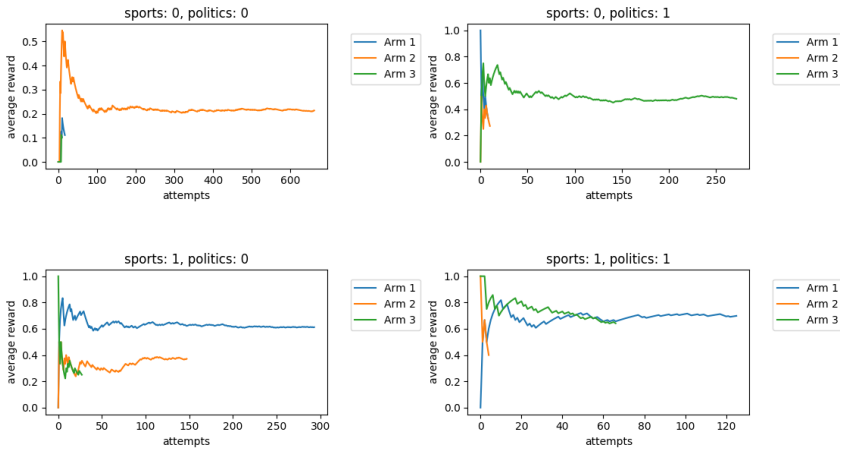


Figure 4.3: Offline evaluation generated dataset E-Greedy with 7% exploration

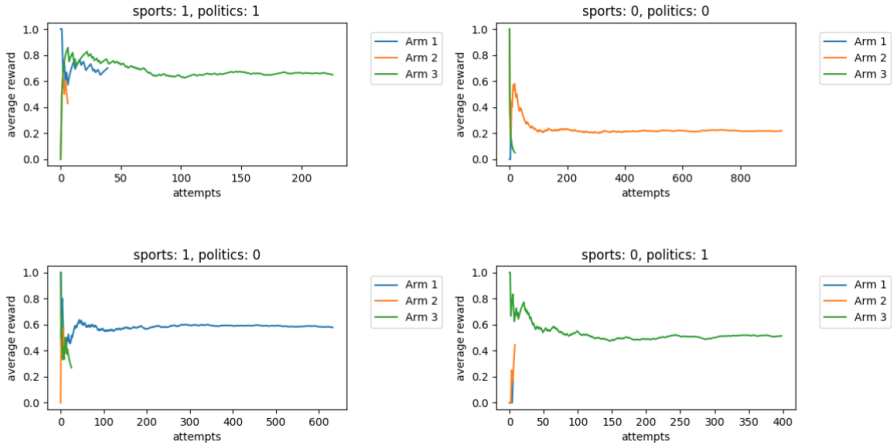


Figure 4.4: Offline evaluation generated dataset E-Greedy with 7% exploration - bad converge on sports:1 and politics:1

