Foundations and Trends® in Information Retrieval

Bandit Algorithms in Information Retrieval

Suggested Citation: Dorota Głowacka (2019), "Bandit Algorithms in Information Retrieval", Foundations and Trends® in Information Retrieval: Vol. 13, No. 4, pp 299–424. DOI: 10.1561/1500000067.

Dorota Głowacka
University of Helsinki
glowacka@cs.helsinki.fi

now
the essence of knowledge
Boston — Delft

Contents

Bandit Algorithms in Information Retrieval

Dorota Głowacka

University of Helsinki; glowacka@cs.helsinki.fi

ABSTRACT

Bandit algorithms, named after casino slot machines sometimes known as "one-armed bandits", fall into a broad category of stochastic scheduling problems. In the setting with multiple arms, each arm generates a reward with a given probability. The gambler's aim is to find the arm producing the highest payoff and then continue playing in order to accumulate the maximum reward possible. However, having only a limited number of plays, the gambler is faced with a dilemma: should he play the arm currently known to produce the highest reward or should he keep on trying other arms in the hope of finding a better paying one? This problem formulation is easily applicable to many real-life scenarios, hence in recent years there has been an increased interest in developing bandit algorithms for a range of applications. In information retrieval and recommender systems, bandit algorithms, which are simple to implement and do not require any training data, have been particularly popular in online personalization, online ranker evaluation and search engine optimization. This survey provides a brief overview of bandit algorithms designed to tackle specific issues in information retrieval and recommendation and, where applicable, it describes how they were applied in practice.

Dorota Głowacka (2019), "Bandit Algorithms in Information Retrieval", Foundations and Trends® in Information Retrieval: Vol. 13, No. 4, pp 299–424. DOI: 10.1561/1500000067.

1

Introduction

Over the last decade there has been an increased interest in application of bandit algorithms in information retrieval (IR) and recommender systems. The aim of this survey is to provide an overview of bandit algorithms inspired by various aspects of IR, such as click models, online ranker evaluation, personalization or the cold-start problem. Each section of the survey focuses on a specific IR problem and aims to explain how it was addressed with various bandit approaches. Within each section, all the algorithms are presented in chronological order. The goal is to show how specific concepts related to bandit algorithms, e.g. graph clustering with bandits, or a specific family of bandit algorithms, e.g. dueling bandits developed over time. Gathering all this information in one place allows us to explain the impact of IR on the development of new bandit algorithms as well as the impact of bandit algorithms on the development of new methods in IR. The survey covers papers published up to the end of 2017.

Why Bandits?

Bandit algorithms derive their name from casino slot machines, sometimes referred to as one-armed bandits. In this scenario, a gambler is

faced with a row of such machines. The gambler has to make a number of decisions, such as which machines to play or how many times to play each machine. The problem is that each machine provides a random reward from a probability distribution specific to that machine. The gambler aims to maximize the sum of the rewards by playing different machines. Thus, the gambler needs to make a trade-off between exploiting the machine with the highest expected payoff so far and exploring other machines to get more information about their expected payoffs.

In the 1950's Herbert Robbins realized the importance of the problem and constructed convergent population selection strategies for sequential design of experiments (Robbins, 1985). A couple of decades later John Gittens constructed a theorem, called the Gittins index, that gave an optimal policy for maximizing the expected discounted reward (Gittins, 1979). Later on, some approximate solutions based on *epsilon* strategies (Sutton and Barto, 1998) as well as Bayesian methods, such as Thompson sampling (Thompson, 1933), were developed to solve the bandit problem. The last two decades has seen an immense interest in the study of bandit algorithms, starting with the development of Upper Confidence Bound (UCB) (Agrawal, 1995) strategies. In UCB algorithms, however, every bandit arm is independent and does not pass any information about its payoff generating distribution to other bandit arms. This led to the development of linear and contextual bandits (Auer, 2002; Li *et al.*, 2010b), where a linear dependency between the expected payoff of an arm and its context is assumed. In comparison to independent bandit strategies, linear bandits can lead to elimination of arms with low payoff earlier during the exploration phase thus allowing the player to focus on trying arms with a potentially higher payoff.

There are a number of reasons why bandit algorithms have gained a high level of popularity in many applications. They are quick and easy to implement, they do not require any training data, and they allow for continuous testing/learning, which makes them highly applicable to any online application with a continuous stream of data. Thus, over the years bandits have been applied in many areas: clinical trials (Villar *et al.*, 2015; Williamson *et al.*, 2017), adaptive routing (Awerbuch and Kleinberg, 2008), auctions (Nazerzadeh *et al.*, 2016), financial portfolio design (Shen *et al.*, 2015), cognitive modelling (Głowacka *et al.*, 2009),

games (Kocsis and Szepesvári, 2006), and, as this survey shows, in information retrieval.

Organization of the Survey

The survey is organized as follows. Chapter 2 introduces bandit algorithms and gives a brief overview of four broad classes of bandit algorithms: *epsilon* strategies, independent arms bandits based on upper confidence bound, linear bandits with dependent arms, and Thompson sampling. These broad categories of bandit strategies form the basis of more specialized algorithms discussed in the remaining chapters. Other types of bandit algorithms with specific applications are introduced in relevant chapters rather than being briefly introduced in Chapter 2. Chapter 3 summarizes bandit algorithms inspired by three click models: the *Cascade Model* (Section 3.1), the *Dependent Click Model* (Section 3.2) and the *Position Based Model* 3.3. The following two chapters discuss bandit based approaches to ranking (Chapter 4) and ranker evaluation (Chapter 5). Of particular interest to the reader might be Section 4.1, where the first bandit algorithms applied to ranking are described. Chapter 5 focuses mostly on dueling bandits algorithms and their application to ranking. In Chapter 6, various bandit approaches used in recommender systems are described. The chapter talks about personalization (Section 6.1), social network based bandits (Section 6.2), collaborative filtering with bandits (Section 6.3), optimization through feature learning (Section 6.4) and multiple arms evaluation (Section 6.6). Section 6.1.1 talks in more detail about contextual bandits in the context of advertising and recommender systems by introducing some of the classic algorithms in this area, such as LinUCB (Li *et al.*, 2010a). Finally, Chapter 7 briefly touches on other areas of information retrieval where bandits are gradually introduced, such as short text recommendation (Section 7.1), multimedia retrieval (Section 7.2), and web-page layout optimization (Section 7.3). The appendices contain explanations of the abbreviations and mathematical symbols used throughout the survey.

Who is this Survey Intended for?

The survey is primarily intended for two groups of readers: (1) IR researchers interested in bandit algorithms or more broadly in reinforcement learning, and who would like to know how and where bandits algorithms have been applied in IR; (2) machine learning researchers generally interested in practical applications of machine learning techniques and challenges posed by such practical applications; (3) data scientists interested in algorithmic solutions to issues regularly encountered in information retrieval and recommender systems.

The survey provides a general overview of the bandits methods discussed and as such it should be accessible to anyone who completed introductory to intermediate level courses in machine learning and/or statistics. The reader is advised to consult specific papers referenced throughout the text to learn more about theoretical analysis or implementation details of specific algorithms. All the chapters are self-contained and can be read in isolation, although references to related concepts in other sections are provided throughout the survey. Each section provides a chronological development of a specific approach or family of algorithms, where most of the later developments build upon or improve earlier findings. Chapter 2 is primarily aimed at readers with little knowledge of reinforcement learning and bandits. Readers not familiar with these topics are encouraged to start with this chapter before proceeding to the rest of the survey.

2

Reinforcement Learning and Bandit Algorithms

This chapter provides a brief introduction to reinforcement learning and introduces the main types of bandit algorithms relevant to IR applications.

2.1 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 1998) is about learning from the interaction with the environment how to map situations to actions in order to maximize a reward. Unlike, e.g. supervised learning where training examples are provided, in reinforcement learning, the learner is not given any instructions or examples, but instead must discover, usually through trial and error, which actions yield the most reward. In this sense, reinforcement learning aims to mimic learning processes in biological or cognitive sense. For example, a baby or a toddler (the learner) might discover that his/her volume or frequency of crying (the actions) is correlated with how fast he/she can attract attention (the reward) from his/her carers (the environment). Thus, over time through trial and error the baby can learn that the higher the frequency of his/her crying, the faster the response of the carers, where the response of the carers is the reward and the speed of the response is the value

of the reward – the faster the response, the higher the reward. These basic concepts of reinforcement learning can be easily translated into learning in interactive systems. For example, a recommender system (the learner) makes product suggestions to the end user (the environment) and obtains rewards from the user in the form of clicks or purchases. The more items are clicked or purchased, the higher the reward.

A characteristic specific only to reinforcement learning is the trade-off between exploration and exploitation. To accumulate a high reward, the learner must select actions that, when tried in the past, produced high rewards. However, in order to find actions that give high rewards, the learner has to try actions that it has not selected before. Thus, the agent has to exploit its current knowledge of actions that it has already experienced in order to obtain reward, but it also has to explore new actions in order to possibly improve the reward in the future. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. For example, a recommender system suggested movies from three categories to a specific user: crime drama, action drama and romantic movies (the exploration phase). The user only selected romantic movies to watch. At this stage, the recommender system could simply decide that this is the only category of movies that the user likes and only every recommend that types of movies (the exploitation phase). However, it may also be the case that there are other movie categories never recommended by the system that the user likes even more than romantic movies. It might be worth for the system to suggest movies from other categories not presented before (continue with exploration) in the hope that the user will find movies from some of the newly presented categories even more attractive than romantic movies and thus buys/watches even more movies (increased reward for the system).

Another important aspect of reinforcement learning is that it explicitly considers the whole problem of a goal-directed learner interacting with an uncertain environment. This is in contrast to other areas of machine learning that focus on specific subproblems, such as classification or clustering, without specifying how the ability to perform such a task would be useful. Reinforcement learning agents, on the other hand, have explicit goals, can sense aspects of their environments, and can

choose actions to influence their environments. Additionally, the learner has to operate despite significant uncertainty about the environment it has found itself in.

There are four main subelements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment. A policy defines how the learner will behave in a given situation. In other words, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. A reward signal defines the goal. At each time step, the environment sends to the learner a single number called the reward. The learner's objective is to maximize the cumulative reward it obtains in the long run. The reward thus specifies what are the positive and negative events for the learner. For example, in case of recommender system, the positive event is the user buying or watching the recommended movie. The long term goal of a recommender system would be to maximize the number of suggested movies that the user buys or watches (maximizing the cumulative reward). The reward signal may lead to changes in policy, i.e. if an action selected by a given policy leads to a low reward, then the policy may be altered to select a different action in that situation in the future. For example, if the user tends to ignore action movies, then the recommender system might reduce the number of movies from this categories being presented to the user. While the reward indicates how good a given action was right after it was tried, a value function specifies what is good in the long run, i.e. the value of a state is the total amount of reward a learner can expect to accumulate over the future. For example, a state might provide a low immediate reward but it may still have a high value in the long run because it is regularly followed by other states with high rewards. The fourth element of a reinforcement learning systems is a model of the environment that allows the learner to predict how the environment will behave, e.g. what reward might be expected from a given action taken by the learner.

2.2 What are Bandits?

In this section, we will briefly describe the main aspects of bandits algorithms and introduce some popular bandit algorithms and strategies.

The list of algorithms described in this section is not exhaustive. We only focus on bandit algorithms that are regularly referred to in most of the section of the survey and so the brief description here serves as a quick reference point. Certain family of bandit algorithms that are confined only to one chapter, e.g. dueling bandits (Section 5.1) or graph-based bandits (Section 6.2.1), are only describe in more detail in that particular section.

In terms of reinforcement learning, bandit algorithms provide a simplified evaluative setting that involves learning to act in one situation (or one state) only (Sutton and Barto, 1998). In its simplest formulation, a multi-armed bandit problem, i.e. a bandit with more than one arm, consists of a set of K probability distributions $\langle p_1, \dots p_K \rangle$ with associated expected values $\langle \mu_1, \dots \mu_K \rangle$ and variances $\langle \sigma_1^2, \dots, \mu_K^2 \rangle$. Initially, the p_i are unknown to the player. At each round, $t = 1, \dots, T$ the learner selects an arm a_i and receives a reward $\rho_i(t) \sim p_i(t)$. In a practical application, the learner could be a recommender system with the arms corresponding to product categories and the reward being a click or the act of purchase. In the simplest bandit formulation, the rewards of each arm are independent of each other so at each round t the learner only sees the reward associated with arm a_i tried at this round and no information is provided about any other arms. The learner has two, seemingly conflicting, goals: finding out which arm has the highest expected value, while trying to gain as many high rewards as possible. In case of a recommender system that would correspond to finding out which product categories (arms) are of most interest to the user, while trying to ensure that as many products are purchased by the user as possible (high reward). Bandit algorithms specify a strategy by which the learner should choose an arm a_i at each round t. The most popular performance measure for bandit algorithms is the total expected regret defined as the difference between the total reward accumulated so far and the total reward if the learner always pulled the arm with the highest payoff:

$$R_T = T\mu^\star - \sum_{t-1}^{T} \mu_i(t),$$

where $\mu^\star = \max_{i=1,\dots,K} \mu_i$ is the expected reward from the best arm and T is the total number of rounds or trials.

Another point often mentioned in connection with bandit algorithms is the time horizon, i.e. how long should the exploration phase last, which could be defined in terms of the number of trials or the actual time-span of exploration. In certain application, defining the time-horizon can have an effect on the performance of the algorithm, e.g. in the case of a recommender system there might differences in user behaviour in different days of the week so the exploration phase should cover at least a whole week to capture all the variations.

ϵ-greedy

The ϵ-greedy algorithm is a popular heuristic for handling the exploration/exploitation trade-off. It is widely used due to its simplicity in terms of implementation and because it can easily generalize to many sequential decision problems (Kuleshov and Precup, 2014), where an agent's choice of action now depends on the actions they will choose in the future, for example scheduling.

At each round t, ϵ-greedy selects the arm with the highest empirical mean with probability $1 - \epsilon$, and selects a random arm with probability ϵ. In other words, given initial empirical means $\hat{\mu}_1, \dots, \hat{\mu}_K$:

$$p_i(t+1) = \begin{cases} 1 - \epsilon + \epsilon/k & \text{if } i = \arg\max_{j=1,\dots,K} \hat{\mu}_j(t) \\ \epsilon/k & \text{otherwise} \end{cases}$$

The ϵ-greedy algorithm forms a basis of a number of algorithms discussed in this survey and is often used as a baseline in testing.

Upper Confidence Bound

The Upper Confidence Bound (UCB) (Agrawal, 1995) family of algorithms is a simpler and more elegant implementation of the idea of optimism in the face of uncertainty (Lai and Robbins, 1985). Unlike ϵ-greedy, many of the UCB algorithms have theoretical guaranties in terms of its performance. One of the simplest UCB algorithms, UCB1 (Auer *et al.*, 2002a), maintains the empirical means of each arm as well

as the number of times that each arm has been played $n_i(t)$. Initially, each arm is played once. At round t, the algorithm chooses the arm $a_i(t)$ to play as follows:

$$a_i(t) = \arg \max_{i=1,\ldots,K} \left(\hat{\mu}_i + \sqrt{\frac{2 \ln t}{n_i}} \right)$$

At round t, the expected regret of UCB1 is bounded by:

$$8 \sum_{i:\mu_i<\mu^\star} \frac{\ln t}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right) \sum_{i}^{k} \Delta_i,$$

where $\Delta_i = \mu^\star - \mu_i$.

Auer *et al.* (2002a) also propose UCB1-Tuned, which performs better in practice than UCB1 but comes without theoretical guarantees. The main feature of UCB1-Tuned is that it takes into account the variance of each arm and not only its empirical mean. At round t, the algorithm picks arm a_i as follows:

$$a_i(t) = \arg \max_{i=1,\ldots,K} \left(\hat{\mu}_i + \sqrt{\frac{\ln t}{n_i} \min\left(\frac{1}{4}, V_i(n_i)\right)} \right),$$

where $V_i(t) = \hat{\sigma}_i^2(t) + \frac{2 \ln t}{n_i(t)}$. The estimate of the variance $\hat{\sigma}_i^2(t)$ can be computed by maintaining the empirical sum of squares of the reward, in addition to the empirical mean.

Due to the fact that in both algorithms each arm has to be played multiple times in order to calculate its empirical mean and variance, simple upper confidence bound strategies do not scale up to problems that require a large number of arms. However, the UCB1 algorithm is often used as a baseline in many experiments mentioned throughout this survey.

Linear and Contextual Bandits

Contextual bandits (Zhou, 2015) are an extension of the multi-arm bandit problem, where at each round the player has access not only to a bandit arm but also to a context associated with this iteration. The player's aim is to collect enough information about how the context

vectors and rewards of the arms relate to each other, so that the next best arm to play can be predicted by considering the feature vectors. For example, in a news personalization system, each news articles could be treated as an arm, and the features of both articles and users as contexts. The system then selects articles for each user to maximize click-through rate or dwell time. Contextual bandits play an important role in recommender systems. In Chapter 6, we will look in more detail how contextual bandits have been applied to improve personalization or the cold-start problem.

Many contextual bandits assume a linear dependency between the expected reward of an arm and its context (linear bandits). LinRel (Auer, 2002) was one of the first methods to extend the UCB algorithm to contextual cases. LinRel assumes that for each arm a_i there is an associated feature vector x_i and the expected reward of arm a_i is linear with respect to its feature vector. The algorithm maintains an implicit or explicit representation of the estimate $\hat{w}$ of the unknown weight vector w which maps context features to relevance scores. In each round t, LinRel algorithm obtains an estimate $\hat{w}_t$ by solving the linear regression problem $y_t \approx X_t \cdot \hat{w}_t$, where $y_t = (y_1, \ldots, y_{t-1})$ is the column vector of relevance scores, or rewards, received so far, and $X_t = (x_1, \ldots, x_{t-1})$ is the matrix of row feature vectors of datapoints (arms) tried up to time t. Based on the estimated weight vector $\hat{w}$, LinRel calculates an estimated relevance score $\hat{y}_i = x_i \cdot \hat{w}$ for each arm a_i that has not yet been tried by the player. The arm with the largest upper confidence bound for the relevance score is presented to the player. The upper confidence bound for an arm a_i is calculated as $\hat{y}_i + \alpha\hat{\sigma}_i$, where $\hat{\sigma}_i$ is an upper bound on the standard deviation of the relevance estimate $\hat{y}_i$. The constant α is used to adjust the confidence level of the upper confidence bound. In a regularized version of the algorithm, in each round t for each arm a_i, LinRel calculates:

$$c_i = x_i \cdot (X_t^T X_t + \gamma I)^{-1} X_t^T,$$

where γ is the reguralization parameter and I is the identity matrix. The arm that maximizes $c_i \cdot y_t + \frac{\alpha}{2} \parallel c_i \parallel$ for some specified constant $\alpha > 0$.

LinUCB (Li *et al.*, 2010a) is another contextual linear bandit similar to LinRel that has been used extensively in recommender system and personalization. We describe it in more detail in Section 6.1.

Gaussian Process UCB (GP-UCB) (Dorard *et al.*, 2009; Srinivas *et al.*, 2010) is another contextual bandit algorithm inspired by the UCB algorithm popular in IR and recommender systems. The Gaussian Process can be viewed as a prior over a regression function f. GP-UCB is a Bayesian approach to infer the unknown reward function f. We describe it in more detail in Section 6.1.

2.2.1 Thompson Sampling

Thompson Sampling (Thompson, 1933) is one of the oldest heuristics for solving the exploration/exploitation trade-off in a multi-armed bandit setting. It is a randomized algorithm based on Bayesian principles. Its theoretical foundations are not as strong as that of UCB algorithms, however, its empirical performance tends to be significantly better compared to many state-of-the art bandit methods – hence its frequent use in many practical applications, in particular in the area of recommender systems.

The main idea behind Thompson sampling is to randomly draw each arm according to its probability of being optimal. In a Bayesian setting, which is frequently used in recommendation and information retrieval, Thompson sampling can be explained as follows (Chapelle and Li, 2011). The set of past observations D is made of triplets (x_i, a_i, ρ_i), where x is a representation of a context, a is action taken (arm pulled) and ρ is the reward. D is modeled using a parametric likelihood function $P(\rho \mid a, x, \theta)$ depending on some parameters θ. Given some prior distribution $P(\theta)$, the posterior distribution of these parameters is given by the Bayes rule $P(\theta \mid D) \propto \Pi P(\rho_i \mid a_i, x_i, \theta) P(\theta)$. If the aim is to maximize the immediate reward (exploitation), then the action that maximizes the expected reward $\mathbb{E}(\rho \mid a, x)$ should be chosen. But in an exploration/exploitation setting, the probability matching heuristic consists in randomly selecting an action a according to its probability of being optimal. In the standard K-armed Bernoulli bandit, each action corresponds to the choice of an arm a and the reward follows

a Bernoulli distribution with mean $\theta_i^\star$. It is standard to model the mean reward of each arm using a Beta distribution. The instantiation of Thompson sampling for the Bernoulli bandit is given in Algorithm 1. It is easy to adapt the algorithm when different arms use different prior Beta distributions.

Algorithm 1 Thompson sampling for the Bernoulli bandit

Require: α, β prior parameters of a Beta distribution
$S_i = 0, F_i = 0, \forall_i$ //Success and failure counters
for all $t = 1, \ldots, T$ **do**
  for all $i = 1, \ldots, K$ **do**
    Draw θ_i according to $Beta(S_i + \alpha, F_i + \beta)$
  end for
  Draw arm $i = \arg\max_i \theta_i$ and observe reward ρ
  if $\rho = 1$ **then**
    $S_i = S_i + 1$
  else
    $F_i = F_i + 1$
  end if
end for

3

Click Models and Bandit Algorithms

Click models learn from user clicks to help understand and incorporate users' implicit feedback (Chuklin *et al.*, 2015). Previous studies of user click behaviour provide a spectrum of hypotheses and models on how an average user examines and clicks documents returned by a search engine with respect to the submitted query. This chapter provides a brief overview of bandit algorithms inspired by click models, most notably the *Cascade Model* (Craswell *et al.*, 2008), the *Dependent Click Model* (DCM) (Guo *et al.*, 2009) and the *Position Based Model* (PBM) (Richardson *et al.*, 2007).

The chapter is divided into three sections. Section 3.1 is devoted to the Cascade Model, Section 3.2 to the DCM, and Section 3.3 to the PBM. In each section, we first introduce the basic tenets behind each click model and then briefly describe the algorithms incorporating ideas from each model in a chronological order.

3.1 Cascade Model

A popular way to model user behaviour in web search is through the cascade model (Craswell *et al.*, 2008). The model makes two assumptions: (1) linear traversal through the ranking; and (2) that items below the

clicked result are not examined. The user views search results from top to bottom, deciding whether to click each result before moving to the next. The documents before the first click result are assumed to be not attractive because the user examines them without clicking on any of them. The documents after the first clicked result are unobserved because the user never examines them.

Formally, the model can be described as follows. Let us assume that there is a list K of items A that come from a ground set $E = 1, \ldots, L$, such as a list of documents. The user traverses a list of K items $A = (a_1, \ldots, a_K) \in \Pi_K(E)$, where $\Pi_K(E)$ is the set of K permutations of set E. The model is parametrized by attraction probabilities $w \in [0, 1]^E$. After the user examines item a_K, the item attracts the user with probability $w(a_k)$, independently of other items. If the user is attracted by item a_k, she clicks on it and stops the search. Thus, the probability that item a_k is examined by the user is $\Pi_{i=1}^{k-1}(1 - w(a_i))$, while the probability that the user finds at least one item attractive is $1 - \Pi_{i=1}^{K}(1 - w(a_i))$. This objective is maximized by K most attractive items.

3.1.1 Cascading Bandits

Kveton *et al.* (2015a) develop an online learning version of the cascade model called *Generalized Cascading Bandit*. In this model, the learning agent aims to learn the attraction probabilities of items. The goal of the agent is to maximize its total reward.

In theoretical terms, the model is formulated as a stochastic combinatorial partial monitoring problem (Kveton *et al.*, 2015c) represented by a tuple $B = (E, P, K)$, where $E = 1, \ldots, L$ is a ground set of L items, P is a probability distribution, and $K \leq L$ is the number of recommended items. Weights w are drawn from the probability distribution P and $w_t(e)$ is the preference of the user for item e at time t. If the user clicks on an item, then $w_t(e) = 1$.

The interaction between the learning agent and the user can be described as follows. At time t, the agent recommends a list of K items $A_t = (a_1^t, \ldots, a_K^t)$ to the user. The user examines the list, from the first

item a_1^t to the last a_K^t, and clicks on the first attractive item. If the user is not attracted by any item, the user does not click on any item.