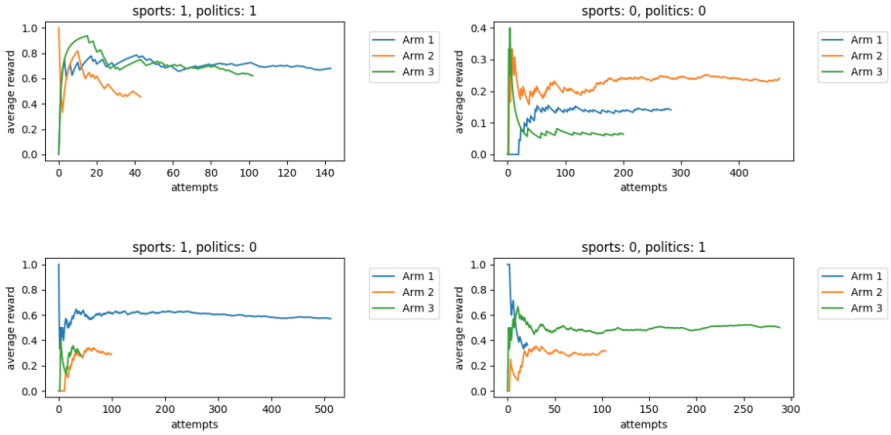


Figure 4.5: Offline evaluation generated dataset LinUCB with alpha = 7

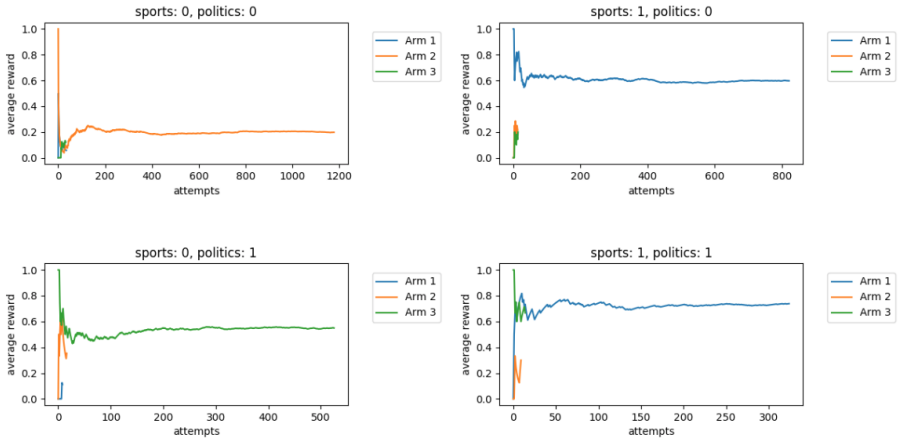


Figure 4.6: Online validation generated dataset E-Greedy with 7% exploration

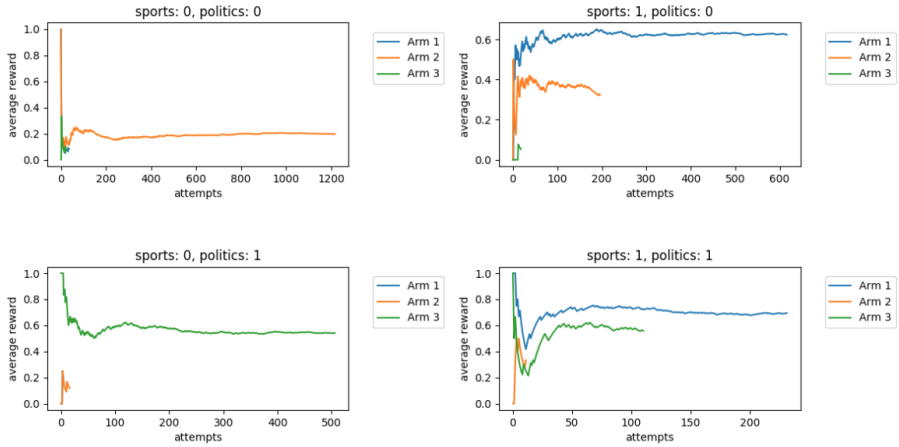


Figure 4.7: Online validation generated dataset LinUCB with alpha = 5

4.2 Public dataset results

The overall performance of policies on public dataset suggests that LinUCB is better than E-Greedy policy both on the training dataset Figure 4.8 and on the validation dataset on Figure 4.9. This suggests that LinUCB is a better policy for this dataset. Unlike the generated dataset, here we observed a significant difference in expected reward from different policies and not just a difference in the learning rate. The experiment also revealed that the expected reward changes over time since even a random policy appears to have an increasing average reward over time on both the training and validation dataset. Both policies under investigation, E-Greedy and LinUCB, appear to adapt to this changing average well and there was no need to supplement the context with time-related features.
From a practical standpoint, however, the LinUCB model was much harder to fine-tune. Changing alpha parameter significantly affected the performance of the algorithm and incorrect values resulted in lower performance compared to E-Greedy. We suspect this is a result of having a very large number of arms, (on average 20 possible articles at every time point) accompanied with high variance in performance of each arm. We also observed that the static nature of LinUCB resulted in higher number of ignored log values when the predicted arm was different from observed. This implies that offline evaluation of LinUCB policy requires a higher number of logged data points than the E-Greedy policy.

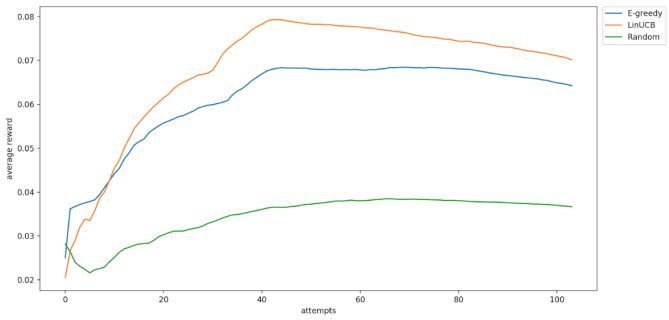


Figure 4.8: Learning of public dataset using naive estimator

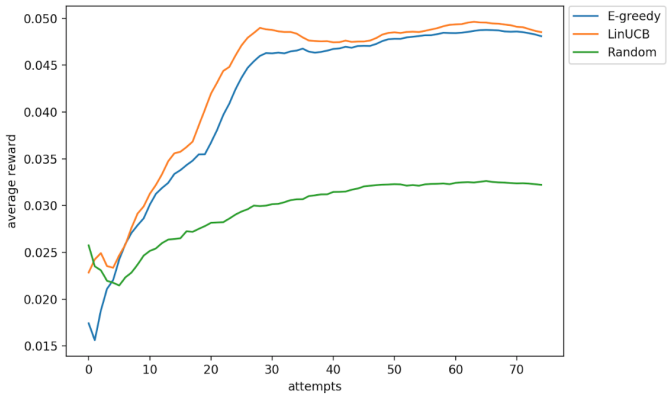


Figure 4.9: Validation of public dataset using naive estimator

4.3 Company dataset results

The company dataset results in Figure 4.11 confirm that both greedy policy and ucb policy perform better than random policy. However, the performance of the ucb policy seems quite unstable compared to the greedy policy. The results vary significantly in different executions with different alpha parameters (see Figure 4.11, Figure 4.13 and Figure 4.15). We observed from our offline experiments that the ucb policy mostly chooses the first best arm and ignores other arms if the alpha parameter is low. With a high alpha parameter instead, the ucb policy always tries different arms and rarely picks a same arm twice, which leads to the LinUCB policy skipping most of logged data points due to this randomness. We also realize that it is essential to fit the ucb policy with enough data points in order to build a good model. From our observations with company datasets, we can conclude that the greedy policy is often performed better, thereby providing more stable predictions than the ucb policy for small datasets.

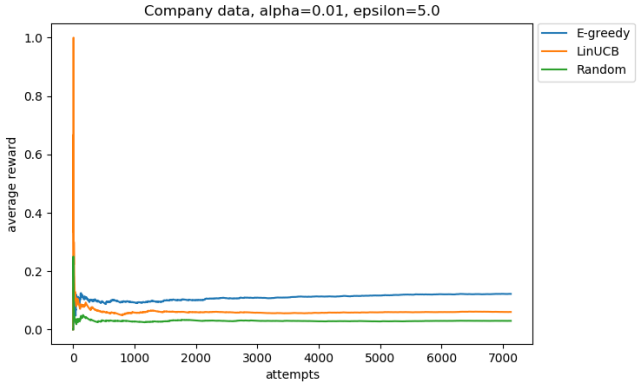


Figure 4.10:
Learning of company dataset using naive estimator
Run 1, alpha=0.01, epsilon=5