The agent at time t receives feedback:

$$C_t = \arg_{min}\, 1 \leq k \leq K : w_t(a_k^t) = 1,$$

which is the click of the user. Since the user clicks on the first attractive item in the list, the observed weights of all recommended items at time t can be determined from C_t. The reward of the agent at time t can be written in several forms. For example, as $\max_k w_t(a_k^t)$ at least one item in list A_t is attractive; or as $f(A_t, w_t)$, where:

$$f(A, w) = 1 - \Pi_{k=1}^{K}(1 - w(a_k)).$$

The weights of the items in the ground set E are distributed independently and the weight of any item at time t is drawn independently of the weights of the other items.

Kveton *et al.* (2015a) propose two algorithms to solve the learning variant of the cascade model: CascadeUCB1 and CascadeKL-UCB. CascadeUCB1 is motivated by UCB1 (Auer *et al.*, 2002a) (Section 2.2) and CascadeKL-UCB is motivated by KL-UCB (Garivier and Cappé, 2011). KL-UCB is an online, horizon-free index policy for stochastic bandit problems shown to be more efficient and stable compared to various UCB policies. The pseudocode for both algorithms is in Algorithm 2. The algorithms differ only in how they estimate the upper confidence bound (UCB) $U_t(e)$ on the attraction probability of item e at time t. After that, they recommend a list of K items with largest UCBs:

$$A_t = \arg\max f(A, U_t).$$

After the user provides feedback C_t, the algorithms update their estimates of the attraction probabilities $w(e)$ for all $e = a_k^t$.

The UCBs are computed as follows. UCB on the attraction probability of item e at time t is:

$$U_t(e) = \hat{w}_{T_{t-1}(e)}(e) + c_{t-1,T_{t-1}(e)},$$

where $w_s(e)$ is the average of s observed weights of item e, and $T_t(e)$ is the number of times that item e is observed in t steps, and:

$$c_{t,s} = \sqrt{(1.5 \log t)/s}$$

is the radius of a confidence interval around $w_s(e)$ after t steps. In CascadeKL-UCB, the UCB is:

$$U_t(e) = \max q \in [w_{T_{t-1}(e)}(e), 1] :$$
$$T_{t-1}(e) D_{KL}(w_{T_{t-1}(e)}(e) \| q) \leq \log t + 3 \log \log t,$$

where $D_{KL}(p\|q)$ is the Kullback–Leibler (KL) divergence between two Bernoulli random variables with means p and q.

Algorithm 2 Cascading Bandits Algorithm

```
//Initialization
Observe w_0 ~ P
∀_e ∈ E : T_0 ← 1
∀_e ∈ E : w_1(e) ← w_0(e)
for all t = 1, ..., n do
  Compute UCBs U_t(e)
  // Recommend a list of K items and get feedback
  Let a_1^t, ..., a_K^t be K items with largest UCBs
  A_t ← (a_1^t, ..., a_K^t)
  Observe click C_t ∈ 1, ..., K
  //Update statistics
  ∀_e ∈ E : T_t(e) ← T_{t-1}(e)
  for all k = 1, ..., min{C_t, K} do
    e ← a_k^t
    T_t(e) ← T_t(e) + 1
    ŵ_{T_{t(e)}}(e) ← (T_{t-1}(e) ŵ_{T_{t-1(e)}}(e) + 1{C_t = k}) / T_t(e)
  end for
end for
```

The upper bounds of both algorithms are $O(\log t)$, linear in the number of items L, and they improve as the number of recommended items K increases. The bounds do not depend on the order of recommended items. Experimental results based on simulations show that CascadeKL-UCB outperforms CascadeUCB1 (Kveton *et al.*, 2015a), which is in concordance with the general observation that KL-UCB tends to outperform UCB1 when the expected payoffs of arms are low (Garivier and Cappé, 2011).

3.1.2 Combinatorial Cascading Bandits

Combinatorial Cascading Bandits (Kveton *et al.*, 2015b) generalizes cascading bandits to arbitrary combinatorial constraints, which allows the algorithm to learn what items might be attractive to the user starting from any list of K items out of L. At each step the agent chooses a set of items and receives a reward of one if and only if the weights w of all chosen items are one. The weights of the items are binary, stochastic, and drawn independently of each other. The agent only observes the index of the first chosen item with weight zero. In the combinatorial cascading bandits framework, there is no strong assumption that the items in list K are ordered according to their degree of relevance or that the user only clicks on one item in the list. In information retrieval systems or recommenders, this approach can facilitate the selection of a list of K items that reduces the probability that the user will not find any of the recommended items attractive.

The combinatorial cascading bandit problem can be solved through the CombCascade algorithm (Kveton *et al.*, 2015b) – an algorithm that also belongs to the UCB family. At time t, CombCascade operates in three stages. First, it computes the UCBs U_t on the expected weights of all items in E:

$$U_t(e) = \min\{\hat{w}_{T_{t-1}(e)}(e) + c_{t-1,T_{t-1}(e)}, 1\},$$

where $\hat{w}_s(e)$ is the average of s observed weights of item e, $T_t(e)$ is the number of times that item e has been observed in t steps, and $c_{t,s} = \sqrt{(1.5 \log t)/s}$ is the radius of a confidence interval around $\hat{w}_s(e)$. Next, CombCascade chooses the optimal solution with respect to these UCBs:

$$A_t = \arg\max f(A, U_t).$$

Finally, CombCascade observes the index of the first item in A_t with weight zero O_t and updates its estimates of the expected weights based on the weights of the observed items:

$$w_t(a_k^t) = \mathbb{1}\{k < O_t\} \qquad k = 1, \ldots, \min\{O_t, \mid A_t \mid\}$$

for all items a_k^t such that $k \neq O_t$. CombCascade is computationally efficient and its regret is polynomial.

3.1.3 Linear Cascading Bandits

All the algorithms discussed so far have one main disadvantage – they are not practical for problems when L is large. This is due to the assumption of independence of each item in E, which means that the user would have to see every item in the dataset at least once before the system could start behaving intelligently. To address this issue, Zong *et al.* (2016) propose linear cascading bandits, an online learning framework that makes the assumption that the attraction probabilities of items are a linear function of the features of items[1].

The key assumption is that the attraction probability of each item $e, w(e)$, can be approximated by a linear combination of some known d-dimensional feature vector $x_e \in \mathbb{R}^{d\times 1}$ and an unknown d-dimensional parameter vector of $\theta^\star \in \mathbb{R}^{d\times 1}$, which is shared among all items. The assumption is that there exists $\theta^\star$ such that $w(e) \approx x_e^T \theta^\star$ for any $e \in E$.

Two learning algorithms were proposed to solve this problem (Zong *et al.*, 2016): *cascading linear Thompson sampling* (CascadeLinTS) and *cascading linear UCB* (CascadeLinUCB). When the above linear generalization is perfect, their regret is independent of L and sublinear in t, which makes them suitable for learning to recommend from large ground sets E. The CascadeLinTS algorithm is based on Thompson Sampling (Agrawal and Goyal, 2012; Thompson, 1933) (Section 2.2.1), while the CascadeLinUCB algorithm is based on linear UCB (Wen *et al.*, 2015).

Both algorithms represent their past observations as a positive-definite matrix $M_t \in \mathbb{R}^{d\times d}$ and a vector $B_t \in \mathbb{R}^{d\times 1}$. Specifically, let X_t be a matrix whose rows are the feature vectors of all observed items in t steps and Y_t be a column vector of all observed attraction weights in t steps. Then:

$$M_t = \sigma^{-2} X_t^T X_t + I_d$$

is the gram matrix in t steps and:

$$B_t = X_t^T Y_t,$$

where I_d is a $d \times d$ identity matrix and $\sigma > \theta$ is a parameter that controls the learning rate.

[1] See Section 6.1 for more information about linear bandits.

The algorithms operate in three stages. First, they estimated the expected weight of each item e based on their model of the world. CascadeLinTS randomly samples parameter vector θ_t from a normal distribution and then estimates the expected weight as $x_e^T \theta_t$. CascadeLinUCB computes an upper confidence bound $U_t(e)$ for each item e:

$$U_t(e) = \min\{x_e^T \theta_t + c\sqrt{x_e^T M_{t-1}^{-1} x_e}, 1\}.$$

Second, both algorithms choose the optimal list A_t. Finally, they receive feedback, and update M_t and B_t:

$$M_t = M_t + \sigma^{-2} x_e x_e^T$$
$$B_t = B_t + x_e \mathbb{1}\{C_t = k\}$$

CascadeLinTS was evaluated on several recommendation problems for various dataset sizes and number of features: restaurant recommendation, song recommendation and movie recommendation. In general, for small datasets the regret of CascadeLinTS and CascadeUCB1 is similar. However, as the size of the dataset increases, the regret of CascadeUCB1 is orders of magnitude larger compared to CascadeLinTS. Additionally, CascadeLinTS performs well irrespective of the number of features.

3.1.4 Contextual Cascading Bandits

Another approach to incorporating context into the cascading bandits model was proposed by Li *et al.* (2016c). Contextual information includes various user and item information (Section 6.1). Li *et al.* also introduce position discounts to the cascading bandit. Cascading bandits (Kveton *et al.*, 2015a) treat all positions in the list of K items suggested to the user equally, however, in practical applications different positions may result in different rewards. For example, in online recommendation, it is more satisfactory for the user to find their item of interest as early as possible in the list. To model these preferences, position discounts are applied whereby the items in the recommended set are presented in the decreasing order of UCBs.

The user is presented with K items. Each item a is represented by a feature vector x. Each feature vector x_a combines the information of

the user and the corresponding base arm a. For example, if the the user at time t is characterized by a feature vector u_t and the base arm a has a feature vector g_a, then $x_{t,a} = u_t g_a^T$ is the combined feature vector of base arm a at time t. The learning agent also knows of the user's past history $\mathcal{H}$, which contains feedback information at all time $s < t$, as well as contextual information at time t.

Additionally, the reward received by the agent is subject to position discounts $\gamma_k \in [0, 1], k \leq K$. For example, if the user selects the first item on the recommended list, the learning agent will receive reward 1 and if the user selects an item further down the list, the learning agent will receive a discounted reward γ_k.

A solution to this problem is the C^3-UCB algorithm. After computing the UCBs based on which a list of items is selected for presentation to the user, the algorithm computes an estimate of $\hat{\theta}_t$, which can be viewed as a ridge regression problem:

$$\hat{\theta}_t = (X_t^T X_t + \lambda I)^{-1} X_t^T Y_t,$$

where X is a matrix with each row being the feature vector of items presented to the user up to time t, i.e. $\gamma_k x_{s,a_k}$, and Y_t is a column vector whose elements are the weights of the corresponding feature vectors in X, i.e. $\gamma_k w_s(a_k)$, with regularization parameter $\lambda > 0$. A new confidence radius $\beta_t(\sigma)$ is also calculated:

$$\beta_t(\sigma) = R\sqrt{\ln\left(\frac{\det(V_t)}{\alpha^d \sigma^2}\right)} + \sqrt{\lambda},$$

where $V_t = X_t^T X_t + \lambda I$ and R is the current regret. Finally, the UCB is defined as:

$$U_t(a) = \min\{\hat{\theta}_{t-1}^T x_{t,a} + \beta_{t-1}(\sigma) \parallel x_{t,a} \parallel_{V_t^{-1}-1}, 1\}$$

Experimental results on movie recommendation using the *MovieLens* dataset show that in comparison to CombCascade (Kveton *et al.*, 2015b), C^3-UCB's rewards are on average 3.5 times higher, which indicates that incorporating more contextual information into the algorithm can greatly improve the quality of the recommended items.

In the Cascade Model, the positions of the items in the recommended list are not taken into account in the reward process. Indeed, some of

the experimental results presented in Kveton *et al.* (2015a) point to the counter-intuitive conclusion that, from the user's perspective, the optimal strategy from the perspective of the learning agent might be to show the most relevant items at the end of the list in order to maximize the amount of observed feedback.

To overcome these limitations, Combes *et al.* (2015) introduced weights attributed to positions in the list, with a click on position $k \in \{1, \ldots, K\}$ providing a reward w_k with decreasing sequence $(w_k)k$ to enforce the ranking behaviour. However, no rule is given for setting the weights $(w_k)k$ that control the order of importance of the positions. The authors propose an algorithm based on KL-UCB (Garivier and Cappé, 2011) called *Parsimonious Item Exploration* PIE(l)[2]. At time t, a list of items $A_t = (a_1^t, \ldots, a_K^t)$ is presented to the user. The list is composed of K items with the highest empirical means sorted in decreasing order – these are the "leaders" at time t. The set $\mathcal{B}(t)$ contains items which are not in the set of leaders and whose indices are larger than the empirical mean of item a_K^t. The set $\mathcal{B}(t)$ may potentially contain items that are better than the worst current leader. Additionally, we create a list $U_i^k(t)$ obtained by considering the $K-1$ first items of the set A_t and placing item i at position k. The PIE(l) algorithm proceeds as follows. If $\mathcal{B}(t)$ is empty, then the leader is selected as $u_t = A_t$. Otherwise, $u_t = A_t$ is selected with probability $\frac{1}{2}$ and $u_t = U_{i(t)}^k(t)$ with probability $\frac{1}{2}$, where $i(t)$ is chosen from $\mathcal{B}(t)$ uniformly at random. The goal of the algorithm is to ensure that the items on the list presented to the user are relevant but also maintain a level of diversity at the same time. The PIE(l) algorithm has a low complexity and at each round $O(T + L\log(T))$ operations. Experimental results using the MovieLens dataset show that PIE(l) significantly outperforms KL-UCB.

3.2 Dependent Click Model

The cascade model (Section 3.1) assumes that a user abandons examination of the ranked list of items after the first click. However, in many

[2]Note that l in PIE(l) indicates the position in the list of presented K items, thus, the name of the algorithm should be PIE(k) in order to be consistent with the notation used in this chapter. However, the original name of the algorithm is PIE(l).

real-world applications multiple clicks are possible, especially for informational queries which tend to have a relatively large average number of clicks per query session (Section 4.2). Guo *et al.* (2009) propose the *dependent click model* (DCM) which generalizes the cascade model to multiple clicks by including a set of position-dependent parameters to model probabilities that the user returns to the search result page and resumes the examination after a click.

As in the cascade model (Section 3.1.1), the DCM model assumes that the user scans a list of K items $A = (a_1, \ldots, a_K)$ from the first item a_1 to the last a_K. The model is parameterized by item-dependent attraction probabilities $w \in [0,1]^E$ and position-dependent termination probabilities $v \in [0,1]^K$. The user interacts in this model as follows. After the user examines item a_k, the item attracts the user with probability $w(a_k)$, independently of the other items. If the user is attracted by item a_k, the user clicks on it and terminates the search with probability $v(k)$. In this case, it is assumed that the user is satisfied with item a_k and does not examine the remaining items. If the user is not attracted by item a_k, or the user is attracted but does not terminate, the user examines the next item a_{k+1}. The probabilities $w(a_k)$ and $v(k)$ are conditional on that the user examines the item, and that the examined item is attractive. Thus, the probability that the user leaves satisfied given list A is $1 - \Pi_{k=1}^{K}(1 - v(k)w(a_k))$. This objective is maximized by K most attractive items, where the k^{th} most attractive item is placed at the position with the k^{th} highest termination probability.

Katariya *et al.* (2016b) propose a learning variant of the DCM model called DCM bandits. At time t, the learning agent recommends a list of K items. After the user examines the list, the learning agent receives a vector of observations $c_t \in \{0,1\}^K$ indicating the clicks of the user. In particular, $c_t(k) = 1$ if the user clicks on item a_k^t at position k at time t. The learning agent receives a reward r_t, which is unobserved. The reward is binary and is one if the user is satisfied with at least one item in the presented list, i.e. an item is attractive to the user, $w_t(e) = 1$, and its position leads to termination, $v_t(k) = 1$.

Learning in DCM bandits seems difficult because the observations c_t are insufficient to identify whether the recommended items lead to a reward. For example, the learning agent recommends items $A_t =$

$(1, 2, 3, 4)$ and observes user clicks $c_t = (0, 1, 1, 0)$. This feedback can be interpreted in two ways. The first explanation is that item 1 is not attractive, items 2 and 3 are attractive, and that the user does not exit at either positions 2 or 3. The second explanation is that item 1 is not attractive, items 2 and 3 are attractive, and that the user does not exit at position 2, but exits at position 3. In the first case, the learning agent receives no reward; in the second one, it does. Since the reward is not directly observed, DCM bandits can be viewed as an instance of partial monitoring. However, DCM bandits cannot be solved efficiently by existing algorithms for partial monitoring because the action set is combinatorial. In DCM bandits, an additional mild assumption is imposed to allow efficient learning: the order of the termination probabilities is known to the learning agent.

The proposed solution to DCM bandits is an algorithm called dcmKL-UCB, which, similarly to CascadeKL-UCB (Section 3.1.1), is motivated by KL-UCB (Garivier and Cappé, 2011). First, the algorithm computes the UCBs U_t on the attraction probabilities of all items in E at time t:

$$U_t(e) = \max\{q \in [w, 1] : w = \hat{w}_{T_{t-1}(e)}(e), \\ T_{t-1}(e) D_{KL}(w \parallel q) \leq \log t + 3 \log \log t\},$$

where $D_{KL}(p \parallel q)$ is the Kullback–Leibler (KL) divergence between Bernoulli random variables with means p and q; $\hat{w}_s(e)$ is the average of s observed weights of item e; and $T_t(e)$ is the number of times that item e is observed in t steps. Second, dcmKL-UCB recommends a list of K items with largest UCBs:

$$A_t = \arg \max_{A \in \Pi_K(E)} f(A, U_t, v),$$

After the user provides feedback c_t, dcmKL-UCB updates its estimates of $\hat{w}(e)$ up to position $\min\{C_t^{last}, K\}$, where C_t^{last} is the position of the last click. The estimates of weights is:

$$\hat{w}_{T_t(e)} = \frac{T_{t-1}(e) \hat{w}_{T_{t-1}(e)}(e) + c_t(k)}{T_t(e)}$$

The algorithm dcmKL-UCB was evaluated on the Yandex dataset, an anonymized search log of 35M million search sessions, and compared to

KL-UCB (Garivier and Cappé, 2011). The regret of dcmKL-UCB is at least half than that of the baseline.

3.3 Position-Based Model

The Cascade Model (Section 3.1) and the DCM (Section 3.2) models assume that a portion of the recommended list is explicitly examined by the user, which allows the learning agent to access the rewards corresponding to the unbiased user's evaluation of each item. In the Position-Based Model (PBM) (Richardson *et al.*, 2007), each position in the list is also endowed with a binary *examination variable* which is equal to one only when the user paid attention to the corresponding item. However, this variable, which is independent of the user's evaluation of the item, is not observable. Compared to the Cascade model, the PBM is challenging due to the fact that the learning agent only observes actual clicks, however, non-clicks may also provide additional information but this information tends to be ambiguous. Thus, combining observations made at different positions becomes a non-trivial statistical task.

3.3.1 Contextual Information and PBM Bandits

Lagrée *et al.* (2016) propose a model that exploits information about the display position bias. Their model is characterized by examination parameters $(\kappa_k)_{1\leq l\leq K}$, where κ_k is the probability that the user effectively observes the item in position k. At time t, a list of items $A(t)$ is shown to the user and the learning agent observes the user feedback but the observation $Z_k(t)$ at position k is censored being the product of two independent Bernoulli variables $Y_k(t)$ and $C_k(t)$, where $Y_k(t) \sim \mathcal{B}(\kappa_k)$ is non null when the user considered the item in position k, which is unknown to the learning agent, and $C_k(t) \sim \mathcal{B}(\theta_{A_k(t)})$ represents the actual user feedback to the item shown in position k, where θ is the arm expectation. The learning agent receives a reward $r_{A(t)} = \Sigma_{k=1}^{K} Z_k(t)$. It is assumed that the examination parameters (κ_k) are known to the learning agent – they can be estimated, e.g. from historical data.

The authors introduce two algorithms for the PBM problem. The PBM-UCB algorithm requires an "exploration bonus" based on UCB

derived from Hoeffding's inequality:

$$U_e(t,\delta) = \frac{S_e(t)}{\tilde{N}_e(t)} + \sqrt{\frac{N_e(t)}{\tilde{N}_e(t)}}\sqrt{\frac{\delta}{2\tilde{N}_e(t)}},$$

where $S_e(t) = \Sigma_{l=1}^{K} S_{e,k}(t)$, $S_{e,k}(t) = \Sigma_{s=1}^{t-1} Z_k(s)\mathbb{1}\{A_t(s) = e\}$, $N_e(t) = \Sigma_{l=1}^{K} N_{e,k}(t)$, $N_{e,k}(t) = \Sigma_{S=1}^{t-1} Z_k(s)\mathbb{1}\{A_t(s) = e\}$. The UCB also incorporates bias-corrected versions of the counts: $\tilde{N}_e(t) = \Sigma_{l=1}^{K} \tilde{N}_{e,k}(t)$, $\tilde{N}_{e,k}(t) = \Sigma_{S=1}^{t-1} Z_k(s)\kappa_k\mathbb{1}\{A_t(s) = e\}$. The algorithm sorts optimistic indices in decreasing order and pulls the corresponding first K arms.

The PBM-PIE algorithm is an adaptation of the PIE(l) algorithm (Combes *et al.*, 2015) introduced for the Cascade Model (Section 3.1.4). At each round, the learning agent potentially explores at position K with probability $\frac{1}{2}$ using the UCB for each arm e, which also incorporates the examination parameters (κ_k). In other positions, $k = 1, \ldots, K-1$, PBM-PIE selects the arms with the largest estimates $\hat{\theta}_e(t) = S_e(t)/\tilde{N}_e(t)$. Simulation experiments show that PBM-PIE outperforms PBM-UCB and the baseline KL-UCB, which in turn outperforms PBM-UCB.

The drawback of the approach proposed by Lagrée *et al.* (2016) is the fact that the examination parameters are estimated off-line, which limits the usage of these algorithms. Komiyama *et al.* (2017) expands PBM by making an additional assumption about the examination parameters $\kappa(k)$, namely it is natural that items that are high up the list receive more clicks. Thus, $\kappa_1 > \kappa_2 > \ldots > \kappa_K > 0$ and this order is known. Additionally, $\kappa_1 = 1$. This model involves $E + K$ parameters $\{\theta_i^\star\}_{i\in[E]}$ and $\{\kappa_k^\star\}_{k\in[K]}$, where K is the number of arms and K is the number of slots in a list of items presented to the user. Each arm $e \in [E]$ is associated with a distinct parameter $\theta_i^\star \in (0,1)$, and each slot $k \in [K]$ is associated with a parameter $\kappa_k^\star \in (0,1]$. At each round t, the system selects k items (arms) for presentation and receives a corresponding binary reward (click or non-click) for each item. The reward is drawn from a Bernoulli distribution. The parameters except for $\kappa_1 = 1$ are unknown to the learning agent. The goal for the learning agent is to maximize the cumulative rewards.

The solution to this problem is provided by *Permutation Minimum Empirical Divergence* (PMED) algorithm. The algorithms contains two loops: $L_C = L_C(t)$ is the set of K-allocations in the current loop, and

$L_N = L_N(t)$ is the set of K-allocations that are to be drawn in the next loop. In the L_C a set of permutations of $[E] \times [K]$ of each pair of arm and slot is produced. Loop L_C provides a uniform exploration over all pairs of arms and slots. For every $m \in [E]$, let v_m^{mod} be an K-allocation $(1 + \mod E(m), 1 + \mod E(1+m), \ldots, 1 + \mod E(K+m-1))$, where $\mod_E(x)$ is the minimum non-negative integer among $\{x - cE : c \in \mathbb{N}\}$. If some pair (e, k) of arm and slot is not allocated $\alpha\sqrt{\log t}$ times, a corresponding K-allocation is put into the L_N loop. If no pair is put to L_N and L_C is empty, then the algorithm puts top K arms with the largest $\{\hat{\theta}_e(t)\}$ into L_N. By introducing this bipartite matching algorithm, the number of possible candidate arms is reduced, which allows the algorithm to run in polynomial time. Experimental results on synthetic and real data show that in comparison to PBM-PIE and dcmKL-UCB, (Section 3.2) the gap between PMED and existing algorithms is not substantial, however, the existing algorithms suffer larger regret than PMED.

3.3.2 Stochastic Rank-1 Bandits

Katariya *et al.* (2016a) and Katariya *et al.* (2017) propose stochastic rank-1 bandits, a class of online learning problems where at each step a learning agent chooses a pair of row and column arms, and receives the product of their values as a reward. This problem can be directly applied to PBM, where the item in the list is clicked only if it is attractive and its position is examined. Under these assumptions, the pair of the item and position that maximizes the probability of clicking is the maximum entry of a rank-1 matrix, which is the outer product of the attraction probabilities of items and the examination probabilities of positions. The problem is defined by a tuple $\{K, L, P_u, P_v\}$, where K is the number of rows, L is the number of columns, P_u is a probability distribution over a unit hypercube $[0, 1]^K$, and P_v is a probability distribution over a unit hypercube $[0, 1]^L$. At time t, the learning agent chooses arm $(i_t, j_t) \in [K] \times [L]$, and then observes $u_t(i_t)v_t(j_t)$, which is also its reward, where u_t and v_t are probabilities vectors drawn from P_u and P_v. The goal of the agent is to maximize its expected cumulative reward in t steps.

The stochastic rank-1 bandits problem can be solved through an elimination algorithm called Rank1Elim. The key idea in Rank1Elim is to explore all remaining rows and columns randomly to estimate their expected rewards and then eliminate those rows and columns that seem suboptimal.

In the algorithm, $h_l^U(i)$ is the index of the most rewarding row whose expected reward is believed by Rank1Elim to be at least as high as that of row i in stage l. Initially, $h_0^U(i) = i$. When row i is eliminated by row i_l in stage l, $h_{l+1}^U(i)$ is set to i_l; then when row i_l is eliminated by row $i_{l'}$ in stage $l^{'} > l$, $h_{l'+1}^U(i)$ is set to $i_{l'}$; and so on. The corresponding column quantity, $h_l^V(j)$ is defined and updated analogously.

At each stage of the algorithm has two main steps: exploration and elimination. In the row exploration step, each row $i \in I_l$ is explored randomly over all remaining columns J_l such that its expected reward up to stage l fulfills a pre-defined condition. To guarantee this, column $j \in [L]$ is sampled randomly and then substituted with column $h_l(j)$, which is at least as rewarding as column j. The observations are stored in reward matrix $C_l^U \in \mathbb{R}^{K \times L}$. The column exploration step is analogous. In the elimination step, the confidence intervals of all remaining rows $i \in I_l$ are estimated from matrix $C_l^U \in \mathbb{R}^{K \times L}$, and the confidence intervals of all remaining columns $j \in J_l$ are estimated from $C_l^V \in \mathbb{R}^{K \times L}$.

Rank1Elim's regret scales linearly with the number of rows and columns on instances where the minimum of the average row and column rewards μ is bounded away from zero. However, Rank1Elim fails to be competitive with straightforward bandit strategies, such as UCB1, as $\mu \to 0$. Consider, for example, a setting when $K = L$ and the row and column rewards are Bernoulli distributed. Let the mean reward of row 1 and column 1 be Δ, and the mean reward of all other rows and columns be 0. For this setting, $\mu = \Delta/K$, and consequently the regret of Rank1Elim is $\mathcal{O}(\mu - 2\Delta - 1K \log t) = \mathcal{O}(K3 \log t)$. However, a naive bandit algorithm that treats each row-column pair as unrelated arms has $\mathcal{O}(K2 \log t)$ regret. Katariya *et al.* (2017) propose that this drawback of Rank1Elim can be eliminated by replacing the UCB1 confidence intervals used by Rank1Elim by strictly tighter confidence intervals

based on KL divergences. The new algorithm is called Rank1ElimKL (Garivier and Cappé, 2011).

The performance of Rank1ElimKL, Rank1Elim and UCB1 was compared on models derived from the *Yandex* dataset. The experimental data averaged over 20 most frequent queries in the dataset shows that Rank1ElimKL's regret is 10.9% lower than that of UCB1, and 79% lower than that of Rank1Elim.

Stochastic click bandits (Zoghi *et al.*, 2017) is a learning framework for maximizing the expected number of clicks in a broad class of click models, including the Cascade Model and the Position Based Model. The approach is implemented through BatchRank algorithm. In order to avoid biases due to the click model, the placement of items is randomized. Next, the batches of items are recursively divided into more and less attractive items, which results in a sorted list of K items, where the k-th most attractive item is placed at position k.

The batches are explored as follows. In stage i of batch b, $len(b)$ least observed items remaining in batch b are randomly selected, where $len(b)$ is number of positions in batch b. The items are displayed at random positions. If the number of these items is less than $len(b)$, then they are mixed with randomly chosen more observed items. This exploration helps two avoid biases due to click models in two ways: (i) it is uniform and no item in a given batch is explored more than once than any other item in the same batch; (ii) any item in a given batch appears in any other list with that item with the same probability. After collecting user feedback (clicks), KL-UCB upper and lower confidence bounds (Garivier and Cappé, 2011) (Section 3.1.1) are computed for all the items in a given batch. Next, the algorithm tests whether the batch can be divided into two new batches. The values of lower confidence bound of each item in the batch are compared one by one to the value of the highest upper confidence bound in the batch. The first batch contains items all the items in the batch up to the position of the last item where the above condition is fulfilled. The second batch contains the remaining items and is over the remaining positions. The indices of the new batches are initialized to 0. If the batch cannot be divided, items that cannot be placed in position k or above based on the value of their upper confidence bound are eliminated.

Table 3.1: Summary of bandit algorithms based on click models

Model	Algorithm	Basis
Cascade	CascadeUCB1	UCB1
	CascadeKL-UCB	KL-UCB
	CombCascade	UCB1
	CascadeLinTS	Thompson Sampling
	CascadeLinUCB	UCB1
	C^3-UCB	UCB1
	PIE(l)	KL-UCB
DCM	dcmKL-UCB	KL-UCB
PBM	PBM-UCB	UCB1
	PBM-PIE	KL-UCB
	PMED	KL-UCB
	Rank1Elim	UCB1
	Rank1ElimKL	KL-UCB
Stochastic	BatchRank	KL-UCB

BatchRank was compared to CascadeKL-UCB (Kveton *et al.*, 2015a) (Section 3.1.1) and RankedExp3 (Radlinski *et al.*, 2008a) (Section 4.1) using the Yandex dataset. In the Cascade Model, CascadeKL-UCB outperforms BatchRank, while in the PBM, BatchRank outperforms CascadeKL-UCB in terms of regret overtime. BatchRank outperforms RankedExp3 in all experiments.

3.4 Summary

In this chapter, we provided a brief overview of bandit algorithms inspired by click models. The summary is provided in Table 3.1. In all the discussed cases, incorporating assumptions about item attractiveness and position biases derived from click models into a more basic bandit policies, such as UCB1 (Auer *et al.*, 2002a) (Section 2.2) and KL-UCB (Garivier and Cappé, 2011), leads to the creation of more robust bandit

algorithms which can be incorporated into information retrieval and recommender systems.

The UCB1 (Auer, 2002) and KL-UCB (Garivier and Cappé, 2011) form the basis of almost all the proposed algorithms. In general, KL-UCB based algorithm achieve lower regret compared to those based on simple UCB1 policies, which translates into more relevant items being suggested to the user earlier on in the search. The inferior performance of UCB1 based algorithms is largely due to the assumption independence of each arm, which means that the algorithm requires more feedback from the user, and it may fail to scale to large datasets, which might be problematic for real-life applications. The performance of both UCB1 and KL-UCB based algorithms is further improved by incorporating linearity, as in CascadeLinUCB (Section 3.1.3), or more contextual information, as in PBM-PIE (Section 3.3.1).

4

Ranking and Optimization

This chapter focuses on the application of bandit algorithms in three related areas of information retrieval: diversification of search results (Section 4.1), search engine optimization and offline bandit policy evaluation (Section 4.2), and query auto-completion and recommendation (Section 4.3). Section 4.1, in particular, introduces a number of the bandit approaches, most notably the *Ranked Bandit Algorithm* (Radlinski *et al.*, 2008a) and *Bandits in Metric Spaces* (Slivkins *et al.*, 2010), that gave rise to new lines of research both in information retrieval and in theoretical machine learning.

4.1 Diversifying Ranking with Bandits

Traditional approaches to ranking rely on training data, provided in the form of human judgments, to assess the relevance of documents to a query. The relevance score for each document is computed independently of other documents in the dataset and then the documents are ordered by decreasing score (Joachims, 2003). This often leads to rankings with redundant results which are too similar to one another. User studies, however, show that diversity at high ranks is often preferred (Teevan *et al.*, 2007).

Radlinski *et al.* (2008a) formulate an approach to document ranking that: (1) learns from users' click data rather than manually labeled relevance judgments; (2) in an online manner; (3) taking into consideration the dependency between documents. The goal is to increase diversity in the top k ranked documents and thus maximize the probability of the user finding a relevant document in the top k positions of a ranking.

Two algorithms are proposed. *Ranked Explore and Commit* (REC) is a greedy strategy that assumes that the user's interests as well as the document dataset remain constant over time. REC iteratively selects documents for each rank. At each rank position k, every document a_j is presented a fixed number of times and the number of clicks it receives is recorded. After n presentations, the document that received the highest number of clicks in a given rank is permanently assigned to that rank, which means that REC often performs badly if the user's interests change. The *Ranked Bandit Algorithm* (RBA)[1] avoids this problem by incorporating exploration to the process of assignments of documents to rank positions. RBA runs an instance of a multi-armed bandit (MAB) (Auer *et al.*, 2002a) (Section 2.2) for each rank. Each of the k copies of the MAB algorithm maintains a value for every document in the dataset. The algorithm MAB1 is responsible for selecting the document to show at rank 1. Next, the algorithm MAB2 determines which document is shown at rank 2, etc. If the same document was selected at a higher rank, then another document is picked randomly for the lower rank. This process is repeated until all the top k rank positions are filled. If the user clicks on one of the top k documents, the reward for the arm corresponding to that document for the MAB at this rank is set to 1, while the reward for the arms corresponding to all the other documents is set to 0. The bandits corresponding to each rank are independent of each other. The RBA approach gave rise to two algorithms: RankUCB1, based on UCB1 (Auer *et al.*, 2002a), and RankEXP3, based on the EXP3 algorithm (Auer *et al.*, 2002b). EXP3 is designed for the adversarial setting with no assumptions on how the clicks are generated, while UCB1 makes the assumption that

[1] A generalization of the Ranked Bandit Algorithm, in an abstract setting without a specific application to Information Retrieval, was discovered independently by Streeter and Golovin (2008) and Streeter *et al.* (2009).

the user's interests do not change. In small scale simulation experiments RankUCB1 performs much better than RankEXP3.

RBA assumes that all the documents are independent, which means that the algorithm does not scale up to large datasets. Taking document similarity and ranking context into account can increase the scalability of a ranking algorithm. Slivkins *et al.* (2010) and Slivkins *et al.* (2013) extend the ranked bandit approach to online learning to rank in metric spaces (Kleinberg *et al.*, 2008) by explicitly considering correlation of clicks and similarity between documents.

The similarity between documents is derived from a tree hierarchy, where closer pairs of documents are considered to be more similar. Each document $a \in A$ is a leaf in the tree. On this tree, the distance between any two nodes is exponential in the depth of their least common ancestor, with base $\epsilon \in (0, 1)$. An alternative notion of document similarity is correlation between clicks with the click probability $\mu(a)$ for document a if it appears in the top rank position. Each rank position $i > 1$ is examined by the user only if all the documents in higher positions are not clicked, so the click probabilities for this position are conditional on this event. Thus, there is a subset of documents $S \subset A$, where the assumption is that all documents in S are not relevant to the user – this event is indicate as Z_S.

In order to traverse the tree, the "zooming algorithm" (Kleinberg *et al.*, 2008) is used. It maintains a set of active subtrees which partition the leaf set. In each round, the active subtree with the maximal index is chosen, which is the best upper confidence bound on the click probabilities in this subtree. It is defined via the confidence radius:

$$rad = \sqrt{4 \log(T)/(1 + \sharp \text{samples})},$$

where T is the time horizon. The algorithm then zooms in on a given active subtree, de-activates it and activates all its children.

In order to improve the performance of the above algorithm, the context is taken into consideration as well. In each round, a context x is revealed, the algorithm chooses a document a, and the payoff is an independent $\{0, 1\}$ sample with expectation $\mu(a \mid x)$. Contexts are leaves in a context tree. Further, metrics $\mathcal{D}$ and $\mathcal{D}_x$ provide similarity information about documents and contexts, respectively. Thus, for any

two documents a, a' and any two contexts x, x':

$$| \mu(a \mid x) - \mu(a' \mid x') | \leq \mathcal{D}(a, a') + \mathcal{D}_x(x, x').$$

The algorithm maintains a set of active strategies in the form of (τ, τ_x), where τ is a subtree in the document tree and τ_x is a subtree in the context tree. At each round, the active strategies partition the space of all (document, context) pairs. In each round, a context x arrives, and one of the active strategies (τ, τ_x) with $con \in \tau_x$ is selected based on its maximal upper confidence bound. Next, a document $a \in \tau$ is picked uniformly at random.

In the case of ranked bandits in metric spaces, the context is provided by the set of documents S, i.e. documents that the user saw in higher rank positions but did not click. The MAB algorithm at each rank position knows the document set S of documents in the upper positions. For each round, the click probabilities for MAB in a given rank position are given by $\mu(\cdot \mid Z_S)$. The resulting ranked algorithm is called *RankContextZoom*.

The *RankContextZoom* algorithm was tested in simulations using a document collection of 32000 in a binary ϵ-exponential tree metric space with $\epsilon = 0.837$. The simulation was run over a 5-slot ranked bandit setting, learning the best 5 documents. In this setting, the performance of RankEXP3 and RankUCB1 is comparable to picking documents randomly, which is due to the large size of the dataset and slow convergence rates of these algorithms, while RankContextZoom improved the performance substantially.

The problem of diversified ranking has also been approached through linear submodular functions (Yue and Guestrin, 2011). In this setting, documents are represented by a set of topics or concepts. The goal is to recommend documents that do not cover highly overlapping topics. A set function F maps sets of recommended articles A to real values, e.g., the total information covered by A. For each topic i, there is a function $F_i(A)$ showing how well the recommended document set A covers topic i. The total utility of recommending A is:

$$F(A \mid w) = w^T F_1(A), \ldots, F_i(A),$$

where w is a weight parameter vector indicating the user's interest in each topic. Thus, $F(A \mid w)$ corresponds to the weighted information coverage of A, which varies from user to user. When making recommendations, the aim is to select a set of documents A that maximizes $F(A \mid w)$.

This approach is illustrated through a bandit algorithm called LSBGreedy. At iteration t, a set of documents A is presented to the user. Each document a is represented by a set of basis coverage functions $F_1, \ldots, F_i$. The documents are selected for presentation based on their coverage functions and the previous user feedback, i.e. clicks on a document, and the rewards for each recommended document are observed. At each iteration t, an estimate of w_t is calculated via linear regression based on the previous feedback:

$$w_t = M_t^{-1} C_t,$$

where M_t^{-1} is the inverse covariance matrix of the submodular features of the previously selected articles, and C_t is the user's feedback provided so far.

The upper confidence bound for each document a is calculated as:

$$\mu_t + \alpha \sqrt{\Delta(a \mid A_t) M_t^{-1} \Delta(a \mid A_t)},$$

where $\mu_t = w_t \Delta(a \mid A_t)$ is a mean estimate of utility gain from adding a new document a to the dataset A_t, and α is the parameter controlling exploration. The documents with the highest upper confidence bound are selected for presentation to the user.

The performance of LSBGreedy was compared in simulations to a number of bandit algorithms using the Yahoo! news dataset. In terms of average simulated user feedback, LSBGreedy significantly outperformed the ϵ-greedy algorithm (Sutton and Barto, 1998) (Section 2.2). LSBGreedy's performance was comparable to RankLinUCB – an algorithm obtained through the merger of RBA and LinUCB (Li *et al.*, 2010a) (Section 6.1.1), which indicates that combining bandits with submodular functions obtains similar results to contextual bandits.

Hofmann *et al.* (2011b) and Hofmann *et al.* (2013b) investigate mechanisms to balance exploration and exploitation in listwise and

pairwise learning to rank methods (Liu, 2009). The pairwise approach takes as input pairs of documents with labels specifying which one is preferred and then learns a classifier that predicts these labels. In pairwise learning, training can be performed with implicit feedback in batches. First, implicit feedback is collected using results of an initial ranking function. Next, the feedback is collected and the algorithm is trained. Thus trained system is then evaluated with users. However, in this approach the documents in the top positions come from the current best ranking function, which means they are highly likely to be similar to one another with little diversity. In order to remedy this problem, Hofmann *et al.* (2013b) first define two document lists: one exploratory and one exploitative. These lists are then combined to balance exploration and exploitation. The exploitative list is generated according to the scores from a trained ranking algorithm, while the exploratory list is generated by randomly sampling documents associated with a query.

The exploration and exploitation is balanced by an ϵ-greedy style algorithm (Sutton and Barto, 1998) the difference being that while in ϵ-greedy only a single action is selected in each round, in the online approach to pairwise learning multiple actions are selected and presented at the same time. The relative number of documents from the exploratory and exploitative lists varies depending on the value of $\epsilon \in [0, 1]$. For each rank, a document from the exploitative list is selected with probability $1 - \epsilon$ and a document from the exploratory list is selected with probability ϵ. The higher the values of ϵ, the more exploratory the combined list is. With $\epsilon = 0$, the resultant list of documents is purely exploitative.

Listwise approaches aim to optimize an evaluation measure that takes into consideration an entire document list. The exploration – exploitation balance in this approach is introduced through the dueling bandit algorithm (Yue and Joachims, 2009) (Section 5.1), which requires only relative evaluation of the quality of two document lists through implicit feedback. In this approach, there is a method $f(list_1, list_2)$ for comparing two document lists. At each round t, the algorithm observes a query, based on which two document lists are produced: one exploitative and one exploratory. The exploitative list is produced using the current

exploitative weight vector w_t, which had the best performance so far. The exploratory list is produced using an exploratory weight vector w_t' generated by moving w_t in a random direction by a pre-defined step of size σ. The two lists are then compared using the function $f(list_1, list_2)$. If the exploratory weight vector w_t' is considered to provide the better ranking compared to the current exploitative weight vector w_t, then w_t is updated by moving it towards w_t' by step size α.

The particular comparison function is inspired by the balanced interleave method (He *et al.*, 2009), which constructs an interleaved result list by randomly selecting a starting list and then interleaving the two lists. After observing user clicks on the result list, a preference between the two lists is inferred based on the number of clicks that its top K results received. The proposed interleave function is more probabilistic in nature by randomly selecting the list to contribute the document at each rank of the result list. The parameter α controls the exploration rate affecting the selection of the exploratory list, i.e. the higher the value of α, the more exploratory documents there will be in the combined list. For each rank of the result list to be filled, the algorithm picks one of the two lists and the highest ranked document that is not yet in the combined result list, is added at this rank. The result list is displayed to the user. For each clicked document, a click is attributed to one of the contributing lists.

The two pairwise and listwise approaches for incorporating the exploration – exploitation trade-off in learning to rank were tested extensively on two standard collections for learning to rank: LETOR 3.0 and LETOR 4.0 (Liu *et al.*, 2007) using the Dependent Click Model (Section 3.2). Overall, in pairwise learning when the feedback is noisy, increasing exploration rate improves online performance of the approach, with the best performance achieved when result lists contain about one half exploratory and one half exploitative documents. When feedback is reliable, online performance is best in a purely exploitative setting. In listwise learning, balancing exploration and exploitation can significantly improve online performance over a purely exploratory algorithm – best performance is achieved with low level of exploration, resulting on average in only two documents from the exploratory list.

In the formulation of learning to rank with exploration and exploitation proposed by Vorobev *et al.* (2015), each unique query is associate with a dedicated bandit algorithm. Each document related to the query is treated as an arm. At each step, as a response to the issued query the algorithm generates a search result page (SERP) and observes the user's response. User's satisfaction with a document translates into a reward of 1 for the corresponding arm, where satisfaction is defined as a click with a long enough dwell time or the last click on the SERP. An arm is considered to be pulled only if the user examined the snippet of the corresponding document. The cumulative reward over the first K query issues equals the cumulative number of satisfied clicks.

In this approach, the bandit algorithm is assumed to work for each query independently. The number of documents considered by each bandit (corresponding to a query) is quite small and is chosen for exploration on the basis of prior feedback and general knowledge about the query and the user. In order to create a list of documents to present to the user, the documents with the highest upper confidence bound are used to fill the top k positions in the result list. There are several ways to balance the exploration and exploitation in this approach. First, a ranking obtained from any learning to rank algorithm can be combined with prior distributions of upper confidence bounds for a given bandit algorithm. Then, the bandit algorithm starts from the performance of the learning to rank algorithm and improves over iterations according to the collected information. Second, the user behavior features participating in the learning to rank model can be periodically updated by making use of clicks gathered by the bandit algorithm. Third, query-document features can be used to propagate this information over other documents or queries.

The approach was tested with collected logs of live stream of search queries submitted to Yandex during a two-week period using UCB1 (Auer *et al.*, 2002a) (Section 2.2) and Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1). Both algorithms increase the performance of the search system in terms of cumulative clicks over a 10 day period compared to the currently used baseline in the search system. In general, utilizing prior information from the current production system as well

as increasing the level of exploration greatly increases the cumulative performance of the proposed approach.

4.2 Off-line Policy Evaluation

Optimizing an interactive search system can be challenging due to the lack of appropriately labelled, idiosyncratic personal preferences of users, and complexities of search engines, where one change in a small component of the system can have a major impact on the ranked results. Initial studies of this problem concentrated on theoretical analysis of how to find an optimal bandit policy off-line using a set of existing data. *Exploration scavenging*[2] (Langford *et al.*, 2008) examines the problem of evaluating a policy in the contextual bandit setting using only observations collected during the execution of another policy and provide theoretical results for a principled method for policy evaluation as long as the exploration policy chooses arms independent of side information and each action is explored sufficiently often. In a similar vein, Strehl *et al.* (2010) theoretically evaluate a method for solving the warm start problem for exploration from logged data in the contextual bandit setting. The approach is based on propensity score (Horvitz and Thompson, 1952), which allows for importance weighting of the bias and estimation based on the previously gathered data.

These theoretical analyses gave rise to a number of studies of off-line policy evaluation in an IR setting. An unbiased offline evaluator proposed by Li *et al.* (2011b) takes as input a bandit algorithm i and a desired number of valid events on which to base the evaluation. The method steps through the stream of logged events one by one. If, given the current history H_{t-1}, the policy π_i chooses the same arm as the one selected by the logging policy, then the event is added to the history, and the total payoff of policy pi_i updated. If, however, the policy π_i selects a different arm from the one taken by the logging policy, then the event is entirely ignored, and the algorithm proceeds to the next event without any change in its state.

[2]See also Section 7.3 for application of *Exploration scavenging* to web-page layout optimization with bandits.

The offline evaluation method was applied to a large-scale, real-world problem with variable arm sets, i.e. news recommendation on Yahoo! front page Today module. The offline method was used with three bandit algorithms: ϵ-greedy (Sutton and Barto, 1998), UCB (Auer, 2002) and LinUCB (Li *et al.*, 2010a). CTR estimates for the three algorithms show that the evaluation results are highly consistent across different random runs with small variance.

The method was further developed with the aim of inferring the average reward $\rho(\pi)$ if policy π is used to choose actions in the bandit problem (Li *et al.*, 2015). The approach relies on randomized data collection. At each round t, the environment chooses a context x and reward ρ for each action (arm) a from some unknown distribution and only reveals the context. Based on the context, a multinomial distribution is computed over the actions A. A random action a is drawn according to the distribution, and the corresponding reward ρ_a and probability p_a are logged. This process produces a set of exploration data in the form (x, a, ρ_a, p_a). The exploration data can be used to evaluate the value of an arbitrary policy π offline using the following estimator:

$$\hat{\rho}_{offline}(\pi) = \sum_{x,a,\rho_a,p_a} \frac{\rho_a \cdot \mathbb{S}(\pi(x) = a)}{p_a},$$

where $\mathbb{S}(c)$ is the set-indicator function that evaluates to 1 if condition c holds true, and 0 otherwise. This estimator is unbiased for any π, provided that every component p in the probability distribution is nonzero, which allows randomize action selection.

When comparing the offline estimates of two policies π, it is necessary to obtain reliable confidence intervals to assess whether the difference in the point estimates by the above equation are significant. First, from the exploration dataset D, the unbiased estimate of the standard deviation $\hat{\sigma}$ of the random variable $\frac{\rho_a \cdot \mathbb{S}(\pi(x)=a)}{p_a}$ is computed. Then, a 95% confidence interval is constructed. Such an approximation works well when the size of D is large and the ratios ρ_a/p_a are bounded.