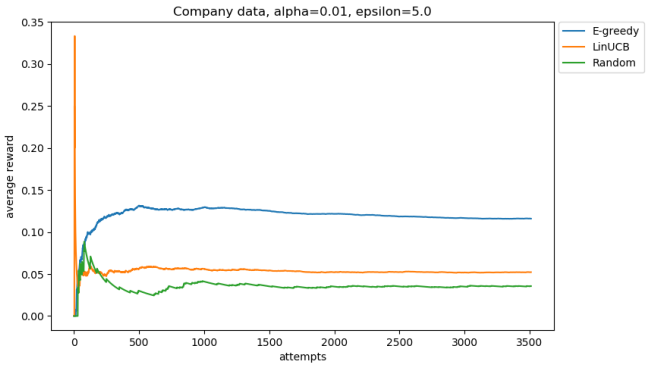


Figure 4.11:
Validation of company dataset using naive estimator
Run 1, alpha=0.01, epsilon=5

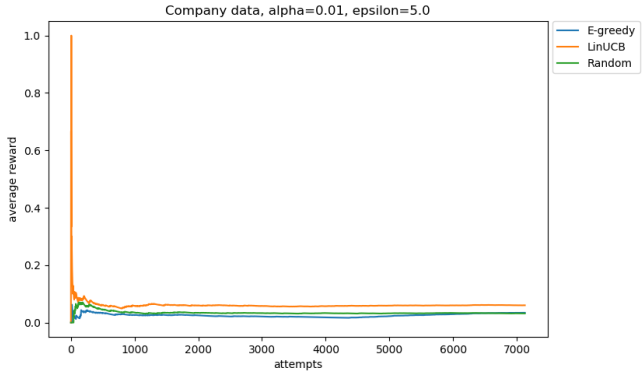


Figure 4.12:
Learning of company dataset using naive estimator
Run 2, alpha=0.01, epsilon=5

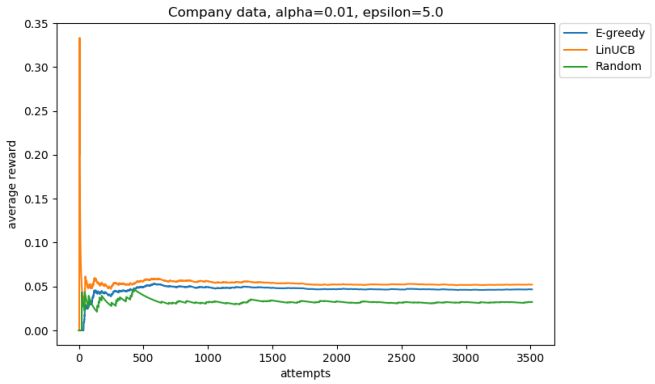


Figure 4.13:
Validation of company dataset using naive estimator
Run 2, alpha=0.01, epsilon=5

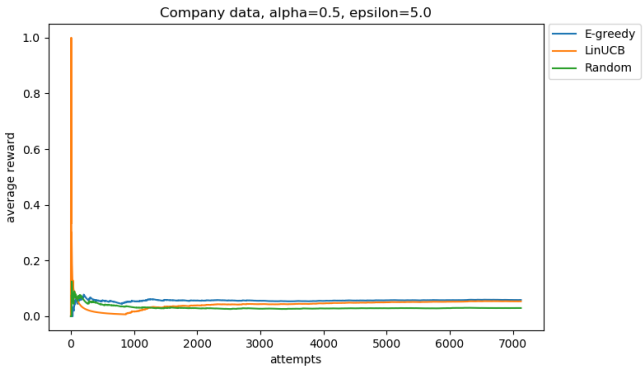


Figure 4.14:
Learning of company dataset using naive estimator
Run 3, alpha=0.5, epsilon=5

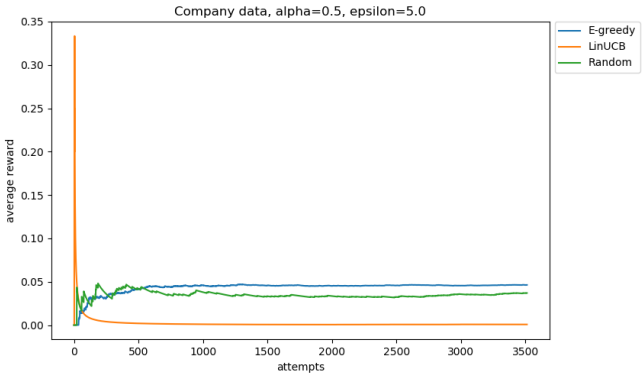


Figure 4.15:
Validation of company dataset using IPS estimator
Run 3, alpha=0.5, epsilon=5