Li *et al.* (2015) provide a detailed analysis of using the above approach to optimize *speller* – a crucial component in a commercial search engine. It enables the translation of queries with typing errors into their correct forms so that the search engine can match and rank relevant

results. The focus of their analysis is to train a model that selects a subset of candidates with corrected spelling for a misspelt input query. The goal of selecting multiple query candidates is to mitigate the risk of proposing an inappropriate correction for a given context or user.

Jie *et al.* (2013) propose a unified framework based on contextual bandit for the search federation problem, i.e. merging the vertical results, such as images, news, shopping, etc., with the web documents to show to the user. Given a query, the system predicts the expected reward for each vertical taking into consideration implicit user feedback, then organizes the SERP to maximize the total reward. Click-skip based reward, which assumes a cascade model of user behavior (Craswell *et al.*, 2008) (Section 3.1), is the proxy for user satisfaction and reward calculation.

The system has runtime and offline components. When a query comes to the search engine, it is sent to the federation layer to gather results from various backends responsible for different verticals. Based on the ϵ-greedy algorithm (Sutton and Barto, 1998) (Section 2.2), the layer decides whether to perform exploration on the query. If exploration is chosen, the verticals are slotted randomly and the features along with the context are logged.

In the exploitation mode, based on the predicted rewards of the verticals and the web documents, the algorithm starts from the top of the page, and decides the vertical to be slotted at each allowed position subject to business constrains. For each position, the vertical with the highest expected reward is selected and it is slotted there as long as its expected reward is higher than that of the web document intended for the same slot. Offline, the logs are aggregated and processed to compute the rewards for each of the events of interest (Li *et al.*, 2010a). Based on that, model(s) are trained and evaluated. The best model is used in the runtime system. When deployed in Yahoo!, the system significantly increased CTR.

4.3 Query Auto-completion and Recommendation

In information retrieval, query auto-completion (QAC) refers to the following functionality: given a prefix consisting of a number of characters entered into a search box, the user interface proposes alternative ways of extending the prefix to a full query (Cai and De Rijke, 2016). Ranking query completions is a challenging task due to the limited length of prefixes entered by users, the large volume of possible query completions matching a prefix, and the broad range of possible search intents. In this section, we describe how bandits algorithms can be applied to query auto-completion.

Durand *et al.* (2017) tackle the QAC problem using a mixture of engines. At time t, a list of K auto-completion suggestions is displayed to the user according to the current query prefix. User satisfaction with a given query is measured through user clicks. The aim is to assign an engine to each slot of the QAC list such that this engine is responsible for providing the query suggestion displayed in this slot. Two approaches are proposed to select the engine to use for each rank position: the ranked model based on the ranked bandits algorithm (Radlinski *et al.*, 2008a) (Section 4.1) with one bandit algorithm for each slot, and the cascade model, based on the cascade bandits algorithm (Kveton *et al.*, 2015a) (Section 3.1.1) The ranked model does not share information obtained from feedback on the same engine placed in different slots. The cascade model for query recommendation uses a single bandit algorithm for the whole setting and it merges all information obtained for a given engine so far irrespective for which slot it was used. These approaches were employed to learn a mixture of four real engines for filling 5 positions of a QAC field using three real datasets built by taking full length queries performed on websites over a one month period. The cascade model manage to gather 5% more clicks than the ranked model.

Exploitative Ranked Bandits Algorithm (ERBA) (Wang *et al.*, 2017b) is another algorithm for QAC based on ranked bandits (Radlinski *et al.*, 2008a). While ERBA is an exploitative counterpart of ranked bandits, i.e. ERBA chooses the next best arm upon conflicts, the ranked bandit algorithm chooses an arbitrary arm, which is a purely exploratory approach. Thus, if the recommended query has already been selected at

rank k, ERBA selects the second query as the next best arm excluding the presented arms according to some criteria of the particular bandit algorithm used to make a query suggestion at a given rank. To utilize prior knowledge from query logs, Thompson Sampling (Chapelle and Li, 2011) (Section 2.2.1) is used to select a query suggestion at each position. Experimental results with Yahoo! search query logs show that Thompson sampling ERBA substantially outperforms UCB1 ranked bandits in terms of CTR for different lengths of queries.

A modified version of Thompson sampling (Chapelle and Li, 2011) has also been applied to query recommendation (Hsieh *et al.*, 2015). The proposed algorithm makes a simplifying assumption that the engagement rate on a specific search suggestion is independent of the other $K-1$ suggestions present, which is necessary to accommodate multiple query suggestions per request. This assumption allows the reduction of the size of the action space. Based on observations of different search scenarios in practice, two additional modifications were made. First, the F parameter in the Beta distribution indicating the number of failures is updated by $\frac{1}{K-1}$ instead of 1. In this way, "one" failure (but not $K-1$ failures) is assigned if one success is conferred to some arm at each round, and the failure is shared by all the remaining arms. This helps to prevent potential under-exploration that might occur as a result of the independence assumptions. Second, a rate parameter γ, to accommodate the ignorance of not observing a user action, was introduced. Intuitively, γ is a scale factor to control the rate at which the posterior uncertainty decreases when no user decision is spotted. For example, at $\gamma = K$, the algorithm sees no-response as dissatisfaction.

Li *et al.* (2016a) address the issue of the user's complex information needs. In a majority of retrieval systems, a single active query is used to retrieve relevant documents. When the user's information need has multiple aspects, the query must represent the union of these aspects. The solution is to keep multiple, simpler queries active and, based on the user's feedback, these queries take turns to retrieve documents. The number of relevant documents, treated as rewards and obtained from past result pages of each query, indicates which query to explore next. The approach is based on UCB bandits with a sliding window (Garivier and Moulines, 2011), which considers only the last i plays. In many

cases, however, the subtopics of the information need of the user are not defined upfront and gradually change during the search process. New queries can be generated through an interactive retrieval process, for example through relevance feedback, and added to the bandit algorithm used to retrieve the initial search results. An algorithm inspired by Monte Carlo tree search (Srinivasan *et al.*, 2015) is proposed, where the UCB score of a new query is estimated by its similarity to existing queries in the pool. The similarity is calculated using a bag of words representation of queries and Gaussian Process kernel (see Section 6.1 for more details). For a new query i_n, a new arm n is introduced and its UCB score is estimated as:

$$UCB_n(t) = \frac{\hat{R}_n(t)}{\hat{N}_n(t)} + c\sqrt{\frac{\log \min(t, \tau) + \hat{N}_n(t)}{\hat{N}_{n(t)}}},$$

where $\hat{R}_n(t)$ is the estimate sum of rewards of the n-th arm in the last τ rounds, $\hat{N}_n(t)$ is the estimated number of times arm n has been played in the first t rounds and c is a constant controlling the tendency to explore the new arm.

4.4 Summary

While Chapter 3 describes a number of new bandit algorithms inspired by click models developed in IR, this chapter focuses on specific applications of bandits in IR (e.g. query auto-completion). The case studies and applications discussed above again show the dichotomy between independent multi-armed bandits, such as UCB1, and contextual or Bayesian bandits, such as Thompson sampling. For example, initial implementations of the ranked bandit algorithms based on UCB, although easy to implement, do not scale to large datasets and perform much worse than later implementations based on contextual bandits, which also scale up to larger datasets. Thompson sampling, or its various modifications, seems to be the algorithm of choice for many applications – it is easy and quick to implement and at the same time it allows to incorporate prior knowledge about the user or the system, which greatly speeds up the learning process.

Another observation concerns the role of exploration in information retrieval applications. A number of applications discussed in this chapter used the ϵ-greedy algorithm (Sutton and Barto, 1998) or its variants as their benchmark. Compared to other bandit policies, most notably UCB and Thompson sampling, ϵ-greedy is more aggressive in terms exploitation. In general, ϵ-greedy tends to underperform in comparison to simple UCB strategies as well as contextual and Bayesian bandits, which points to the importance of exploration in many IR applications. This preference for more exploratory results may be due to the noise in user response or the search system's inability to understand the user's intent, which may be mitigated through exploration thus giving the user more varied selection.

5

Ranker Evaluation

Traditional ranker evaluation methods rely on offline metrics, such as MAP or nDCG (Sanderson, 2010). However, such methods rely on the existence of test collections, which do not necessarily reflect the needs of the current user of an IR system. Online evaluation can be performed using interleaved comparison methods (Hofmann *et al.*, 2013a; Joachims, 2003) and multileaved comparisons (Schuth *et al.*, 2015). This chapter covers approaches to online ranker evaluation using bandit algorithms in conjunction with various interleaving methods. The recent increased interest in online ranker evaluation led to the development of a number of new bandit algorithms, in particular a new class of algorithms called dueling bandit algorithms. The majority of this chapter is devoted to the dueling bandit problem (Section 5.1) and its extension multi-dueling bandits (Section 5.4). The last section of the chapter discusses the application of bandits to test collection assessments (Section 5.5).

5.1 Dueling Bandits and Interleave Filtering

In dueling bandits learning happens "on-the-go" through preference feedback, i.e. from comparisons between a pair of actions (Busa-Fekete *et al.*, 2018; Sui *et al.*, 2018). This approach makes dueling bandits

well-suited for modeling settings that elicit subjective human feedback through preference form.

Yue and Joachims (2009) proposed an online learning framework for real-time learning from observed user behavior. The approach only requires pairwise comparisons that can be inferred from implicit feedback. In the dueling bandits problem[1] the only actions are comparisons (or duels) between two points within a space X (e.g., a parameterized space of retrieval functions in a search engine). Any single comparison between two points x and x' (e.g., individual retrieval functions) is determined independently of all other comparisons with probability

$$P(x \succ x') = \frac{1}{2} + \epsilon(x, x'),$$

where $(x, x') \in [-1/2, 1/2]$. In the search example, $P(x \succ x')$ refers to the fraction of users who prefer the results produced by x over those of x'. The measure $\epsilon(x, x')$ can be regarded as the distinguishability between x and x'. Algorithms can then learn only via observing comparison results, e.g. interleaving (Radlinski *et al.*, 2008b).

The following regret formulation is used to quantify the performance of an on-line algorithm:

$$R = \sum_{t=1}^{T} \epsilon(x^\star, x_t) + \epsilon(x^\star, x_t^\prime),$$

where x_t and $x_t^\prime$ are the two points selected at time t and $x^\star$ is the best point known only in hindsight. The regret corresponds to the fraction of users who would prefer the best retrieval function $x^\star$ over the selected ones x_t and $x_t^\prime$.

Yue and Joachims (2009) implemented the dueling bandit problem through the Dueling Bandit Gradient Descent (DBGD) algorithm. DBGD maintains a candidate x_t and compares it with a neighboring point $x_t^\prime$ along a random direction. If $x_t^\prime$ wins the comparison, then an update is taken along that direction, and then projected back into the

[1] See Busa-Fekete *et al.* (2014) for an approach to PAC rank elicitation similar to dueling bandits. The method consists of sorting a given set of options based on adaptive sampling of stochastic pairwise preferences. For theoretical analysis on reducing the dueling bandit problem to a stochastic MAB (see Ailon *et al.*, 2014).

space X. DBGD requires two parameters which can be interpreted as the exploration σ and exploitation α step sizes. The α parameter is required for the gradient descent algorithms used in DBGD. Since DBGD probes for descent directions randomly, this introduces a gradient estimation error that depends on the parameter σ.

The dueling bandit gave rise to a number of algorithms that can be applied in ranking optimization. Interleaved Filter (IF) (Yue *et al.*, 2012a) is an exploration algorithm that aims to selected the optimal bandit[2]. It maintains a candidate bandit $\hat{b}$ and, through simulations, compares $\hat{b}$ with all other remaining bandits via round robin scheduling (i.e., interleaving). Any bandit that is empirically inferior to $\hat{b}$ with $1-\sigma$ confidence is removed. The value of σ is calculated as $\sigma = 1/TK^2$, where T indicates the number of iterations and K is the number of bandits in the dataset. When a bandit b is empirically superior to $\hat{b}$ with $1-\sigma$ confidence, then $\hat{b}$ is removed and b becomes the new candidate $\hat{b}$. Afterwards, all empirically inferior bandits are removed. This process is repeated until only one bandit remains, which is the best bandit.

In later work, Yue and Joachims (2011) extend the dueling bandits problem to a relaxed setting where preference magnitudes can violate transitivity. The approach is called *Beat-the-Mean*. The assumption in the original setting of the dueling bandit problem (Yue *et al.*, 2012a) was that user preferences satisfy strong transitivity. For example, if there are three strategies A, B and C and users prefer A to B by 55%, and B to C by 60%, then by strong transitivity, users will prefer A to C at least 60%. However, results from experimental interleaving show that strong transitivity is often violated in practice. Furthermore, on top of weak transitivity, Yue and Joachims (2011) made the additional assumption of triangular inequality, such that for any triplet of bandits $b_1 \succ b_j \succ b_k$, the distinguishability of bandit pairs is $\epsilon(b_1, b_k) \leq \epsilon(b_1, b_j) + \epsilon(b_j, b_k)$. This assumption can be viewed as a diminishing returns property.

Beat-the-Mean proceeds in a sequence of rounds and maintains a working set W_t of active bandits during each round t. For each active bandit $b_i \in W_t$ an estimate is maintained for how often b_i beats the

[2]See Urvoy *et al.* (2013) for a generalization of dueling bandits for situations where the environment parameters reflect the idiosyncratic preferences of a mixed crowd.

mean bandit $\bar{b}_t$ of W_t, where comparing b_i to $\bar{b}_t$ is functionally identical to comparing b_i with a bandit sampled uniformly from W_t. In each iteration, a bandit with the fewest recorded comparisons is compared with $\bar{b}_t$. The iteration ends when a bandit $b^{'}$ is separated from the best one by a sufficient confidence margin. All the comparisons involving $b^{'}$ are removed, and $b^{'}$ is removed from W_t. The algorithm terminates when only one active bandit remains.

Beat-the-Mean algorithm was compared in simulations against the Interleave Filter (Yue *et al.*, 2012a). In a setting where strong transitivity holds, the average regret of both methods increases linearly with the number of bandits considered, however, the regret of Interleave Filter shows much higher variance. In a setting where stochastic preferences only satisfy relaxed transitivity, similar behavior can be observed, however, Interleave Filter suffers from super-linear regret with significantly higher variance. This indicates that, compared to Beat-the-Mean, Interleave Filter is not very robust when transitivity is relaxed.

Dudık *et al.* (2015) consider the problem of learning in the dueling bandit setting using contextual information. The approach is rooted in a game theoretic concept of *von Neumann winner* – a randomized policy that beats or ties every other policy and does not require strong, and often unrealistic, assumptions found in previous dueling bandit approaches. In the relative feedback setting of dueling bandits, it is often difficult to determine what the overall best action (arm) is as there is no measure of the absolute quality of actions. Through relaxing the assumption that there is a single action (arm) that can beat all the others, it is easier to find a good solution to the dueling bandit problem. The idea is to find a probability vector w in ΔK, where ΔK is the simplex of vectors in $[0,1]^K$ whose entries sum to 1, such that $\sum_{i=1}^{K} w(i)P(i,j) \geq 0$ for all arms j, where $P(i,j)$ is the expectation of arm i to beat arm j, which itself is an entry in a preference matrix P. Thus, for every arm j, if i is selected randomly according to distribution w, the chance of beating j in a duel is at least $1/2$. A distribution w with this property is called a von Neumann winner for the preference matrix P.

In order to incorporate context into this scenario, it can be assumed that at each round t, a context c_t and preference matrix P_t are chosen by Nature. The context is revealed to the learner but the preference matrix remains hidden. Based on the context c_t, then the learner selects two arms for a duel, however, the expectation of the outcome is determined by the (hidden) preference matrix. The goal is to learn which action to select as a function of the context, i.e. to learn the mapping π from contexts to arms. This allows the extension of von Neumann winner to incorporate context by reducing it to the non-contextual setting. Each mapping π can be regarded as a meta-arm and a preference matrix can be defined over these meta-arms. Thus, von Neumann winner in the contextual setting can be defined as an (ordinary) von Neumann winner for the meta-preference matrix with mappings. Detailed studies on how to compute (or approximate) contextual von Neumann winners can be found in Dudık *et al.* (2015).

5.2 Condorcet Winner

As mentioned earlier, in dueling bandits at each time-step, two rankers (i, j) are compared and with some probability ranker i beats ranker j, with preference probabilities stored in matrix $P = [p_{i,j}]$. A *Condorcet winner* (Urvoy *et al.*, 2013) is a ranker $rank_1$ such that $p_{1,i} > \frac{1}{2}$ for all $i > 1$, i.e. when interleaved with any other ranker, the Condorcet winner is expected to win. In this section, we describe a number of related algorithms that select rankers to compare by first choosing a ranker that might be a good candidate for the Condorcet winner, while the second selected ranker is a candidate that has the best chance of disproving the hypothesis that the first ranker is the Condorcet winner.

Zoghi *et al.* (2014b) extends the Upper Confidence Bound (UCB) algorithm (Auer *et al.*, 2002a) to the K-armed dueling bandit setting (Yue *et al.*, 2012a). The approach uses estimates of the pairwise probabilities to select a promising arm and applies UCB with the winner as a benchmark. The *Relative Upper Confidence Bound* (RUCB) is the implementation of this approach. While previous dueling bandit algorithms, such Interleave Filter (IF) (Yue *et al.*, 2012a) and Beat-the-Mean (BTM) (Yue and Joachims, 2011) (Section 5.1) use the exploration horizon to

set their internal parameters so that for each t, there is a separate algorithm IF_t or BTM_t, RUCB does not require the specification of the horizon as input. This makes RUCB more practical in application because it is difficult to know in advance how to set the horizon: if the horizon is set too long, the algorithm is too exploratory and takes longer to find the best arm, and if it is too short, the algorithm may fail to find the best arm before the horizon is reached.

The RUCB algorithm can be described as follows. In each time step t, RUCB goes through three stages. First, all the arms are placed in a pool of potential champions. Then, each arm is compared against all other arms optimistically and the upper confidence bound is computed: $U_t(ij) = \mu_t(ij) + c_t(ij)$, where $\mu_t(ij)$ where is an estimation of the number of times arm i beats arm j at time t, and $c_t(ij)$ is an optimism bonus that increases with t and decreases with the number of comparisons between i and j. In the case of $U_t(ii)$, its value is set to $\frac{1}{2}$. If $U_t(ij) < \frac{1}{2}$ for any j, then arm i is removed from the pool. If there is a single potential champion left at the end of this process, then this arm is put in the set $\mathcal{B}$ of the hypothesized best arm. The set $\mathcal{B}$ is either empty or contains only one arm. An arm is removed from $\mathcal{B}$ if it loses to another arm. Next, from the remaining potential champions, a champion arm is selected by sampling uniformly at random if $\mathcal{B}$ is empty; and if $\mathcal{B}$ is not empty, the probability of picking the arm in $\mathcal{B}$ is set to $\frac{1}{2}$, while the remaining arms are given equal probability of being chosen. In the second stage, regular UCB is performed on the set of arms $a_{1,c} \ldots a_{K,c}$, where a_c indicates the current reference arm. The arm $a_j = \arg\max_j U_t(jc)$ is selected. In the third stage, the pair of arms (a_c, a_j) is compared and their similarity scores are updated.

RUCB was compared to Beat-the-Mean bandit algorithm in ranker evaluation using the interleaved comparison (Radlinski *et al.*, 2008b). In this setting, documents proposed by two different ranking functions are interleaved and the resultant list is presented to the user. The user's clicks are then used to infer noisy preferences for one of the ranking functions, which can be modelled as a k-armed dueling bandit problem with each arm corresponding to a ranking function. The data used for the experiment consisted of randomly chosen subsets from the pool of 64 ranking functions provided by LETOR, a standard IR dataset,

yielding k-armed dueling bandit problems with $k \in \{16, 32, 64\}$. In all the experiments, RUCB accumulates the least regret, while the regret of Beat-the-Mean grows linearly with time.

The Relative Minimum Empirical Divergence (RMED) algorithm (Komiyama *et al.*, 2015) improves upon the theoretical results for RUCB. For each arm, the algorithm computes the empirical divergence defined as

$$I_i(t) = \sum_{j|p_{i,j}<1/2} d(\hat{p}_{i,j}(t), 1/2) N_{i,j}(t),$$

where $\hat{p}_{i,j}(t)$ is the algorithm's empirical estimate of the preference probability $p_{i,j}$ at time t, and $N_{i,j}(t)$ is the number of comparisons between arm i and any arm j that beats i in the first t time-steps, while $d(p, q)$ is the KL-divergence between two Bernoulli distributions with parameters p and q. The likelihood that arm i is a Condorcet winner is $\exp(-I_i(t))$ and it is used to pick the arms that will be compared against each other at time $t + 1$.

Simulation results in a setting similar to the one used for testing RUCB (Zoghi *et al.*, 2014b) described above in this section show that RMED significantly outperforms RUCB and similar dueling bandit algorithms in terms of regret accumulation.

Relative Confidence Sampling (RCS) (Zoghi *et al.*, 2014a) is an algorithm related to RUCB. It differs from RUCB in one crucial respect: the use of sampling when conducting a round robin tournament to select a champion arm. The intuition behind this change is that improved performance can be obtained by maintaining posterior distributions over the expected value of each arm and sampling from those posteriors to determine which arm to select. Sampling based methods, which rely on posteriors, do not easily gravitate toward the extremes thus leading to the selection of more appropriate arms.

The RCS algorithm takes as input a set of ranking functions (rankers) and an oracle, e.g. an interleaved comparison method that returns a noisy estimate of which ranker is the winner. RCS has one parameter α, which controls the exploration rate of the algorithm: the higher the value of α, the longer it takes to settle on a single ranker. RCS also maintains a scoresheet where all the comparisons are stored. The algorithm proceeds in two phases. In phase one, a tournament is simulated based on the

current scoresheet, i.e., samples θ_{ij} are collected for each pair of rankers (i, j) with $i > j$ from the posterior Beta distribution. RCS sets $\theta_{ji} = 1 - \theta_{ij}$ and $\theta_{ii} = \frac{1}{2}$. Ranker i beats ranker j in the simulated tournament if $\theta_{ij} > \frac{1}{2}$ There are two ways in which a champion ranker could be selected. In the first scenario, the champion ranker is the ranker that beats all other rankers in this tournament. If no ranker beats all other rankers, then the champion ranker is the one that has been the champion least often. After a number of iterations, enough inferior rankers will be eliminated for the algorithm to always select the best option. The second stage of the algorithm is similar to RUCB (Zoghi *et al.*, 2014b) described above in this section, i.e. the UCB is calculated for all the arms with the current champion as reference and using the same formula as RUCB. RCS picks the ranker with the highest UCB. Finally, the champion ranker and the ranker with the highest UCB are compared against each other using a real interleaved comparison and the scoresheet is updated accordingly.

The RCS algorithm was compared against RUCB using two large-scale learning to rank datasets: the Microsoft Learning to Rank dataset and the Yahoo! Learning to Rank Challenge dataset. Probabilistic interleave (Hofmann *et al.*, 2011a) was used to compare a pair of rankers and a probabilistic user model (Craswell *et al.*, 2008) was used to model the user's click behaviour. RCS accumulates on average a third less regret than RUCB, i.e. RCS finds the best ranker faster and makes less severe errors compared to RUCB.

Although RUCB and RCS outperform earlier dueling bandit algorithms, such as Interleave Filter (Yue *et al.*, 2012a) and Beat-the-Mean (Yue and Joachims, 2011) (Section 5.1), they have difficulty scaling to large K. Interleave Filter and Beat-the-Mean make additional assumptions to facilitate the identification of the best ranker, however, these assumptions may be too restrictive for specific applications, such as web search. mergeRUCB (Zoghi *et al.*, 2015b) is an algorithm that bridges the gap between these two approaches: it makes only weak assumptions about the k-armed dueling bandit problem but requires only $O(K)$ comparisons and therefore performs well when many rankers need to be compared. mergeUCB groups rankers into small batches, which reduces the number of comparisons before rankers can be eliminated.

When comparing different rankers, one comparison may often not be enough to determine which of a pair of rankers is better since feedback is stochastic. The number of required comparisons may be even larger if two rankers are similar and it is difficult to distinguish between them. This case is problematic because both rankers might be weak in the overall pool of rankers and comparing them to each other multiple times will increase the regret. In mergeRUCB, the best ranker in the batch is used to eliminate the rest. If a batch contains only similar rankers and is slow in eliminating rankers, it is combined with other more varied batches.

Similarly to RUCB, mergeRUCB proceeds in stages. Before the first stage, the rankers are grouped into small batches $\mathcal{B}_i$. Within each stage, interleaved comparisons are performed among rankers from the same batch. At any given time, the choice of rankers to compare against each other, and to eliminate throughout the process, is determined by their UCBs, which are calculated in the same manner as in RUCB (Zoghi *et al.*, 2014b) described above. The algorithm proceeds until the number of remaining rankers becomes small. At this point, the stage is concluded and by pairs of batches are merged together to form bigger batches. This initiates the next stage, and the process repeats until a single ranker remains.

MergeRUCB was compared against RUCB and Beat-the-Mean using four large-scale learning to rank datasets: the Microsoft Learning to Rank dataset, the Yandex Internet Mathematics 2009 dataset and two of the Yahoo! Learning to Rank Challenge datasets. Probabilistic interleave (Hofmann *et al.*, 2011a) was used to compare a pair of rankers and a probabilistic user model (Craswell *et al.*, 2008) was used to model the user's click behaviour. The experiments show that, as the number of rankers increases (going from 136 to 245 to 700), so does the difference between the performance of mergeRUCB and the remaining algorithms, i.e. the higher the number of arms, the higher the difference in regret between the three algorithms, with the regret of mergeRUCB being the lowest.

5.3 Copeland Dueling Bandits

Copeland Dueling Bandits (Zoghi *et al.*, 2015a) is another extension of the k-armed dueling bandit problem. While RUCB (Zoghi *et al.*, 2014b) (Section 5.2) and its derivatives makes the assumption that there is an arm that beats every other arm with probability greater than 1/2, Copeland Dueling Bandits relax this assumption and instead focus on finding Copeland winners, i.e. arms that beat the greatest number of other arms.

Two algorithms are proposed for the Copeland setting: the *Copeland Confidence Bound* (CCB) and Scalable Copeland Bandits (SCB). CCB is inspired by RUCB (Zoghi *et al.*, 2014b) and is based on the principle of optimism followed by pessimism. It maintains optimistic and pessimistic estimates of the preference matrix, where the optimistic preference matrix is used to select the potential winner and the pessimistic preference matrix is used to select an opponent. The opponent is an arm considered likely to discredit the hypothesis that the arm selected from the optimistic preference matrix is indeed a winner. The optimistic estimate of the score of each arm is calculated based on its past performance, i.e. how a given arm fared as an optimistic winner throughout history. The "short-listed" possible winners are placed in set $\mathcal{B}_t$. To maintain $\mathcal{B}_t$, when the optimistic score of an arm is lower than the pessimistic score of another arm, the arm is removed from $\mathcal{B}_t$. The opponent arm is selected based on its confidence interval maintained by the preferences matrices for each arm a_i and a_j. In relation to a given arm a_c, there are three types of arms: (1) arms a_j with the confidence region strictly above 0.5, (2) arms a_j with the confidence region strictly below 0.5, and (3) arms a_j where the confidence region contains 0.5. An arm of type (1) or (2) may become an arm of type (3) as the size of the confidence intervals increases as time goes on. CCB always chooses a winner from arms of type (3) as they are the most informative about the optimistic score of arm a_c. The favoured arms of type (3) are the ones that confidently managed to beat arm a_c in the past i.e., arms that in some past round were of type (2). Such arms are maintained in a shortlist of "formidable" opponents $\mathcal{B}_t^i$ that are likely to confirm that

a_i is not a winner. The sets $\mathcal{B}_t^i$ speed up the elimination of suboptimal winners.

The SCB algorithm is designed to handle dueling bandit problems with large numbers of arms. It relies on a reduction to a k-armed bandit problem with direct access to a noisy version of the score of arm a_i, which is obtained by comparing a_i to a random arm a_j until it becomes clear which arm beats the other. A KL-divergence based arm elimination algorithm (Garivier and Cappé, 2011) is used to implement an elimination tournament with confidence regions based on the KL-divergence between probability distributions. Combining this approach with the squaring trick greatly reduces the number of partitions required for comparing different arms. SCB repeatedly compares arm a_i to other arms but force-terminates if an increasing threshold is reached. If it terminates early, then the identified arm is played against itself until the threshold is reached.

CCB and SCB were compared to RUCB in the setting of ranker evaluation described in more detail in Section 5.2 above and in (Zoghi *et al.*, 2014b). RUCB's regret grows linearly, while that of CCB and SCB flattens out as the number of iterations increase. In general, the regret of CCB is about 3/4 lower compared to SCB.

The Copeland Winners Relative Minimum Empirical Divergence (CW-RMED) algorithm (Komiyama *et al.*, 2016) also exploits the structure of the Copeland dueling bandit problem. Compared to CCB algorithm discussed above, it requires less exploration to find winner arms.

At the beginning of each round t, CW-RMED checks if there exists a pair of arms i, j that has not been drawn $O(\log t)$ times or if $\hat{p}_{i,j}(t)$ – the estimate of preference probability $p_{i,j}$ at time t – is very close to 1/2. If one of these conditions is met, then the algorithm immediately draws that pair. Otherwise, it enters the loop that sequentially draws each pair from the sets of superiors of arm i, i.e. the set of arms that beat arm i. If the observation is sufficient to identify an arm as a Copeland winner, it exploits by adding it to the candidates of the pairs that will be drawn in the next loop. Otherwise, it draws the pairs with the number of observations below the minimum requirement for identifying arm i as a winner with high confidence. A computationally

more efficient version of CW-RMED called ECW-RMED reduces the amount of exploration even further. In experimental results with the Microsoft Learning to Rank dataset in a setting similar to the one used in Zoghi *et al.* (2014b), ECW-RMED significantly outperformed CCB in terms regret accumulation.

Thompson sampling (Section 1) has also been applied to general Copeland dueling bandits. The Double Thompson Sampling (D-TS) algorithm (Wu and Liu, 2016) selects the first and the second candidate through Thompson sampling. For each pair of arms a_i, a_j (with $i \neq j$), there is a Beta prior distribution for its preference probability $p_{i,j}$. These distributions are updated according to the comparison results $B_{i,j}(t-1)$ and $B_{j,i}(t-1)$, where $B_{i,j}(t-1)$ (and $B_{j,i}(t-1)$) indicates the number of times when arm a_i (a_j) beats arm a_j (a_i) up to time t.

At each time t, the algorithm has two phases to select the first and the second candidate arms. When choosing the first candidate, first the RUCB (Section 5.2) of $p_{i,j}$ is used to create a set of possible winners by eliminating the arms that are unlikely to be the Copeland winner. The algorithm then samples $\theta_{i,j}(t)$ from the posterior Beta distribution and the first candidate is selected by majority voting, i.e., the arm within the set of possible winners that beats the most arms according to $\theta_{i,j}(t)$ is selected to be the first candidate. The ties are broken randomly. A similar procedure is applied to select the second candidate a_t – new samples $\theta_{i,t}$ are generated and the arm with the largest θ selected as the second candidate.

The D-TS algorithm was tested using the Microsoft Learning to Rank dataset, with a setting similar to the one described in Zoghi *et al.* (2015a) – a preference matrix for 136 rankers is derived, where each ranker is a function that maps a user's query to a document ranking and can be viewed as one arm in dueling bandits. In Condorcet dueling bandits, D-TS performs much better than existing algorithms, in particular when compared to RCS (Zoghi *et al.*, 2014a) with respect to regret. In non-Condorcet dueling bandits, D-TS significantly reduces the regret compared to the UCB-type algorithms.

5.4 Multi-dueling Bandits

Early applications of dueling bandit algorithms to ranker evaluation (Section 5.1) focused on comparing only two rankers at a time. Brost *et al.* (2016b) propose a generalization of the dueling bandit problem that uses simultaneous comparisons of an unrestricted number of rankers. The multi-dueling bandit algorithm maintains optimistic estimates of pairwise winning probabilities and plays arms that have a chance of being the winner. When there is a single candidate, the algorithm plays only that candidate. When there are multiple candidates, the algorithm explores by comparing them all. The estimates of pairwise winning probabilities are based on empirical counts of wins and losses.

The algorithm maintains two types of UCBs. $U_t(ij)$ is the optimistic estimate of arm a_i winning in a tournament with arm a_j and it is analogous to that used in Zoghi *et al.* (2014b) (Section 5.2):

$$U_t(ij) = \frac{g_t(ij)}{N_t(ij)} + \sqrt{\frac{\alpha \ln t}{N_t(ij)}},$$

where $N_t(ij)$ is the number of times up to round t that i and j were compared with each other, $g_t(ij)$ is the number of times that arm i beat arm j, and the parameter α controls the width of the UCB. An additional UCB $V_t(ij)$ with wider upper bound is maintained with the aim of increasing parallel exploration:

$$V_t(ij) = \frac{g_t(ij)}{N_t(ij)} + \sqrt{\frac{\beta\alpha \ln t}{N_t(ij)}},$$

where the parameter $\beta \geq 1$ controls how much wider it is than the UCB of $U_t(ij)$. When there is more than one candidate for a winner according to the narrow confidence bounds U, an exploration round is triggered and arms that could be winner candidates according to the wide confidence bounds V are compared. In both cases, potential winners are arms i for which $U_t(i) \geq \frac{1}{2}$, $V_t(i) \geq \frac{1}{2}$. At each iteration t, if there is only a single potential winner according to U, then this arm is selected. If there are several potential winners, then all the arms $V_t(i) \geq \frac{1}{2}$ are selected for comparison.

The multi-dueling bandit algorithm was compared against RUCB (Zoghi *et al.*, 2014b) and mergeUCB (Zoghi *et al.*, 2015b) using the

Microsoft Learning to Rank dataset, the Yandex Internet Mathematics 2009 dataset and two of the Yahoo! Learning to Rank Challenge datasets. Probabilistic interleave (Hofmann *et al.*, 2011a) was used for RUCB and mergeUCB, and Sample Only Scored Multileave (Brost *et al.*, 2016a) was used for multi-dueling bandit. In terms of cumulative regret, multi-dueling bandit substantially outperforms the two baselines by almost 2 orders of magnitude. Additionally, as the number of rankers increases the cumulative regret increases for RUCB and mergeRUCB, while it is almost independent of the number of rankers for the multi-dueling bandit.

The SelfSparring approach (Sui *et al.*, 2017) reduces the multi-dueling bandits problem to a conventional bandit setting, which is similar to the approach to reducing the dueling bandit problem to a MAB problem (Ailon *et al.*, 2014). SelfSparring uses a stochastic MAB algorithm to independently sample a set of K arms to duel. A prior distribution is used to initialize the sampling process. The preference feedback can be any type of comparisons ranging from full comparison over the K arms (a full preference matrix for all pairs) to single comparison of one pair (just two valid entries in the preference matrix). After the feedback, the posterior distribution over arms is updated. Two algorithms to implement SelfSparring are proposed: IndSelfSparring assumes independent arms and is based on Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1), while KernelSelfSparring uses Gaussian processes (Srinivas *et al.*, 2010) (Section 6.1.1) to make predictions about preference function f based on noisy evaluations over comparisons.

Following the simulation setting of Brost *et al.* (2016b) on the MSLR dataset (described above in this section), IndSelfSparring was compared against the multi-dueling bandit algorithm. The results, in terms of cumulative regret, show that IndSelfSparring significantly outperforms the baseline.

5.5 Pooling Based Evaluation and Bandits

Relevance judgements are used for creating test collections that are later employed for evaluation of IR tasks and techniques. However, relevance

judgements are produced by human assessors, which makes them slow and expensive to collect. For this reason, many test collections are built through pooling, where only a subset of a given document dataset is judged for each topic. In this approach, the more relevant documents are presented to the assessors, the more reliable the evaluation is.

Multi-armed bandits provide a way to model document adjudication in pooling-based evaluation to ensure that the assessor spends most of their time assessing relevant documents (Losada *et al.*, 2016). Given a query and a set of runs, each run can be seen as a bandit arm that can be played to obtain a document to be judged. The reward is binary relevance of the document with respect to the query. Playing an arm means getting the next ranked document from a given run starting from rank 1. Additionally, the judgement ordering problem is non-stationary because the quality of the runs changes when moving down in the rankings and so the probabilities of relevance of the runs change. However, stationary bandit algorithms may concentrate too much on runs with old wins, leading to suboptimal solutions. One way to tackle this problem is to use a weighted average of the past rewards and the last reward. This can be incorporated into a Bayesian bandit model, such as Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1). The parameters of the posterior distribution of a given run i are: $\alpha = 1 + jrel_i$ and $\beta = 1 + jret_i - jrel_i$, where $jrel_i$ is the number of judged documents that are relevant and were retrieved by run i, and $jret_i$ is the number of judged documents that were retrieved by run i. Given the binary relevance of the last document judged, rel_l, the parameters of the runs retrieving this document are updated as: $jrel_i = rate \cdot jrel_i + rel_l$ and $jret_i = rate \cdot jret_i + 1$. If $rate = 1$ this is the standard method, where all rewards count the same. If $rate > 1$, the method gives more weight to early relevant documents and if $rate < 1$, more weight is given to the last relevant document found. The best performing setting was when $rate = 0$.

The method was compared against stationary Thompson sampling and *MoveToFront* (Cormack *et al.*, 1998) – a popular pooling method based on rank biased precision, using a number of TREC datasets. The average number of relevant documents found at different number of judgments performed was the highest in almost all the cases. The

performance of non-stationary Thompson sampling only started to be outperformed by other methods when the number of judgments reached 2000. Stationary Bayesian methods and *MoveToFront* achieved very similar performance to each other.

5.6 Summary

The main contribution of this chapter is the introduction of the dueling bandit problem that, in conjunction with various interleave methods, gave rise to a number of techniques for online ranker evaluation. The initial dueling bandit problem proposed by Yue and Joachims (2009) was further developed into numerous algorithms, where some of the initial assumptions were relaxed or additional information incorporated to capture more real-life aspects of user's behaviour in an information retrieval setting. For example, *Beat-the-Mean* bandit (Yue and Joachims, 2011) relaxes the transitivity assumptions related to user's preferences, while *Copeland Dueling Bandits* (Zoghi *et al.*, 2015a) break the assumption that there exists an arm that beats all the others with probability higher than 1/2. Other approaches combine game theoretic principles (Dudık *et al.*, 2015) or UCB algorithms (Zoghi *et al.*, 2014a) with the dueling bandit mechanism. Further extensions of the dueling bandit problem include generalizations to an unrestricted number of rankers (Section 5.4). In general, experimental results indicate that often the relaxation of the theoretical assumptions leads to algorithms that better reflect real-life user behaviour and produce improved experimental results.

6

Recommendation

This chapter covers the application of bandit algorithms in recommender systems. Recommender Systems are defined as software tools and techniques that provide suggestions for items that are most likely to be of interest to a particular user (RicciLior *et al.*, 2001). Bandit algorithms are a popular method used in recommendation to improve personalization (Section 6.1), often through exploiting the graph structure of a social network (Section 6.2.1) or through the application of collaborative filtering (Section 6.3). Optimization is another area of the application of bandit algorithms in recommender systems: faster learning of relevant features (Section 6.4), detecting changes in user's interest (Section 6.5) or enabling feedback to multiple items simultaneously (Section 6.6).

6.1 Personlization and the Cold Start Problem

A crucial aspect of a recommender system is the ability to suggest relevant items to users based on their background (e.g. age, gender, level of education) or previous queries/purchased items. In this section, we provide an overview of the application of contextual bandits in recommendation (Section 6.1.1) as well as a number of other approaches

used in personalization, such as ranked bandits (Section 6.1.3) and Thompson sampling (Section 2.2.1).

6.1.1 Contextual Bandits

The query-ad-clustering algorithm (Lu *et al.*, 2009) is a version of a contextual multi-armed bandit where the context comes from a metric space. The algorithm proceeds in phases $i = 0, 1, 2, \ldots$ consisting of 2^i rounds each. At the beginning of a phase, the algorithm partitions the query space S into disjoint sets (clusters) $S_1, S_2, \ldots, S_n$ each of diameter $diam$ at most $diam = 2 - \frac{i}{j\cdot + z\cdot}$ and $n = c \cdot 2^{\frac{j\cdot i}{j\cdot + z\cdot}}$, where $j\cdot$ and $z\cdot$ are the covering dimensions of S and c is a constant to ensure that the covering numbers of S are bounded. In each round t of the current phase i, when a query q_t is received, the algorithm determines the cluster S_f of the partition to which q_t belongs. Cluster S_f is fixed in the following way. For each ad, find how many times that ad has been displayed for a query from the set S_f during the current phase up to round t and calculate the corresponding empirical average payoff μ_t of that ad. In round t, the algorithm displays the ad that maximizes the upper confidence index $\mu_{t-1}(ad) + Rad_{t-1}(ad)$, where $Rad_t = \sqrt{\frac{4i}{1+n_t(ad)}}$ is the confidence radius.

However, in many web-based scenarios, the content and its popularity frequently change. Additionally, there are new users with no historical consumption record whatsoever, which makes personalized recommendation more difficult (the so called *cold start* problem). Contextual bandits allow a system to balance the two competing goals: maximizing user satisfaction in the long run, and gathering information about goodness of match between user interests and content. LinUCB (Li *et al.*, 2010a) is a contextual bandit problem that sequentially selects articles to serve users based on contextual information of the user and articles, while simultaneously adapting its article selection strategy based on user click feedback to maximize total user clicks in the long run. The algorithm maintains matrix D_a of dimension $m \times d$ at trial t, whose rows correspond to m contexts that are observed previously for article a, while $C_a \in \mathbb{R}^m$ is the corresponding response vector with click/no-click user feedback. Applying ridge regression to the data (D_a, C_a) gives an

estimate of the coefficients:

$$\hat{\theta}_a = (D_a^T D_a + I_d)^{-1} D_a^T C_a,$$

where I_d is the $d \times d$ identity matrix.

At each trial t, the algorithm selects for presentation the article (arm) with the highest payoff:

$$p_{t,a} = \hat{\theta}_a^T x_{t,a} + \alpha \sqrt{x_{t,a}^T A_a^{-1} x_{t,a}},$$

where $A_a = D_a^T D_a + I_d$, α is a parameter that controls the variance and x is the feature vector of an arm. After observing the reward (click/non-click) for a given $x_{t,a}$, the matrix A and vector C are updated with the new observed datapoint $x_{t,a}$ and its corresponding reward.

The algorithm was tested using around 40 million events from the Yahoo! Today module. Overall, there was more than 10% uplift in CTR with LinUCB compared to ϵ-greedy and a smaller uplift of around 3% compared to UCB (Auer *et al.*, 2002a).

Latent Contextual Bandits (LCB) algorithm (Zhou and Brunskill, 2016) learns the set of latent models from prior users and leverages the learned models to make recommendations for new users. The algorithm consists of two phases. In phase one, LCB runs LinUCB on the first j users to collect training data. In phase two, LCB trains/re-trains latent models using the collected data. Latent user classes are modelled using a mixture of linear regressions. A policy is constructed for each learned latent model and then a contextual bandit algorithm adaptively selects across the policies for a new task. In LCB, the policy set is often smaller than the set of policies considered by generic contextual bandit approaches. Additionally, LCB automatically constructs the set of policies instead of relying on an oracle or expert to provide a good set. More precisely, LCB constructs one policy for each learned latent model and then runs a preselected contextual bandit algorithm that takes in the set of n learned policies for the n latent contextual bandit tasks.

The algorithm was tested using the Yahoo! news dataset. Thompson sampling (Chapelle and Li, 2011) was used with LCB. With 15 and 30 latent models, LCB improved the CTR by about 5% and 10%, respectively, compared to LinUCB.

The contextual combinatorial bandit problem (Qin *et al.*, 2014) is another approach to incorporate prior knowledge into the algorithm. The set $\mathcal{S}_t \subseteq 2^{[k]}$ is the set of all possible subsets of arms at round t. Each set of arms $S_t \in \mathcal{S}_t$ is called a super arm. At each round t, the user observes k feature vectors corresponding to k arms. Then, the user chooses one super arm to play and observes the payoff for each arm in S_t. The payoff of each arm i is:

$$\mu_i = \theta^T x_i + \epsilon_i,$$

where θ is a parameter unknown to the user and the noise ϵ is a zero-mean random variable.

C^2UCB is an efficient algorithm for the contextual combinatorial bandit problem. The basic idea is to maintain a confidence set for the true parameter θ. For each round t, the confidence set is constructed from feature vectors $x_1, \ldots, x_{t-1}$ and observed payoffs of selected arms $\{\mu_i(1)\}_{i \in S_1}, \ldots, \{\mu_i(t-1)\}_{i \in S_{t-1}}$ from previous rounds. Using this confidence set of the parameter θ and feature vectors of arms X_t, the algorithm computes an upper confidence bound for each payoff $\{\mu_1(t), \ldots, \mu_k(t)\}$. The upper confidence bounds and feature vectors of arms X_t are then passed on to the oracle. The algorithm plays the super arm returned by the oracle and uses the observed payoffs to adjust the confidence sets. Thus, $\hat{\theta}_t = V_{t-1}^{-1} C_{t-1}$, where $V_t = V_{t-1} + \sum_{i \in S_t} x_i(t) x_i(t)^T$ and $C_t = C_{t-1} + \sum_{i \in S_t} \mu_i(t) x_i(t)$.

The algorithm was compared against LinUCB using the MovieLens dataset. In terms of precision, the C^2UCB algorithm performs marginally better than LinUCB – the average distance between the two algorithms is around 3%.

The CGPrank algorithm (Vanchinathan *et al.*, 2014) exploits prior information specified in terms of a Gaussian process kernel function, which allows to share feedback in three ways: between positions in a list, between items, and between contexts. Given a context, selecting an optimal ranked list of k recommendations is difficult due to the combinatorial number of choices. The problem becomes more tractable under the assumptions that the reward of a list decomposes additively, and that the reward factors into a position dependent effect p_i independent of the item and a relevance effect that is position independent.

It is possible to normalize the feedback received by an item across all positions that it has been shown at up to time t. Thus, given context x_t, if reward $\rho_i(t)$ is observed for some item a shown in position i, $\rho_i(t)/p_i$ provides an unbiased estimate of relevance $\hat{\mu}(a, x_t)$. Consequently, an unbiased estimate for the reward obtained when showing a in position j instead is given by $\rho_i(t)p_j/p_i$. In this way, the feedback can be shared across positions.

In order to generalize feedback across items/contexts, the prior information is presented in terms of a positive definite kernel function κ, where for two item/context pairs (a, x) and (a', x'), the kernel $\kappa((a, x), (a', x'))$ represents the assumptions about the similarity of expected rewards when presenting item a in context x as opposed to presenting item a' in context x'. The reward function f can be represented as a linear combination $f(a, x) = \sum_j \alpha_j \kappa((a, x)(a_j, x_j))$. Capturing similarity via kernels allows interpreting the relevance function f as a sample from a Gaussian Process (GP) prior, with covariance (or kernel) function κ. Consequently, the relevance is a collection of normally distributed random variables, one for each item/context pair. They are jointly distributed in a dependent manner via the kernel. This joint distribution can be used to make predictions about unobserved item/context pairs via inference in the GP model. For a new item/context pair (a, x), its predictive distribution for $f(a, x)$ is Gaussian with mean μ and variance σ:

$$\mu_t(a, x) = \kappa_t(a, x)^T (K_t + I)^{-1} f_t$$
$$\sigma_t^2(a, x) = \kappa((a, x), (a, x)) - k_t(a, x)^T (K_t + I)^{-1} k_t(a, x),$$

where $k_t(a, x) = [\kappa((a_1, x_1), (s, z)), \ldots, \kappa((a_t, x_t), (a, x))]^T$ and K_t is the positive semi-definite kernel matrix such that

$$K_{t,i,j} = [\kappa((a_i, x_i), (a_j, x_j))]$$

The balance between exploration and exploitation is achieved through linearly trading off the relative importance of the predictive mean and the predictive variance to score each candidate item, i.e. select the item a that maximizes, for the current context x_t the objective UCB_{a,x_t}:

$$UCB_{a,x_t} = \mu_{t-1}(a, x_t) + \beta_{t,a}^{1/2} \sigma_{t-1}(a, x_t),$$

where $\beta_{t,a}$ is the trade-off factor.

CGPrank was tested using the Yahoo! News dataset and Google e-books recommendation. In various settings of the algorithm, CGPrank consistently outperformed the benchmark of UCB1 (Auer *et al.*, 2002a) (Section 2.2) in terms of CTR.

6.1.2 Other Approaches to Personalization

Thompson sampling (Thompson, 1933) has also been applied in the area of ad recommendation.