5
Conclusion

This thesis aims to provide a comparison of two arm selection policies in the multi-armed bandit problem by applying the policies to three different datasets, where each dataset simulates a common web content optimization problem in real world. The policies are first tested on a generated dataset, which simulates a news recommendation problem with limited arms and categorical features. Then the same policies are applied on a larger public dataset with similar news recommendation setting as well as an increased number of arms and data points. Finally, the policies are tested on a dataset from company A which records the user visits of a website, again with a small number of arms.

In all experiments, both policies have shown improved results compared to the random policy. The results from the company datasets suggest that E-Greedy policy is often more stable, with better performance than LinUCB policy for small datasets with a limited number of data points. For large datasets, LinUCB policy learns quicker at beginning and obtains overall higher performance than E-Greedy policy in multiple tries. The LinUCB model is however difficult to fine-tune. The experiments indicate that the performance of LinUCB policy depends heavily on the choice of the alpha parameter. A high alpha parameter allows LinUCB policy to try new arms more frequently, meaning that a lot data points are required in order to make good predictions.

The datasets consist of recorded user visits in the past and appropriate offline evaluation techniques are implemented to build the models. To further validate the results from offline experiments, an online experiment is conducted on a generated dataset. The experiments on the generated dataset show consistent results between the online experiments and the offline experiments, concluding that the offline evaluation approach in this thesis models the online environment in a proper way.

These differences in algorithm performance across datasets further emphasises the need for easy to use libraries, that can be used by practitioners to compare algorithms performance before they choose a policy to employ. Our evaluation only worked with one prediction model, namely ridge-regression and one type of reward encoding. Future work can improve on this to include other prediction models such as soft-max that can easily be applied in a multi-classification problem and explore the effect of using different reward encodings. Additionally, implementation of Thompson Sampling would be the logical next step in supplementing our analysis for further studies.

Looking back at the motivating questions highlighted in the Introduction chapter, we think practical solutions for the application of MAB algorithms is still an important area of research that needs further efforts. Findings from our study highlight

interesting areas that can benefit from further empirical evidence. Below we have highlighted open questions that might be useful for practitioners.

1. **How much logged-data do they need to make useful estimates of effectiveness**
   The findings for this study suggest that LinUCB has a higher data requirement than E-Greedy when performing offline evaluation. But the results do not generalize into exactly how much data is needed. The results from companyA comparison suggest that around 10000 data points where not enough to reach a stable result. However, we suspect the answer requires further dive into the actual composition of data to determine distribution across different context. We also suspect that other factors such as exploration rate can have a significant effect on exactly how much data is needed. Further studies that could explore the relationship between data requirements and policy option could be very useful.
2. **What policy is the most promising for a specific application**
   This study found that different algorithms worked best on different datasets but the comparison can only be performed after the tuning process. The study also found that algorithm tuning had significant performance implications especially for LinUCB algorithm. Some of the interesting areas for further research could be done on the transferability of parameters from offline evaluations to online experiments. This can further inform practitioners on how to apply their chosen policies. It may also be useful to formalize the process around which parameter choices should be tested to make sure that critical options are tested.
3. **How confident can they be with their estimate**
   This study found consistent results between offline and online and offline evaluation for the generated dataset. Such comparison can be further investigated in future studies to give empirical evidence on how consistent such a result can be. In order to be confident about our results on the other datasets, an online experiment would have to be performed. This is of course costly. But there is a need for future studies to explore the validity of offline experiments with online experiments.