The algorithm described in detail in Section 2.2.1 was tested in the context of ad displays in Yahoo! (Chapelle and Li, 2011). Thompson sampling achieves the best regret compared to LinUCB (Li *et al.*, 2010a) (Section 6.1.1 above) and ϵ-greedy (Sutton and Barto, 1998). The modified version of Thompson sampling with $\alpha = 0.5$ gives slightly better results than the standard version with $\alpha = 1$. In additional tests with news recommendation on the Yahoo! portal, Thompson sampling outperformed the baselines in terms of CTR.

Tang *et al.* (2015) propose a parameter-free bandit strategy, which employs online bootstrap, to derive the distribution of estimated models in an online manner. The bootstrap is a method to derive the distribution of an estimator by data resampling. Instead of specifying a model for data generating, it only uses the information from the observed data. Observations up to time t are stored in $\mathcal{D}$, while $\mathcal{D}_i$ denotes observations only from pulling arm a_i. When any $\mathcal{D}_i$ is small (fewer than 30 observations), then a random arm is selected. When all $\mathcal{D}_i$ are sufficient large, then the offline bootstrap based contextual bandit algorithm has the following steps: (1) for each arm a_i randomly sample n_i observations from $\mathcal{D}_i$ D with replacement and estimate θ_{a_i}, the coefficient vector of the reward prediction model for arm a_i based on maximum likelihood estimation; (2) pull the arm with the highest estimated payoff; (3) receive the reward ρ_{t+1} after pulling the arm. If the steps 1 and step 2 are repeated many times, then there is a collection of bootstrap replications of $\hat{\theta}_{a_i}$, which approximately represent the sampling distribution of $\hat{\theta}_{a_i}$.

In online bootstrap, a random variable $P_j \sim Poisson(1)$ is generated representing the proportion of times the j-th observation is picked to

a bootstrap sample in the offline setting. Then, in the online setting, when the j-th observation is received, the algorithm knows how many times this observation should appear in a bootstrap sample. The online learning algorithm is invoked P_j times to learn j-th observation P_j times. Then, the learned model is approximated to the model that learns observations in a bootstrap sample offline. Stochastic gradient ascent algorithm is used for updating the estimation of each bootstrap replication θ_{a_i}.

The online bootstrap bandit was tested using the Yahoo! news dataset. In a cold-start scenario, it achieved significantly higher CTR compared to LinUCB (Li *et al.*, 2010a) and Thompson sampling. However, its performance was only marginally higher compared to ϵ-greedy and to a version of Thompson sampling that only maximizes the likelihood using stochastic gradient ascent and ignores the regularization from the prior. In general, the performance of the baseline algorithms was greatly affected by the initial parameter setting, which does not happen in the bootstrap bandit.

6.1.3 Improving Cold Start Recommendations

Caron and Bhagat (2013) model the learning of preferences of coldstart users using multi-armed bandits embedded in a social network. Each user u is associated with a multi-armed bandit whose arms correspond to items to recommend. The bandit strategy selects an item to recommend to user u. The reward is 1 if u accepts the recommendation and 0 otherwise, with the reward modelled as a random variable following a Bernoulli distribution.

Two strategies are proposed to leverage neighborhood estimates to improve the learning rate of bandits for cold-start users. The MixPair strategy combines estimates from pairs of neighboring bandits by extending the UCB1 algorithm (Auer *et al.*, 2002a). MixPair considers single edges (u_i, u_j) of the social graph, where user u_i is a cold-start user and the system has some information about user u_j. Next, upper confidence bounds are computed based on all samples seen so far both from MABs associated with u_i and u_j. To aggregate data from the whole neighborhood, the neighbor u_j is re-sampled from the neighborhood u_i

at the beginning of each step t. Two sampling strategies were considered: uniform sampling, based on the assumption that all neighbors are equally similar to user u_i; multi-armed bandit defined on neighbors u_j sampled from the neighborhood of u_i to learn the most similar ones online.

The second strategy, MixNeigh, is a heuristic based on consensus in the neighborhood of a user. The strategy first aggregates reward estimates from all neighbors, and then chooses between this aggregate and the user's empirical estimate using a heuristic based on confidence radii. This strategy interpolates between using the neighborhood estimate as a prior and exploiting the empirical estimate when its precision is good enough. Experimental results using the Last.fm dataset show that MixNeigh is the best strategy for the cold start problem compared to MixPair and UCB1.

Independent Bandit Algorithm (IBA) (Kohli *et al.*, 2013) also aims to support recommendation in scenarios where there is no prior information about the quality of content in the system. The algorithm is based on Ranked Bandit Algorithm (RBA) (Radlinski *et al.*, 2008a) (Section 4.1). It uses several instances of a multi-armed bandit algorithm working independently to recommend a set of articles. This assumption allows for a faster learning rate than online algorithms based on the diversity principle – as is the case with RBA. The main difference between the independent and the ranked bandit algorithm is the feedback. IBA gives a reward of 1 to any article that was clicked while RBA only gives a reward of 1 to the first article that was clicked. The independence between bandit instances in IBA allows learning to happen in parallel as opposed to the sequential learning in RBA, which allows for a quicker convergence of the independent solution.

The algorithm was tested using the MovieLens dataset and compared against RBA. Two algorithms were employed with IBA and RBA, namely UCB1 (Auer *et al.*, 2002a) and ϵ-greedy (Sutton and Barto, 1998) (Section 2.2). The performance of Independent-ϵ-Greedy and Independent-UCB were roughly the same. In half of the experiments, either Independent-ϵ-Greedy or Independent-UCB perform better than Ranked-ϵ-Greedy. In experiments where Ranked-ϵ-Greedy performs

better than the independent solutions, it only begins to perform better after 10000 time steps.

DynUCB (Nguyen and Lauw, 2014) is a bandit algorithm that divides the population of users into multiple clusters and customize the bandits for each cluster, although the algorithm does not involve a graph structure. This clustering is dynamic, i.e., users can switch from one cluster to another as their preferences change. The algorithm starts with k random clusters, which are refined over iterations. DynUCB still maintains n bandits for n users. For each user u_i, its coefficient w_i is learned based on matrix M with the set of items selected by the user in the past and the corresponding reward/click vector C_i. The bandits in the same cluster collaborate with one another so that when generating a recommendation for user u_i at time t, the estimation of expected reward for each arm a is based on cluster level coefficient w_k, learned from cluster level parameters C_k and M_k derived from the parameters C_i and M_i of each user u_i in a given cluster. Thus, each user benefits from the feedback provided by other users in the same cluster. At each iteration, users are reassigned to the cluster whose coefficient w_k is closest to their representation of w_{i_t} – a practice reminiscent of the k-means clustering algorithm. This dynamic reassignment of clusters allows DynUCB to adapt to changing contexts and user preferences over time.

DynUCB was tested using the *Delicious* dataset and the LastFM dataset. At $k = 16$, DynUCB has higher cumulative rewards than LinUCB (Li *et al.*, 2010b), however, only in the long run with $t > 20000$. In the short run (for $t < 20000$), LinUCB tends to have a higher cumulative reward.

6.2 Social Networks and Recommender Systems

Incorporating social components, such as network of relationships between users, into bandit algorithms can also improve the quality of recommendations. These social relationships can be explicitly encoded in a graph, where adjacent nodes/users are deemed similar to one another. When there are many users, it is often possible to identify subgroups where users share interests, thereby facilitating targeting users by means

of group recommendations. Thus, instead of learning a different model for each user, the system only needs to learn a single model for each group.

6.2.1 Graph Clustering with Bandits

The Cluster of Bandits (CLUB) algorithm (Gentile *et al.*, 2014) aims to capture the similarities between neighbouring users in a graph structure. There are n parameter vectors $u_1, u_2, \ldots, u_n$, one per node (user), where nodes within the same cluster V_j share the same vector. The algorithm maintains at time t an estimate w_{i_t} for vector u_i associated with user $i \in V$. Vectors w_{i_t} are updated based on the payoff operating on the context vectors. Every user in V is associated with a linear bandit algorithm[1]. The prototype vector w_{i_t} is the result of a standard linear least squares approximation to the corresponding unknown parameter vector u_i. In particular, $w_{i_{t-1}}$ is defined through the inverse correlation matrix $M^{-1}_{i_{t-1}}$ and the additively updated vector $C_{i_{t-1}}$ that incorporates user feedback. Additionally, the algorithm maintains at time t an undirected graph G_t, whose nodes are the users in V. The algorithm starts off from the complete graph, and progressively erases edges based on the evolution of vectors w_{i_t}. The graph encodes the current partition of V by means of the connected components of G_t. At each iteration t, the algorithm receives the index i_t of the user u_{i_t} to serve along with the associated context vectors, and must select one among them. First, the algorithm determines which cluster V, node i_t belongs to. Next, it builds the aggregate weight vector w for that cluster by taking the prior of the context vectors associated with i and computing the least squares approximation as if all nodes $i \in V$ have been collapsed into one. This weight vector is then used to select an appropriate context for the user. Once this selection is done and the associated payoff associated with the context vector is observed, the matrix with prior observations and feedback vector are updated. Although the update is only performed at node i_t, it will also implicitly update the aggregate weight vector w associated with a given cluster V. Finally, the cluster structure may

[1] See Cesa-Bianchi *et al.* (2013) for more theoretical details of assigning a linear bandit to each user in node clustering.

be modified – the algorithm compares the distance between the weight vectors of all existing edges. If the distance between a node and the remaining nodes within the same cluster is large, then that node is moved to a different cluster and the graph is repartitioned.

CLUB was tested using the Yahoo! news dataset. It achieved higher CTR compared to LinUCB (Li *et al.*, 2010a) (Section 6.1.1).

The Collaborative Linear Bandit algorithm (CoLin) (Wu *et al.*, 2016) also models collaboration between bandits based on graph. However, while in previous graph based models connected users in the graph were assumed to have similar underlying bandit parameters with rewards independent across users, in CoLin neighboring users do not have to share similar bandit parameters but they generate influence over their neighbors' decisions. The "collaboration" among bandits is incorporated in global estimation of the parameter θ, which is derived from matrix M_t with context vectors and vector C_t incorporating obtained payoffs, which store global information shared among all the bandits in the graph. More specifically, the context vector and payoff observed for a given user at time t are propagated through the whole graph via the relational matrix W. It is derived from projected context vectors on every user and sparse vectors with observations of active users only. Due to this information sharing, although some users have not generated any observation yet (i.e., cold start), they can already start from a non-random initialization of their bandit parameters θ. When W is an identity matrix, i.e., users have no influence on each other, the estimation of θ reduces to independently computing n different θs.

CoLin was tested with the Yahoo! news dataset. CoLin outperformed the LinUCB (Li *et al.*, 2010a) baseline. It initially it performed worse than CLUB in terms of CTR, however, its performance improved over iterations. As some popular news articles became out-of-date, CLUB cannot correctly recognize their decreased popularity, and thus provides worse recommendations than CoLin resulting in decreased CTR over time.

COFIBA (Li *et al.*, 2016b) is a graph clustering algorithm that only applies to the case when the content universe is large but known *a priori*, which is often the case in commercial recommender systems. COFIBA, relies on adaptive clustering of both users and items. COFIBA stores

in round t an estimate $w_{i,t}$ of vector u_i associated with user i. Vectors $w_{i,t}$ are updated based on the payoff feedback with a standard linear least-squares approximation to the corresponding u_i. Every user i holds a linear bandit algorithm that operates on the available content through the inverse correlation matrix M and the feedback vectro C subject to additive updates. Based on the local information encoded in the weight vectors $w_{i,t-1}$ and the confidence bounds, the algorithm also maintains a family of clusterings of the set of users, and a single clustering over the set of items.

On both sides, such clusterings are represented through connected components of undirected graphs (as in Gentile *et al.* (2014) discussed at the beginning of this section), where nodes are either users or items. At time t, COFIBA receives the index it of the current user to serve, along with the available item vectors $x_{t,1}, \ldots, x_{t,l}$ and must select one among them. The algorithm computes neighborhood sets V_k based on the current clustering of users. Set V_k should be regarded as the current approximation to the user cluster it belongs to when the clustering criterion is defined by item $x_{t,k}$. Each neighborhood set then defines a compound weight vector w through the aggregation of the corresponding matrices $M_{i,t-1}$ and vectors $C_{i,t-1}$. After receiving the payoff (and updating M and C), COFIBA updates the clusterings of users and items. In round t, there are multiple graphs G_k at the user side and a single graph at the item side. On both user and item sides, updates take the form of edge deletions. Updates at the user side are only performed on the graph G the selected items x. Updates at the item side are only made if it is likely that the neighborhoods of the user significantly changed when considered with respect to two previously deemed similar items. Specifically, if item x_h was directly connected to item x_k at the beginning of round t and, as a consequence of edge deletion at the user side, the set of users that are now likely to be close to x_h is no longer the same as the set of users that are likely to be close to x_k, then this is an indication that item x_h is not inducing the same partition over users as x_k, hence the edge (x_k, x_h) gets deleted.

In order to allow the algorithm to scale, instead of starting off with complete graphs over users, each time a new cluster over items is created, the complete graph is sparsified randomly retaining with high

probability the underlying clusterings over users. This works under the assumption that the latent clusters are not too small.

COFIBA was compared against LinUCB, DynUCB and CLUB. In terms of CTR assessed using the Yahoo! dataset, it performed marginally better than LinUCB and CLUB, however it performed substantially better than DynUCB. In two other datasets concerned with CTR on adverts obtained from Telefonica and Avazu, COFIBA outperformed all the benchmark algorithms by between 5% and 20%.

The Context Aware clustering of Bandits algorithm (CAB) (Gentile *et al.*, 2017) incorporates collaborative effects into inference by making changes in the way items are recommended and the weights w are updated. At time t, before recommending an item to the user, CAB first computes for each item the set of users that are likely to give the item a similar payoff. This set is the estimated neighborhood of user u_i with respect to item x_k. Earlier approaches (described above in this section) update the user proxies w_i by solving a regularized least squares problem with feature representations of items presented previously to user i and the corresponding payoffs. CAB, on the other hand, allows a user u_i to inherit updates obtained through an item being presented to another user u_j if the two users agree on their opinion on item x with a sufficiently level of confidence. If CAB is not confident enough about the opinion it has along the direction x_k, then only the weight w_i associated with user u_i is updated. However, if CAB is confident, then the proxy updates are performed for all users in its estimated neighborhood. All users in the neighbourhood undergo the same update. CAB is very flexible in handling a fluid set of users. Due to its context-sensitive user aggregation step, CAB allows users to be easily added or dropped.

In terms of CTR, CAB substantially outperformed the baselines of CLUB, DynUCB and LinUCB when using the Telefonica dataset, the Avazu dataset and the KDD Cup 2012 dataset.

6.2.2 Spectral Bandits

Spectral bandits provide another approach to capturing similarities between users and/or recommended items. Valko *et al.* (2014) study a bandit problem where the payoffs of arms are smooth on a graph. This

framework applies to online learning problems where each recommended item is a node in a graph and its expected rating is similar to its neighbors. The goal is to recommend items that have high expected ratings. In this setting, the arms are orthogonal to each other. This provides additional information across the arms through the estimation of the parameters of the algorithm. The assumption is that there exists a known orthogonal matrix U such that $w = U\mu$ has a low norm, which means that w can be estimated using penalization and then recover μ, where μ is the mean reward. Given a vector of weights w, its Λ norm is defined as:

$$\| w \|_{\Lambda} = \sqrt{\sum_{l=1}^{L} \lambda_l w_l^2} = \sqrt{w^T \Lambda w}$$

The aim is to penalize the coefficients w that correspond to the eigenvectors with the large eigenvalues.

Spectral bandits are implemented through SpectralUCB algorithm, which is based on LinUCB (Li *et al.*, 2010a) (Section 6.1). The algorithm uses regularized least-squares estimate w_t of the form:

$$w_t = \arg\min_{w} \left(\sum_{v=1}^{t} [\langle x_v, w \rangle - \rho_v]^2 + \| w \|_{\Lambda} \right)$$

A key part of the algorithm are the $c_t \| x \|_{V_t^{-1}}$ confidence widths for the prediction of the rewards:

$$c_t = 2R\sqrt{d \log(1 + t/\lambda) + 2 \log(1/\delta)} + const,$$

where R is the regret, λ and δ are regularization and *const* is a constant. The definition of c_t is based on the effective dimension d, which is specifically tailored to the specific setting. Effective dimension is a proxy for the number of relevant dimensions and is defined as the largest d such that $(d-1)\lambda d \leq \frac{T}{\log(1+T/\lambda)}$ The effective dimension d is small when the coefficients λ_i grow rapidly above T.

Another algorithm called SpectralEliminator scales better with d compared to SpectralUCB. The main idea behind the algorithm is to divide the time steps into sets in order to introduce independence and eliminate the arms that are not promising much faster.

SpectralUCB was tested using the MovieLens dataset. SpectralUCB suffers only about a quarter of regret of LinUCB. The regret is further reduced with SpectralEliminator.

Spectral Thompson Sampling (SpectralTS) (Kocák *et al.*, 2014) is an alternative to SpectralUCB based on Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1), which is used to decide which arm to play. The current user preferences are encoded in vector w_t. The algorithm's knowledge about w_t is represented as the normal distribution $\mathcal{N}(\hat{w}(t), c^2 M_t^{-1})$, where $\hat{w}(t)$ is an approximation of the unknown parameter w and $c^2 M_t^{-1}$ reflects the uncertainty about it, and c is the confidence width. At each time step t a sample $\tilde{w}(t)$ is generated from the distribution $\mathcal{N}(\hat{w}(t), c^2 M_t^{-1})$ and an arm is selected that maximizes $x_i \tilde{w}(t)$, where x_i is the context vector of arm a_i. After receiving a reward, the estimate of w is updated. The computational advantage of SpectralTS compared to SpectralUCB is that it does not require computing the confidence bound for each arm. In SpectralTS, it is only necessary to sample $\tilde{w}$, while SpectralUCB requires computing a M_t^{-1} norm for each of n feature vectors.

The experimental evaluation using the MovieLens dataset show that the performance of SpectralTS is similar or slightly better than that of SpectralUCB in terms of cumulative regret.

6.3 Collaborative Filtering and Matrix Factorization

Recommender systems often rely on past user behavior, e.g. previous transactions or product ratings, without requiring the creation of explicit profiles. This approach is known as *collaborative filtering* (Koren *et al.*, 2009). Collaborative filtering analyzes relationships between users and interdependencies among products to identify the relationships between new users and recommended items. Latent factor models are an alternative approach that makes recommendation by characterizing both items and users on a number of factors inferred from the users' ratings patterns. Matrix factorization is the most popular realization of latent factor models. In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns.

Bresler *et al.* (2014) consider a recommender system with n users and m items. At each time step, each user is recommended an item that they have not consumed before. The user rates the item $+1$ (like) or -1 (dislike). The reward earned by the system is the total number of liked items across all users. The latent preferences for user u_j are represented by vector $x_j \in [0,1]^m$. User u_j likes item i with probability $p_{j,i}$, where item i is likable if $p_{j,i} > 1/2$ and unlikable if $p_{j,i} < 1/2$. The aim is to maximize the expected number of likable items recommended to the user, where the likable items can be recommended in any order. Ratings obtained from other (similar) users are used to recommend new items to user u_i. The assumption is that there are k different types of users, where users of the same type have identical item preference vectors. A user belongs to each user type with probability $1/k$.

Collaborative-Greedy implements the above concept. The algorithm is similar to the ϵ-greedy algorithm (Sutton and Barto, 1998) with the restriction that items cannot be recommended multiple times to the same user[2]. In Collaborative-Greedy, the greedy choice, or exploitation, uses the cosine-similarity measure. The exploration, on the other hand, is split into two types, a standard item exploration in which a user is recommended an item that they have not consumed yet uniformly at random, and a joint exploration in which all users are asked to provide a rating for the next item in a shared, randomly chosen sequence of items. This helps the algorithm to learn about similarity between users. The probability of random exploration, which is decaying with the number of users, is set to $1/n^\alpha$ for some pre-specified value of α. The probability of joint exploration is set to $1/t^\alpha$ and decaying with time.

Nakamura (2014) proposes an approach to collaborative filtering based on UCB (Auer *et al.*, 2002a). The aim is to solve a direct mail problem in which a system everyday selects a set of user-item pairs, sends a recommendation mail of that item to the user and receives a response from the user. Each user/item pair can be selected at most once during the period. In this model, the only information used for

[2]See also Deshpande and Montanari (2012) for an analysis of linear bandits in data poor (high dimensional) regime with specific reference to recommender systems. In the pre-processing step, offline collaborative filtering through matrix completion is solved to compute feature vectors for items.

learning user's preference is user's response and no feature of users and contents is used. User's feedback to the recommendation takes the form of click or purchase that can be converted into a rating. The objective is maximization of the sum of positive ratings given to the recommended items. The total number of recommendations is assumed to be vary small compared to the number of all the user/item pairs. The UCB-based algorithm deterministically selects user/item pairs using an index which depends on both the covariance matrices of the posterior distributions of latent user and item vectors. In order to use the bandit collaborative strategy, two methods to obtain approximations to these covariance matrices are proposed. The first approach applies variational Bayes (VB) to stochastic matrix factorization (Lim and Teh, 2007), while the second approach uses probabilistic matrix factorization (PMF) (Mnih and Salakhutdinov, 2008). These two approaches gave rise to two UCB strategies: UCB-VB and UCB-PMF.

Experimental results based on synthetic data as well as a number of small datasets show that UCB-VB and UCB-PMF significantly outperform their non UCB counterparts in terms of cumulative user rating.

FactorUCB (Wang *et al.*, 2017a) is another approach to matrix factorization based on bandits. Two modifications compared to a traditional bandit approach are introduced. First, to reduce the reward prediction uncertainty on new items, observable contextual features are introduced into the estimation of latent item factors. Second, mutual influence among users is incorporated to reduce the reward prediction uncertainty on new users. The dependency among users is encoded directly into the reward generation assumption for matrix factorization. The observed reward from each user is determined by a mixture of neighboring users. Thus, users are placed on a weighted graph G encoding the affinity relation among users, which allows the performance of the estimation across them simultaneously. Each node $i \in G$ is parameterized by the latent user factor θ_i for user u_i and each edge represents the influence across users in reward generation. This graph is represented as an $n \times n$ stochastic matrix W, in which each element is proportional to the influence that user u_j has on user u_i in determining the reward of different items. W is column-wise normalized.

The reward generation assumptions are enhanced as follows:

$$\rho_{a_t,i} = x_{a_t}^T \Theta^x w_i + v_{a_t}^T \Theta^v w_i + \eta_t,$$

where x_a is a contextual feature of item a, v_a is the vector with latent factors for item a, Θ represents latent user factors decomposed into two separate sub-matrices corresponding to the observed context features x and latent factors v. Thus, not only the observed contextual features but also the estimated latent factors are propagated through the user graph to determine the expected reward of items across users. Such information sharing greatly reduces sample complexity in learning the latent factors for both users and items.

Experimental results using the Yahoo! news dataset and the Last.fm dataset show that factorUCB improves CTR compared to graph based bandit algorithms that do not employ laten factors, such as CoLin and CLUB (Section 6.2.1 for more details).

Bayesian bandit approaches have also been applied to matrix factorization[3]. *Particle Thompson sampling for matrix factorization* (PTS) (Kawale *et al.*, 2015) is a method based on Thompson sampling (Section 2.2.1) augmented with an online Bayesian probabilistic matrix factorization method based on the Rao-Blackwellized particle filter. In PTS, it is assumed that ratings come from the normal distribution with a fixed but unknown latent features (W, V), representing the user and item latent features, respectively. The user rates the recommended items and the system receives this rating as a reward. The system recommends item j at time t using a policy based on the past ratings. The highest expected reward the system can earn at time t is $\max_j W_i^T V_j$, which is achieved if the optimal item $j = \arg\max_j W_i^T V_j$ is recommended.

The main challenge when using Thompson sampling for matrix factorization bandit is the incremental update of the posterior of the latent features (W, V). An efficient Rao-Blackwellized particle filter (RBPF) is used to exploit the structure of the probabilistic matrix factorization model. Since the particle filter needs to estimate a set of

[3]See also Zhao *et al.* (2013) where an interactive collaborative framework is introduced. The framework uses Thompson sampling and several UCB algorithms to illustrate how exploration and exploitation can be balanced in this interactive framework.

non-time-varying parameters, a crucial aspect of RBPF is an effective and efficient MCMC-kernel $K_t(V^{'}, \sigma_W^{'}; V, \sigma_W)$ stationary with respect to the posterior distribution. The design of the move kernel are based on two observations. First, W and σ_V can be used as auxiliary variables, effectively sampling $(W, \sigma_V \mid V, \sigma_W)$ and then $(V^{'}, \sigma_W^{'} \mid W, \sigma_v^{'})$. This move, however, can be highly inefficient due to the number of variables that need to be sampled at each update. The second observation is the key to an efficient implementation: latent features for all users except the current user are independent of the current observed rating. Thus, at time t, it is sufficient to only resample the latent features for the current user. Furthermore, it is also sufficient to resample the latent feature of the current item.