Finally, future studies could also focus on reducing problem setup assumptions to make the model more realistic, for example removing the assumption that all user visits are independent and accounting for possible interaction effect between different arms. Future studies should also focus on solving critical issues surrounding privacy concerns [17] for users personalization algorithms are applied.

Bibliography

[1] Deepak Agarwal et al. "Laser: A scalable response prediction platform for online advertising". In: *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM. 2014, pp. 173–182.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[3] Joeran Beel et al. "A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation". In: *Proceedings of the international workshop on reproducibility and replication in recommender systems evaluation*. ACM. 2013, pp. 7–14.

[4] Alberto Bietti, Alekh Agarwal, and John Langford. "Practical Evaluation and Optimization of Contextual Bandit Algorithms". In: *arXiv preprint arXiv:1802.04064* (2018).

[5] Peter Brusilovski, Alfred Kobsa, and Wolfgang Nejdl. *The adaptive web: methods and strategies of web personalization*. Vol. 4321. Springer Science & Business Media, 2007.

[6] Olivier Chapelle and Lihong Li. "An empirical evaluation of thompson sampling". In: *Advances in neural information processing systems*. 2011, pp. 2249–2257.

[7] Miroslav Dudık, John Langford, and Lihong Li. "Doubly robust policy evaluation and learning". In: *arXiv preprint arXiv:1103.4601* (2011).

[8] Florent Garcin et al. "Offline and online evaluation of news recommender systems at swissinfo. ch". In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 2014, pp. 169–176.

[9] Nathaniel Good et al. "Combining collaborative filtering with personal agents for better recommendations". In: *AAAI/IAAI*. 1999, pp. 439–446.

[10] Thore Graepel et al. "Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine". In: Omnipress. 2010.

[11] Jaya Kawale et al. "Efficient Thompson Sampling for Online Matrix-Factorization Recommendation". In: *Advances in neural information processing systems*. 2015, pp. 1297–1305.

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

[13] 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.

[14] Tyler Lu, Dávid Pál, and Martin Pál. "Contextual multi-armed bandits". In: *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*. 2010, pp. 485–492.

[15] John Maxwell. *Contextual Bandits: LinUCB*. 2017. URL: http://john-maxwell.com/post/2017-03-17/.

[16] Dunja Mladenic. "Text-learning and related intelligent agents: a survey". In: *IEEE intelligent systems and their applications* 14.4 (1999), pp. 44–54.

[17] S. Neel and A. Roth. "Mitigating Bias in Adaptive Data Gathering via Differential Privacy". In: *ArXiv e-prints* (June 2018). arXiv: 1806.02329 [cs.LG].

[18] Seung-Taek Park et al. "Naıve filterbots for robust cold-start recommendations". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 699–705.

[19] Rokach Ricci, Paul Shapira, and Kantor. "Recommender systems handbook". In: *Recommender systems handbook*. Springer. 2011, pp. 1–35.

[20] Herbert Robbins. "Some aspects of the sequential design of experiments". In: *Bull. Amer. Math. Soc.* 58.5 (Sept. 1952), pp. 527–535. URL: https://projecteuclid.org:443/euclid.bams/1183517370.

[21] Daniel Russo et al. "A Tutorial on Thompson Sampling". In: *arXiv preprint arXiv:1707.02038* (2017).

[22] Niranjan Srinivas et al. "Gaussian process optimization in the bandit setting: No regret and experimental design". In: *arXiv preprint arXiv:0912.3995* (2009).

[23] William R Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294.

[24] Andrei Nikolaevich Tikhonov. "On the solution of ill-posed problems and the method of regularization". In: *Doklady Akademii Nauk*. Vol. 151. 3. Russian Academy of Sciences. 1963, pp. 501–504.

[25] Joannes Vermorel and Mehryar Mohri. "Multi-armed bandit algorithms and empirical evaluation". In: *European conference on machine learning*. Springer. 2005, pp. 437–448.

[26] Webscope dataset Yahoo! *ydata-frontpage-todaymodule-clicks-v1_0*. 2009. URL: http://labs.yahoo.com%5C/Academic%5C_Relations.