In experimental results using a number of real-life datasets, such as MovieLens and Yahoo Music, PTS obtains substantially lower regret compared to probabilistic matrix factorization (Mnih and Salakhutdinov, 2008).

6.4 Feature Learning with Bandits

This section provides an overview of the application bandits to learn the most relevant features for personalized recommendation,

6.4.1 User Specific Features

Traditionally, bandit algorithm utilize the entire feature space when learning a user model but in certain applications dimensionality reduction can used to limit the size of a large feature space in order to speed up convergence. However, thus obtained user models capture mostly the characteristics of a typical user and may fail to learn the preferences of a user that deviates from stereotypical users. The challenge is to be able to leveraging prior knowledge to reduce the cost of exploration for new users, while maintaining the representational power of the full feature space to cater to users' personal preferences. Yue *et al.* (2012b) propose a coarse-to-fine hierarchical approach for encoding prior knowledge. While, a coarse, low-rank subspace of the full feature space is sufficient to accurately learn a stereotypical user's preferences, the coarse-to-fine

feature hierarchy allows exploration in the full space when a user is not perfectly modeled by the coarse space. CoFineUCB is the algorithm that automatically balances exploration within the coarse-to-fine feature hierarchy.

For example, there are features that correspond to interest in articles about baseball and cricket. Prior knowledge suggests that users are typically interested in one or the other. Then, a feature subspace can be designed where baseball and cricket topics project along opposite directions in a single dimension. A bandit algorithm can leverage this structure by first exploring at a coarse level to determine whether the user is more interested in articles about baseball or cricket. This approach can be formalized as a hierarchy that is composed of the full feature space and a subspace, where matrix $X \in \Re^{d \times n}$ denotes a n-dimensional subspace, which is constructed by using the top n singular vectors of W containing the user profiles.

The user's preferences are denoted by weight vector $w = X\bar{w} + w_\perp$, where $\bar{w}$ is the projected user profile and $w_\perp$ is the residual, or orthogonal component, of w with respect to X. The principle can be extended to multiple hierarchies:

$$w = X_1(X_2(\dots(X_i w_i + w_{i-1,\perp})\dots w_{1,\perp}) + w_\perp.$$

The CoFineUCB algorithm generalizes the LinUCB algorithm (Li *et al.*, 2010a) (Section 6.1), and automatically trades off between exploring the coarse and full feature spaces. At each iteration t, CoFineUCB estimates the user's preferences in the subspace $\tilde{w}_t$, as well as the full feature space w_t. Both estimates are solved via regularized least-squares regression:

$$\tilde{w}_t = \arg\min \sum_{i=1}^{t-1} (\tilde{w}^\top \tilde{x}_i - \tilde{C}_i)^2 + \tilde{\lambda} \parallel \tilde{w} \parallel^2,$$

where $\tilde{x}_i = X^\top x_i$ denotes the projected features of the action taken at time i, C is the user feedback, and λ is the regularization parameter. Then, w_t is estimated:

$$w_t = \arg\min \sum_{i=1}^{t-1} (\tilde{w}^\top x_i - C_i)^2 + \lambda \parallel w - X\bar{w}_t \parallel^2,$$

which regularizes w_t to the projection of $\bar{w}_t$ back into the full space. CoFineUCB chooses then the item with the largest potential reward to present to the user: $x_t = \arg\max w_t^\top + c_t(x) + \tilde{c}_t(x)$, where $c_t(\cdot)$ and $\tilde{c}_t(\cdot)$ indicate the full space and the subspace, respectively.

CoFineUCB was compared to LinUCB (Li *et al.*, 2010a) in the context of personalized news recommendation. CoFineUCB dramatically outperforms LinUCB applied to the full data, while its performance is comparable to LinUCB applied only to the subspace of the data. Although the experimental results show the benefit of exploring in the subspace, they also demonstrate that for atypical users, the subspace is not sufficient to adequately learn their preferences resulting in linear regret when only the subspace is used.

6.4.2 Feature Clustering

The first approach to learning an optimal online matching between two feature spaces relied on taxonomies (Pandey *et al.*, 2007). In the particular problem tackled in Pandey *et al.* (2007) pages and ads are classified into page and ad taxonomies, respectively. There are two successive levels of the taxonomies with the lower level nodes of the page and ad taxonomies referred to as page-classes and ad-classes, respectively. All page-classes (ad-classes) that are children of the same parent node from the upper level constitute a page-class group (adc-lass group). There is also a page-ad connection matrix $X = S \times A$, where each cell has a CTR value for the corresponding (page-class X, ad-class A) pair. The goal is to learn matrix X so as to maximize the expected total number of clicks. This is translated into a MAB problem, where for each page-class a bandit is create with the arms given by the ad-classes and the payoff probabilities by the CTR values. However, the arms of each bandit and the bandits themselves are not independent since S and A are partitioned into page-class and ad-class groups. The arms in the same group are likely to have similar payoff probabilities in terms of CTR values. The bandit algorithm runs at two levels: the group level followed by a cell level. The switch between the levels occurs when a given criteria for exploration at the group level occurs. A Bayesian framework reminiscent of Thompson sampling

(Chapelle and Li, 2011) (Section 2.2.1) exploits dependencies among the arms. Experimental results show that, compared to a traditional UCB algorithm with independent arms, the proposed approach achieves a substantially higher cumulative reward.

A similar approach was proposed by Daee *et al.* (2016), where feedback from multiple domains is combined, i.e. retrieved items (documents) and their features (keywords). A probabilistic model is employed to account for the relationship between the two domains. The approach assumes that there is a document-keyword matrix where each entry specifies the likelihood of document a_i being generated by keyword k_j. This matrix is generated from the data model that expresses how keywords and the documents are related, e.g. normalized tf-idf representations of documents generated from bag of words. The user provides relevance feedback to both documents and keywords. The unknown parameter θ defines the shared link between relevance (reward) distributions for documents and keywords. After the user has interacted with a set of documents and a set of keywords, the posterior distribution of θ is updated. Thompson sampling (Chapelle and Li, 2011) is applied to balance the exploration and exploitation when selecting which documents and keywords to present to the user.

Wang *et al.* (2016) describe how to learn the hidden features for contextual bandit algorithms. Hidden features are explicitly introduced in the reward generation assumption, in addition to the observable contextual features. This approach can be particularly useful in collaborative filtering solutions based on latent factor models. Through coupling the observed contextual features and the hidden features, the reward can be formalized as follows:

$$\rho_{a_t,u} = (x_{a_t}, v_{a_t})^T(\theta_u^x, \theta_u^v) + \eta_t,$$

where x_{a_t} and v_{a_t} are the observed and hidden features of item a_t, and θ_u^x and θ_u^v are the corresponding bandit parameters, while $\theta_u = (\theta_u^x, \theta_u^v)$ are the unknown preference parameters of user u. The algorithm assumes that the dimension d of hidden features is known to the learner ahead of time.

Due to the fact that only x_{a_t} is shown to the user, the residual between the true reward and the user's estimate no longer has a zero

mean as assumed in most linear contextual bandit algorithms. Instead, the residual of reward estimation is constantly shifted by $v_{a_t}^T \theta_u^v$. Due to the coupling between θ_u and v_a in the reward generation, a coordinate decent algorithm built on ridge regression is used to estimate the unknown bandit parameter θ_u for each user and the unknown hidden feature v_a for each item.

Experimental results using data from the Yahoo! Today module showed that on average the bandit algorithm with hidden features achieves 0.5 higher CTR compared to LinUCB (Li *et al.*, 2010a). In experiments with Last.fm and Delicious datasets, the proposed algorithm achieved higher payoff compared to LinUCB with the difference getting larger as time progresses. The initial improvement was particularly stark with cold-start users showing the advantage of learning hidden features.

6.5 Recommendations with a Limited Lifespan

In many recommendation scenarios, the recommended items, i.e. products, advertisements or news articles, are only relevant or available for a short period of time. This aspect adds a new dimension to the design of bandit algorithms that can be successfully applied in many recommender systems. In *mortal bandits* (Chakrabarti *et al.*, 2009) arms have (stochastic) lifetime after which they expire. In this setting an algorithm needs to continuously explore new arms, unlike in the standard bandit model where arms are available indefinitely and exploration is reduced once an optimal arm is identified. In the mortal bandits model, at the end of each time step t, one or more ads (arms) may die and be replaced with new ads. There are two assumptions: (1) change happens only through replacement of old ads so the number of ads remains constant; (2) as long as an ad is alive, its payoff is fixed. Death can be modeled in two ways. An ad may have a budget l that is known *a priori* and revealed to the algorithm. The ad then dies immediately after it has been selected l times. This case is referred to as budgeted death. Alternatively, each ad may die with a fixed probability p after every time step, irrespective of whether it was selected. This is equivalent to each ad being allocated a lifetime budget l_i drawn from a geometric distribution with parameter p that is fixed when the arm (ad) is born

but is never revealed to the algorithm. In the second case, called timed death, new arms have an expected lifetime of $l = 1/p$.

The reward function is modelled in two ways. In the state-aware (deterministic reward) case, the reward is $\rho_{i_t} = \mu_{i_t}$, where μ_{i_t} is the payoff of arm a_i at time t. This provides complete information about each ad immediately after it is displayed. This approach is implemented through the DetOpt algorithm, where an arm with the highest payoff so far (among all the active arms) is played until it expires. At that point, a random arm is played and its payoff is compared against that of the remaining active arms to find the best arm.

In the state-oblivious (stochastic reward) case, the reward is a random variable that is 1 with probability μ_{i_t} and 0 otherwise. This approach is implemented through a modified version of DetOpt algorithm. The intuition behind this modification, called *Stochastic*, is that instead of pulling an arm once to determine its payoff μ_i, the algorithm pulls each arm n times and abandons it unless it looks promising. A variant, called *Stochastic with Early Stopping*, abandons the arm earlier if its maximum possible future reward will not justify its retention

Additionally, an epoch-greedy heuristic based on standard MAB is proposed in Chakrabarti *et al.* (2009). The heuristic works in two stages: (1) select a subset of k/c arms uniformly at random from the total k arms at the beginning of each epoch; (2) operate a standard bandit algorithm on these until the epoch ends, and repeat. Step 1 reduces the load on the bandit algorithm, allowing it to explore less and converge faster, in return for finding an arm that is probably optimal only among the k/c subset. The constant c and the epoch length balance the speed of convergence of the bandit algorithm, the arm lifetimes, and the arm payoff distribution. The value of c is chosen empirically. This heuristic, is implemented through an extension of the UCB1 algorithm called UCB1k/c.

The mortal bandit setting requires different performance measures than the ones used with static bandits. In the static setting, very little exploration is needed once an optimal arm is identified and so the quality measure is the total regret over time. In the mortal bandit setting the algorithm needs to continuously explore newly available arms. What is then taken into consideration is the long term, steady-state, mean

regret per time step of various solutions. This regret is defined as the expected payoff of the best currently alive arm minus the payoff actually obtained by the algorithm.

The UCB1k/c algorithm, *Stochastic* and *Stochastic with Early Stopping* were tested using real-life data with the UCB1 algorithm as the baseline. The test data consisted of approximately 300 real ads belonging to a shopping-related category when presented on web pages classified as belonging to the same category. In terms of regret per time step, *Stochastic with Early Stopping* performs best, while the regret of UCB1 remains more or less flat throughout the timestamps.

Komiyama and Qin (2014) formulate a variant of the bandit problem, where new arms are dynamically added into the candidate set and the expected reward of each arm decays as the rounds proceed. A time-decaying MAB goes as follows. At each round t, a system selects one arm a_i from the candidate set A_t and receives a random reward $\rho_{i,t}$. The reward information of the other arms is not available. The reward of arm a_i is drawn from a Bernoulli distribution parameterized by $\rho_{i,t}$ which consists of two parts: the sum of a constant part ρ_i and several basic decaying functions $f_k(t - t_i)$, where $t - t_i$ is the number of rounds since arm a_i appears for the first time. The expected reward of arm a_i at round t can be modeled as $\rho_{i,t} = \rho_i + \sum_{k=1}^{n} w_{i,k} f_k(t - t_i)$, where $w_{i,k}$ is the weight associated with the k-th decaying function for arm a_i. The representation of $\rho_{i,t}$ is $x_{i,t}^T \theta_i$, where $x_{i,t}$ is the context vector for arm a_i at time t and θ_i contains the values of functions $f_1(t - t_i), \ldots, f_n(t - t_i)$.

The key to solve the time-decaying MAB problem is to effectively estimate θ_i, i.e. the weights of individual decaying functions for each arm. This can be estimated through linear bandits, which perform an online estimation of linear weights with bandit feedback. The difference between a general linear bandit and the time-decaying bandit is that the latter contains multiple linear bandits: each arm can be considered as an instance of a linear bandit problem whose context consists of a constant term and a series of temporal functions. At each round, the time-decaying bandit algorithm, for each arm $a_{i,t}$ constructs a matrix $M_{i,t}$ and a vector $C_{i,t}$, which are the sum of the covariance and the reward-weighted sum of features, respectively. $\hat{\rho}_{i,t}$, the least

square estimation of the reward at round t, is given as $x_{i,t}^T M_{i,t}^{-1} C_{i,t}$. To guarantee the sufficient amount of exploration, the following confidence bound is introduced $U_{i,t} = w_t \parallel x_{i,t} \parallel_{M_{i,t}^{-1}}$, where $\parallel x \parallel_M$ is the matrix induced vector norm $\sqrt{x^T M x}$. Time-decaying UCB chooses the arm with the maximum UCB index.

Rotting Bandits (Levine *et al.*, 2017) is another variant of the MAB framework, where each arm's expected reward decays as a function of the number of times it has been pulled. This approach is also motivated by many real-world scenarios, such as online advertising or content recommendation. A number of heuristic-based algorithms for the rotting bandit framework have been proposed. The *sliding window average* (SWA) is a heuristic for ensuring, with high probability that, at each time step, the agent did not sample significantly sub-optimal arms too many times. The heuristic is designed for the non-parametric setting, where the only available information is that the expected rewards sequences are positive and non-increasing in the number of pulls. The idea behind SWA is that after a significantly sub-optimal arm has been pulled "enough" times, the empirical average of these pulls would be distinguishable from the optimal arm for that time step. Thus, given any time step, there is a bounded number of significantly sub-optimal pulls compared to the optimal policy.

The parametric setting assumes that there is prior knowledge that the expected reward is comprised of a sum of an unknown constant part and a rotting part known to belong to a set of specific models. Two heuristics are proposed for this setting. The *closest to origin* (CTO) heuristic simply states that the true underlying model for an arm is the one that best fits the past rewards. The fitting criterion is proximity to the origin of the sum of expected rewards shifted by the observed rewards. The *differences closest to origin* (D-CTO) approach is composed of two stages: first, detecting the underlying rotting models, then estimating and controlling the pulls due to the constant terms.

Lacerda (2017) describe a multi-objective ranked bandit algorithm that dynamically prioritizes different recommendation quality metrics during the life cycle of the user in the system. The algorithm is based on the ranked bandit algorithm (Radlinski *et al.*, 2008a) (Section 4.1)

and consists of four elements: (1) a scalarization function, (2) a set of recommendation quality metrics, (3) a weighting scheme, and (4) a base MAB algorithm. The scalarization function defines how to weight each objective to compute a relevance score for each item. The quality metrics considered are accuracy, diversity and novelty. Accuracy is a typical goal of recommender systems ensuring that recommended items are relevant to the query. However, suggesting items that are not easily discovered by the users is also essential, and it can be measured by the diversity and novelty of the recommendations. Diversity is related to the internal differences within a ranking, whereas novelty can be understood as the difference between present and past experiences of the user. Additionally, different scenarios require different objectives prioritization. A weighting regressor is used to dynamically update the weights for each recommendation quality metric. Three different weighting schemes are considered taking into account user, system, and ranking position perspectives.

The multi-objective ranked bandit algorithm was tested using the Yahoo! News dataset. In the dynamic weighting schemes, the highest CTR is obtained when prioritizing the ranking position, with the system prioritization obtaining the lowest results. In terms of quality metric, the best performance is achieved when prioritizing novelty. The best performing base algorithm is Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1), which outperforms LinUCB (Li *et al.*, 2010a) (Section 6.1.1) and UCB1 (Auer *et al.*, 2002a) (Section 2.2).

Bouneffouf *et al.* (2012) tackle the problem of user's content evolution in mobile context-aware recommender systems. For example, in certain situations the user might need the best information that can be recommended by the system, such as, during a professional meeting. In such a situation, the system must exclusively perform exploitation. In the other case, where, for example, the user is using the system at home for entertainment, the system can be exploratory in its recommendations. The user's model is composed of a set of situations with their corresponding user's preferences.

To allow the system to decide whether to exploit or explore, the contextual ϵ-greedy algorithm (Sutton and Barto, 1998) (Section 2.2) is used. The algorithm compares the current user's situation with

the general class of situations. Depending on the similarity between the current situation and its closest situation from the general set of situations, two scenarios are possible. If the similarity is higher than a pre-defined threshold, the ϵ-greedy algorithm is used with $\epsilon = 0$ (exploitation). If the similarity falls below a pre-defined threshold, then the ϵ-greedy algorithm is used with $\epsilon > 0$ (exploration) with the value of ϵ dependant on the level of similarity to a given pre-defined situation. The degree of exploration decreases when the similarity between the two situations increases.

6.6 Simultaneous Multiple Arms Evaluation

In many recommender systems, the user can select or click on multiple items. This situation can be translated into a bandit problem where multiple arms can be pulled simultaneously.

The framework of *combinatorial multi-armed bandit* (CMAB)[4] allows an arbitrary combination of arms into super arms. The CMAB problem contains a constraint $\mathcal{S} \subseteq 2^{[n]}$, where $2^{[n]}$ is the set of all possible subsets of n arms. Every set of arms $S \in \mathcal{S}$ is a super arm of the CMAB problem. In each round, one super arm $S \in \mathcal{S}$ is played and the outcomes of arms in S are revealed. The algorithm does not have the direct knowledge about the problem instance, e.g. how super arms are formed from the underlying arms and how reward is defined. Instead, the algorithm has access to a computation oracle that takes the expectation vector of possible rewards as the input and computes the optimal or near-optimal super arm S. The CMAB approach is implemented through the CUCB algorithm. The algorithm maintains an empirical mean $\hat{\mu}_i$ for each arm a_i. The actual expectation vector μ given to the oracle contains an adjustment term for each $\hat{\mu}_i$, which depends on the round number t and the number of times arm a_i has been played stored in variable n_i. Thus, $\mu_i = \hat{\mu}_i + \sqrt{\frac{3 \log t}{n_i}}$ The algorithm plays the super arm returned by the oracle and updates the variables n_i and $\hat{\mu}_i$. In the model, all arms have

[4] See also Kale *et al.* (2010) for a theoretical analysis of the slate selection problem, where the user can select a subset from K possible actions and receive rewards for the selected actions only. The problem formulation is also inspired by online ad or news selection.

bounded support on [0, 1] but with the adjustment μ_i may exceed 1. If values above 1 are illegal to the oracle, then they are replaced with 1.

The CMAB approach was inspired by advertising scenario, where a website contains a set of web pages and has a set of users visiting the website. An advertiser wants to place an ad on a set of selected web pages on the site but due to budget constraint, the ads can only be placed at most k web pages. Each user visits a certain set of pages, and on each visited page has a certain click-through probability of clicking the ad on the page but the advertiser does not know these probabilities. The advertiser thus aims to repeatedly select sets of k web pages, observe the CTR to learn the click-through probabilities, and maximize the number of users clicking the ads. In this application, page/user pairs can be viewed as arms but they are not played one by one. Instead, these arms form combinatorial structures and in each round, a set of arms (a super arm) are played together. The reward structure is not a simple linear function of the outcomes of all played arms but takes a more complicated form. For example, in the online advertising scenario, for all page-user pairs with the same user, the collective reward of these arms is either 1 if the user clicks an ad on at least one of the pages and 0 if the user does not click any ads on any page.

In a similar vein, Tang *et al.* (2014) explore ensemble strategies for contextual bandit algorithms by aggregating different pulling policies. This is obtained through creating a meta-bandit paradigm that places a hyper bandit over the base bandits to explicitly explore/exploit the relative importance of base bandits based on user feedback. Two algorithms are proposed to address this problem: HyperTS and HyperTSFB. The idea of these two algorithms is to distribute the trials to the base bandit policies. Given a set of policies $\Pi = \{\pi_i, \ldots, \pi_k\}$ and a context x, both algorithms make two decisions to determine which arm to pull. First, they need to select a policy from Π and then, based on that selection, choose the arm a to pull. To address the policy selection problem, both algorithms leverage non-contextual Thompson sampling. Generally, in each trial, the algorithms randomly select a policy $\pi_i \in \Pi$ to maximize the expected reward of policy π_i. Each algorithm uses a different approach to estimating the expected reward.

HyperTS estimates the expected reward of each policy $\pi_i \in \Pi$ using Monte Carlo method. More precisely, let $x_1, \ldots, x_k$ be the contexts of k trials in which π_i is selected. $x_1, \ldots, x_k$ are samples drawn from $p(x)$. For an input context x_j, π_i pulls the arm a_j and receives the reward ρ_j, where a_j is a sample from $p(a = \pi_i(x_j) \mid x_j)$ and ρ_j is a sample drawn from $p(\rho \mid x_j, a_j)$. Thus, (x_j, a_j, ρ_j) is a sample drawn from the joint distribution $p(x, a, \rho), j = 1, \ldots, k$. The Monte Carlo estimate is $\hat{\mathrm{E}}[\rho_{\pi_i}] = \frac{1}{k} \sum_{j=1}^{k} \rho_j$. The rewards $\rho_1, \ldots, \rho_k \in \{0, 1\}$ are drawn from the Bernoulli distribution $p(\rho)$, which is a marginal distribution of $p(x, a, \rho)$. Therefore, $\hat{\mathrm{E}}[\rho_{\pi_i}]$ follows the Beta distribution: $\hat{\mathrm{E}}[\rho_{\pi_i}] \sim Beta(1+\alpha\pi_i, 1+\beta\pi_i)$, where $\alpha\pi_i = \sum_{j=1}^{k} \rho_j$ and $\beta\pi_i = k - \alpha\pi_i$ with α and β being the parameters of the Beta distribution. For the prior, $Beta(1, 1)$ is used. The selected policy is the one maximizing ρ_i.

In HyperTS, the expected reward of each base policy is estimated only from the feedback when that policy is selected, while the feedback of the decision made by other policies is not utilized. If the number of policies in Π is large, then the total number of trials needed to explore the performance of base policies will also be large, while the total reward will be smaller. HyperTSFB (HyperTS with shared feedback) improves the estimation efficiency by fully utilizing every received feedback for expected reward estimation. Given context x, HyperTSFB requires each base policy $\pi_i \in \Pi$ to provide the probability of π_i pulling arm a. Then, even though the policy π_i is not selected, HyperTSFB can still utilizes the feedback for x to estimate the expected reward of π_i. In the implementation, a sampling-based method is used to obtain the value of $p(a|x)$. For each given context x, HyperTSFB is run multiple times and then $p(a|x)$ is estimated according to the frequency of a being selected. $p(a)$ is the marginal probability of a being selected, which is simply approximated by the ratio of a being pulled in all previous trials done by HyperTSFB.

The two algorithms were compared against LinUCB (Li *et al.*, 2010a) (Section 6.1.1), Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1) and ϵ-greedy (Sutton and Barto, 1998) (Section 2.2) using the Yahoo! Today news module. In terms of CTR, both algorithms significantly outperform LinUCB and Thompson sampling. HyperTS

performs marginally worse than ϵ-greedy, while HyperTSFB performs marginally better than ϵ-greedy.

Stochastic combinatorial semi-bandit (Wen *et al.*, 2015) is another online learning problem where at each step the user or the system can select a subset of items subject to combinatorial constraints, and then observes stochastic weights of these items and receives their sum as a payoff. Two algorithms are proposed to solve the combinatorial semi-bandit problem: Combinatorial Linear Thompson Sampling (CombLinTS) and Combinatorial Linear UCB (CombLinUCB), which are respectively motivated by Thompson sampling (Chapelle and Li, 2011) (Section 2.2.1) and LinUCB (Li *et al.*, 2010a) (Section 6.1.1). Both algorithms maintain a mean vector $\hat{\theta}$ and a covariance matrix M, and use Kalman filtering to update them. They differ in how to choose the subset of arms to explore in each round t.

CombLinTS consists of three steps. First, it randomly samples a coefficient vector θ_t from a Gaussian distribution. Second, it selects a subset of arms to explore based on θ_t and the pre-specified oracle. Finally, it updates the mean vector θ_{t+1} and the covariance matrix M_{t+1} based on Kalman filtering.

ComLinUCB also consists of three steps. First, for each arm in the ground set $e \in E$, it computes an upper confidence bound (UCB) weight vector $\hat{w}_{e_t}$. Second, it computes a set of arms to explore based on $\hat{w}_{e_t}$ and the pre-specified oracle. Finally, it updates the mean vector θ_{t+1} and the covariance matrix M_{t+1} based on Kalman filtering.

6.7 Summary

This chapter covered major areas of recommender systems where bandit algorithms have been applied. Bandits have had a major impact in personalized recommendation, mostly through the application of contextual bandits (Section 6.1.1) and Thompson sampling (Section 2.2.1). Some of the bandit algorithms created in the context of personalized recommendation have had a major impact beyond its original application. In particular, LinUCB (Li *et al.*, 2010a) or its derivatives has been applied in all areas of information retrieval (as can be attested by all the remaining chapters of this survey). Graph based bandits is

another class of algorithms that was established as a result of research into leveraging knowledge from social connections to create more personalized recommendation (Section 6.2.1). Other bandit approaches aimed at efficiency improvements in a recommender system were also considered, such as faster discovery of relevant features (Section 6.4), accounting for a limited lifespan of recommended products (Section 6.5), or accounting for the fact that users often select/click on more than one product at a time (Section 6.6).

7

Other Applications

Previous chapters described areas of information retrieval where bandit algorithms have been widely applied: click models (Chapter 3, ranking (Chapter 4), ranker evaluation (Chapter 5) and recommender systems (Chapter 6). In this chapter, we briefly mention other areas of information retrieval where bandits are gradually making inroads: specialized short text recommendation and ranking (tweets or comments) (Section 7.1), multimedia retrieval (Section 7.2), and web-page layout (Section 7.3).

7.1 Specialized Short Text Recommendation

Mahajan *et al.* (2012) develop a bandit algorithm that leverages past comment ratings to rank an article's comments. Commenting environments are highly dynamic with new comments and articles arriving continuously. Additionally, new comments have few, if any, user ratings, and constantly changing content that may have little overlap with previous comments. Thus, simply ranking comments based on their predicted rating scores may lead to poor comments being ranked high and good comments being ranked low because. The problem can be framed as an exploration–exploitation trade-off and solved with a bandit

approach. A contextual bandit algorithm (Section 6.1.1) called LogUCB is proposed.

LogUCB estimates average ratings for comments using logistic regression on features. Logistic regression ensures that average ratings lie in the fixed range [0, 1]. First, a global logistic regression model is created by pooling comments and user ratings data across past articles. This model is used to initialize the parameters of the per-article model. For a given article, a per-article logistic regression model is learned at regular time intervals taking into consideration features and ratings of comments given to that article during a specific time period. LogUCB adopts a UCB-based approach to handle the exploitation-exploration trade-off. For each comment, it uses Bayesian estimation methods to compute the mean rating estimate μ and its variance σ^2. The UCB for each comment is calculated as $\mu + \alpha\sigma^2$, where α is a parameter that controls the variance. The algorithm selects the comments with the top k UCB values to show to users. The system collects user feedback (thumbs-up/down ratings) on the displayed comments until there is a sufficient number of new ratings, at which point a periodic batch update of the per-article model parameters is performed. In experiments with a real-life comments dataset from Yahoo! News, LogUCB outperforms state-of-the-art contextual bandit method LinUCB (Li *et al.*, 2010a) in terms of click prediction accuracy.

Ueda *et al.* (2017) tackle the problem of collecting non-geotagged local tweets via bandits. The aim is to find new users in a specific location (exploration) and collect tweets from them (exploitation). The bandit algorithm tries to find such users based on the rewards that are calculated as the number of tweets a user posts in a specific location. The users to follow are selected by the ϵ-greedy algorithm (Sutton and Barto, 1998) (Section 2.2) based on the largest cumulative proxy reward. The algorithm collects tweets from selected users over a pre-defined time-frame. For each collected tweet, a probability that it came from a given location is estimated. At the end of the time-frame, a proxy reward for the followed user is calculated as the normalized probability of the collected tweets.

7.2 Multimedia Retrieval

Bandit algorithms have also had limited application in image and music retrieval. PinView (Auer *et al.*, 2010) is a content-based image retrieval system that exploits implicit relevance feedback during a search session. The system is based on LinRel (Auer, 2002) – a contextual bandit algorithm described in more detail in Section 2.2, which allows the system to learn a similarity metric between images based on the current interests of the user.

A similar approach to interactive content-based image retrieval was proposed in Hore *et al.* (2015) and Konyushkova and Glowacka (2013). Hierarchical Gaussian Process bandits (Dorard *et al.*, 2009) (Section 6.1.1) are used to capture the similarity between images. Self-Organizing Maps (SOM) (Kohonen, 1998) of image features are used as layers in the bandit hierarchy to improve the efficiency of the algorithm called GP-SOM. SOM provides model vectors that are treated in the algorithm as discretization of the input space. The algorithm applies a 2-layer bandit settings. First, a model vector is selected and then an image is sampled from among the images associated with a particular model vector. The selection is repeated K times to obtain K images to present to the user. Experimental results show that for small values of K, GP-SOM outperforms LinRel.

Interactive music recommendation can also be formulated as an exploration-exploitation trade-off. The approach presented in Wang *et al.* (2014) and Xing *et al.* (2014) focuses on audio content and novelty. A user's preference P_u can be represented as a linear function of music audio content of a song x and parameter vector u representing user preference of different music features: $P_u = ux$. The novelty of a particular song decays immediately after listening to it and then gradually recovers. The novelty recovers following the function: $P_c = 1 - z^{-t/n}$, where t is the time elapsed since the last listening of the song, n is a parameter indicating the recovery speed and z is the decay parameter. The complete user rating model P_l is a combination of user preferences P_u and the novelty score P_c. The Bayes-UCB algorithm

(Kaufmann *et al.*, 2012) is used to recommend songs to the user. Bayes-UCB recommends song l, which maximizes:

$$l = \arg \max_{l=1\ldots,L} Q(\alpha, \lambda_l),$$

where Q satisfies $\mathbb{P}[P_l \leq Q(\alpha, \lambda_l)] = \alpha$ and L is all songs in the database; $\alpha = 1 - \frac{1}{l_t+1}$; and λ_l denotes $p(P_l \mid D_{l_t})$. The set D_{l_t} contains recommendations accumulated up till now and subscript l_t indicates the number of recommendations accumulated so far.

7.3 Web-page Layout

Tang *et al.* (2013) approach ad layout optimization as an instance of a contextual multi-armed bandit problem (Langford and Zhang, 2008). Each of the n possible ad layouts is represented by an arm. When a page is requested by a user, the ad server is called to fill up all reserved ad slots. Each slot has a size constraint and an available set of ad formats to choose from. The ad-server has to choose one of the formats for the slot and the ads that will be shown within the format. A format may consist of more than one ad. A click on any ad within the format is considered a click on the format itself. Every instance to select an ad format on a page is referred to as an *opportunity*.

Context for an opportunity is encoded as a feature vector and can include anything that is known about opportunity i, such as features of a predictive model or hard constraints e.g., size of the space allocated for the ads. The context facilitates the selection of a subset of feasible layouts for each opportunity.

Each opportunity i of a contextual bandit can be decomposed into three steps: (1) At time t, a context represented as a feature vector x and reward ρ are drawn from an unknown distribution $(x_t, \rho_t) \sim \mathcal{D}$. The context is revealed to the user but the reward is not. (2) Based on a policy π, the user chooses an arm $\pi(x_t)$, given the revealed context. (3) The reward ρ_t is revealed based on the user's choice. The policy π may be revised with the data collected for this opportunity. Layout optimization aims to learn a policy which maximizes the average expected reward per opportunity, where the reward is defined as a click.

The problem arises when trying to find the optimal policy as that would require testing many bandit algorithms on live traffic. However, off-line replay (*exploration scavenging* Langford *et al.*, 2008; Li *et al.*, 2011a)[1] can be adapted to provide an accurate estimator for the performance of ad layout policies using only historical data about the effectiveness of layouts.

Offline replay describes a class of sample estimators S:

$$\hat{S}(\pi) = \frac{1}{T} \sum_{t=1}^{T} \sum_{a \in A} \rho_s(x_t) \mathbb{1}[\pi(x_t) = s(x_t)] w_{t,a},$$

where A is a list of layouts (arms), $\mathbb{1}[\cdot]$ is the indicator function, s is a fixed serving policy $\pi \neq s$ and $w_{t,a}$ is a normalization weight:

$$w_{t,a} = \frac{1}{P(s(x_t) = a \mid \pi(x_t) = a)}.$$

In the case of LinkedIn ad layout selection problem studied by Tang *et al.*, the contextual features of an opportunity are the channel (web page) and the layout size. Each channel and size combination has its own set of admissible formats – a layout can only be recommended for one channel and one layout size.

In order to evaluate the replayer estimation framework at scale, it was implemented in the Map/Reduce framework. The testing event set was partitioned into several subsets with each reducer handling only the evaluation for one event subset. Offline evaluation was tested in two scenarios. One scenario considered static recommendation policies, where the models never changed during the evaluation. In this scenario, all models were built before the start of the evaluation. The other scenario considered dynamic recommendation policies with the models updated based on the user feedback recorded from the events that they recommended. The experimental framework was tested with many bandit algorithms (various versions of ϵ-greedy (Sutton and Barto, 1998), UCB (Auer *et al.*, 2002a), and Thompson sampling (Chapelle and Li, 2011)) in the context of a large system, the LinkedIn Ad platform. Thompson sampling was overall the best policy in the evaluation for

[1] See Section 4.2 for more details about search engine optimization with offline evaluation

ad format selection, considering both the cold-start and warm-start settings. Additionally, the proposed offline policy evaluation approach was compared with the online production system to demonstrate its accuracy. The comparison showed that the normalized reward of the off-line system was close to the actual value of the revenue per request obtained in the online production system.

Hill *et al.* (2017) focus on multivariate optimization of interactive web pages. Bandit methodology is applied to explore the layout space efficiently and hill-climbing is used to select optimal content in realtime. To avoid a combinatorial explosion in model complexity, only pairwise interactions between page components are considered.

The problem is formally defined as the selection of a layout a of a web page under a context x in order to maximize the expected value of a reward ρ, which corresponds to the value of an action taken by a user after viewing the web page, such as click, signup or purchase. Context u represents user or session information that may impact a layout's expected reward, such as time of day, device type or user history. u and a are combined to form the final feature vector x. The reward ρ for a given layout and context depends on a linear scaling of x by a fixed but unknown vector of weights w.

There are n arms, one per layout. The algorithm proceeds in discrete time steps t. On trial t, a context u_t is generated and a vector $x_{a,t}$ is revealed for each arm. Thompson sampling (Chapelle and Li, 2011) selects a layout proportionally to the probability of that layout being optimal conditioned on previous observations: $a_t \sim P(a = a^\star \mid u, h_{t-1})$, where H_{t-1} indicates previous observations up to time $t-1$. In practice, this probability is not sampled directly. Instead, model parameters from their posterior are sampled and the layout that maximizes the reward is selected. Weights w are estimated from history H_{t-1}. In the Bayesian linear pro-bit regression used in the approach, the model weights are represented by independent Gaussian random variables, which allows for efficient sampling.

The proposed approach was deployed to a live production system to combinatorially optimize a landing page that promotes purchases of an Amazon service, leading to a 21% increase in purchase rate.

In Wang *et al.* (2017c), Thompson sampling (Chapelle and Li, 2011) is applied for whole-page recommendation. The problem is motivated by news recommendation portal with six slots to display news articles. Whole-page recommendations, i.e. selecting six articles from a larger pool and placing them in the webpage, is a combinatorial problem. The goal is to find an optimal layout configuration to maximize the expected total CTR. To model this scenario, the following ordered combinatorial bandit problem is considered. Given optional context information, instead of selecting one arm, the learner selects a subset of k actions from a base set of A actions and displays them in k different positions from K possible positions. Due to position and layout bias, it is reasonable to assume that for every article and every position, there is a CTR associated with the (content/position) pair, which specifies the probability that a user will click on the content if it is displayed in a certain position. This setting explicitly models the positions of the subset of selected arms, hence it is called *ordered combinatorial semi-bandits*.

More formally, each round t, the user is presented with a (optional) context vector c_t. In order to take layout information into consideration, a feature vector $x_{a,l}$ is constructed for each (action/position) pair (a, l), such that $x_a \in A$, and $l \in \{1, 2, \ldots, L\}$. The user chooses n actions from A to display in n positions. A valid combinatorial subset is a mapping from n different actions to n different positions, i.e. it is a one-to-one mapping $\pi_t : 1, 2, \ldots, n \mapsto (A, \{1, 2, \ldots, L\})$. Each π_t is a superarm. The user receives reward $\rho_{\pi,t}$ for each chosen (action, position) pair. The total reward of round t is the sum of the rewards of each position n. A modified version of Thompson sampling based on a min-cost max-flow network is used to select the best superarm at each round t. Experimental results with the Yahoo! Front Page Webscope datasets show that the proposed approach outperforms simple explore-exploit settings, such as ϵ-greedy and pure exploitation.

7.4 Summary

In this chapter, we provided a brief overview of three areas of information retrieval where the application of bandit algorithms is still rather rare, i.e.

short text ranking/recommendation (Section 7.1), multimedia retrieval (Section 7.2) and web-page layout optimization (Section 7.3). In all the discussed applications, the best preforming approaches were based on contextual or Bayesian bandits with Thompson sampling being the most popular method.

8

Conclusions and Future Directions

This survey covered major applications of bandit algorithms in information retrieval and recommender systems. There are two broad areas where the use of bandits has been particularly prominent: online ranker evaluation (Chapter 5) and personalization in recommender systems (Section 6.1), spurring the development of a large number of algorithms from two "families" of bandit algorithms, i.e. dueling bandits and contextual bandits. The user behavioural aspects as well as other practical considerations, such as advertisers' requirements or type of system's platform, had further impact on the algorithmic development of bandits. Hence, recently we saw a number of algorithms inspired by users' click behaviour in an IR setting (Chapter 3), algorithms that take into consideration the social network of users (Section 6.2), algorithms designed specifically for short-lived ads or news items (Section 6.5), or mobile platforms (Section 6.4).

Bayesian bandits, in particular Thompson sampling, is also a very popular approach applied in virtually all the IR/recommender systems applications discussed in this survey. Thompson sampling's popularity is partly due to its algorithmic simplicity, but largely to its ability to incorporate the uncertainty about, often idiosyncratic, user behaviour.

In spite of very practical underpinnings, a great majority of research into bandits algorithms discussed in this survey focus on theoretical analysis and simulation experiments with existing datasets. The number of studies related to practical incorporation of bandits in a real-life commercial systems is still somewhat limited, e.g. content publishing tool used in Yahoo! (Agarwal *et al.*, 2009) or web-page layout in LinkedIn (Tang *et al.*, 2013). Similarly, there are only a few studies of information retrieval systems based on bandit algorithms, such as PinView (Auer *et al.*, 2010) for image retrieval, or PULP (Medlar *et al.*, 2016) and SciNet (Ruotsalo *et al.*, 2015) for scientific literature search. These systems, however, are rather small-scale compared to a real-life commercial system. Thus, one of the challenges is better understanding of engineering and optimization issues associated with the deployment of bandit algorithms in large-scale commercial systems.

An important direction for future research is development of interactive learning in the bandit setting. At the heart of the majority of existing bandit applications is the assumption that the underlying payoff function is known *a priori* and so, in principle, it can be maximized. In many, if not the majority of, real-life applications, however, the payoff function is not fully known in advance and can only be estimated via interactions of the system with the user. A recent study of interactive submodular bandits is one of the first attempts in this direction (Chen *et al.*, 2017). Another related issue concerns personalization at the individual or even task level (Medlar *et al.*, 2017). Most of the bandit-based personalization discussed in this survey focus on personalization through group assignment. However, this approach risks providing non-personalized or even wrong type of support to users who do not follow main social trends.

Acknowledgements

I would like to thank Alan and Emilia for their patience.

Appendices

A

Algorithms and Methods Abbreviations

Abbreviation	Full Name
BTM	Beat-the-Mean
CAB	Context Aware clustering of Bandits
CCB	Copeland Confidence Bound
CLUB	Cluster of Bandits
CMAB	Combinatorial MAB
COFIBA	Collaborative Filtering Bandit
CoLin	Collaborative Linear Bandit
CTO	Closest to origin
CTR	Click through rate
CUCB	Combinatorial UCB
CW-RMED	Copeland Winners Relative Minimum Empirical Divergence
DBGD	Dueling Bandit Gradient Descent
DCM	Dependent Click Model
D-CTO	Differences CTO
D-TS	Double Thompson Sampling
ECW-RMED	Efficient CW-RMED

Continued on next page

Abbreviation	Full Name
ERBA	Exploitative Ranked Bandits Algorithm
GP	Gaussian Process
IBA	Independent Bandit Algorithm
IF	Interleave Filter
IR	Information Retrieval
KL	Kullback–Leibrel
LCB	Latent Contextual Bandits
Lin	Linear
MAB	Multi-armed bandit
PBM	Position Based Model
PIE	Parsimonious Item Exploration
QAC	Query Auto-Completion
PMED	Permutation Minimum Empirical Convergence
PMF	Probabilistic Matrix Factorization
PTS	Particle Thompson sampling
RBA	Ranked Bandit Algorithm
RBPF	Rao-Blackwellized particle filter
RCS	Relative Confidence Sampling
REC	Ranked Explore and Commit
RMED	Relative Minimum Empirical Divergence
RUCB	Relative Upper Confidence Bound
SCB	Scalable Copeland Bandit
SERP	Search Results Page
SOM	Self-Organizing Map
SWA	Sliding Window Avarage
TS	Thompson sampling
UCB	Upper Confidence Bound
VB	Variational Bayes

B

Symbols

Symbol	Meaning
A	A list of items
a_i	i^{th} item in list A
B_t	Vector with past observations at time t
C_t	User feedback/click at time t
$D_{KL}(p\|\|q)$	KL divergence between two random variables with means p and q
d	Number of dimensions
E	A ground set of items
γ	Position discount
I_d	$d \times d$ identity matrix
K	A list of K presented items
k	Position in list K
κ	Covariance/kernel function
l	Position in list L
κ_k	Probability of observing item in position k
L	Number of items
λ	Regularization parameter

Continued on next page

Symbol	Meaning
M_t	Positive definite matrix with past observations at time t
$N_e(t)$	Number of counts of item e at time t
O_t	Index of item with weight 0
R	Regret
r_t	Regret at time t
ρ	Reward
s	Number of observed weights
σ	Parameter that controls learning rate
$T_t(e)$	Number of times item e is observed in t steps
t	Time/number of steps
θ	Parameter vector
w	Weight
$w(a)$	Weight of item a
$w_t(a)$	Weight of item a at time t
$U(e)$	Upper Confidence Bound of item e
U_t	Upper Confidence Bound at time t
u	User feature vector
v	Termination probability
X	Matrix with rows of feature vectors of items
x	Item feature vector
Y	Column vector of observed weights
Z_k	Observation at position k

References

Agarwal, D., B.-C. Chen, P. Elango, N. Motgi, S.-T. Park, R. Ramakrishnan, S. Roy, and J. Zachariah. 2009. "Online models for content optimization". In: *Advances in Neural Information Processing Systems*. 17–24.

Agrawal, R. 1995. "Sample mean based index policies by o (log n) regret for the multi-armed bandit problem". *Advances in Applied Probability*. 27(4): 1054–1078.

Agrawal, S. and N. Goyal. 2012. "Analysis of thompson sampling for the multi-armed bandit problem". In: *Conference on Learning Theory*. 39–1.

Ailon, N., Z. Karnin, and T. Joachims. 2014. "Reducing dueling bandits to cardinal bandits". In: *International Conference on Machine Learning*. 856–864.

Auer, P. 2002. "Using confidence bounds for exploitation-exploration trade-offs". *Journal of Machine Learning Research*. 3(Nov): 397–422.

Auer, P., N. Cesa-Bianchi, and P. Fischer. 2002a. "Finite-time analysis of the multiarmed bandit problem". *Machine Learning*. 47(2-3): 235–256.

Auer, P., N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. 2002b. "The nonstochastic multiarmed bandit problem". *SIAM Journal on Computing*. 32(1): 48–77.

Auer, P., Z. Hussain, S. Kaski, A. Klami, J. Kujala, J. Laaksonen, A. P. Leung, K. Pasupa, and J. Shawe-Taylor. 2010. "Pinview: Implicit Feedback in Content-Based Image Retrieval". In: *WAPA*. 51–57.

Awerbuch, B. and R. Kleinberg. 2008. "Online linear optimization and adaptive routing". *Journal of Computer and System Sciences*. 74(1): 97–114.

Bouneffouf, D., A. Bouzeghoub, and A. L. Gançarski. 2012. "A contextual-bandit algorithm for mobile context-aware recommender system". In: *International Conference on Neural Information Processing*. Springer. 324–331.

Bresler, G., G. H. Chen, and D. Shah. 2014. "A latent source model for online collaborative filtering". In: *Advances in Neural Information Processing Systems*. 3347–3355.

Brost, B., I. J. Cox, Y. Seldin, and C. Lioma. 2016a. "An improved multileaving algorithm for online ranker evaluation". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM. 745–748.

Brost, B., Y. Seldin, I. J. Cox, and C. Lioma. 2016b. "Multi-dueling bandits and their application to online ranker evaluation". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2161–2166.

Busa-Fekete, R., E. Hüllermeier, and A. E. Mesaoudi-Paul. 2018. "Preference-based online learning with dueling bandits: A survey". *arXiv preprint arXiv:1807.11398*.

Busa-Fekete, R., B. Szörényi, and E. Hüllermeier. 2014. "PAC Rank Elicitation through Adaptive Sampling of Stochastic Pairwise Preferences". In: *AAAI*. 1701–1707.

Cai, F. and M. De Rijke. 2016. "A survey of query auto completion in information retrieval". *Foundations and Trends® in Information Retrieval*. 10(4): 273–363.

Caron, S. and S. Bhagat. 2013. "Mixing Bandits: A Recipe for Improved Cold-start Recommendations in a Social Network". In: *Proceedings of the 7th Workshop on Social Network Mining and Analysis. SNAKDD '13*. Chicago, Illinois: ACM. 11:1–11:9. DOI: 10.1145/2501025.2501029.

Cesa-Bianchi, N., C. Gentile, and G. Zappella. 2013. "A gang of bandits". In: *Advances in Neural Information Processing Systems*. 737–745.

Chakrabarti, D., R. Kumar, F. Radlinski, and E. Upfal. 2009. "Mortal multi-armed bandits". In: *Advances in Neural Information Processing Systems*. 273–280.

Chapelle, O. and L. Li. 2011. "An empirical evaluation of thompson sampling". In: *Advances in Neural Information Processing Systems*. 2249–2257.

Chen, L., A. Krause, and A. Karbasi. 2017. "Interactive Submodular Bandit". In: *Advances in Neural Information Processing Systems*. 140–151.

Chuklin, A., I. Markov, and M. d. Rijke. 2015. "Click models for web search". *Synthesis Lectures on Information Concepts, Retrieval, and Services*. 7(3): 1–115.

Combes, R., S. Magureanu, A. Proutiere, and C. Laroche. 2015. "Learning to Rank: Regret Lower Bounds and Efficient Algorithms". *SIGMETRICS Perform. Eval. Rev.* 43(1): 231–244. DOI: 10.1145/2796314.2745852.

Cormack, G. V., C. R. Palmer, and C. L. Clarke. 1998. "Efficient construction of large test collections". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 282–289.

Craswell, N., O. Zoeter, M. Taylor, and B. Ramsey. 2008. "An experimental comparison of click position-bias models". In: *Proceedings of the 2008 international conference on web search and data mining*. ACM. 87–94.

Daee, P., J. Pyykkö, D. Glowacka, and S. Kaski. 2016. "Interactive Intent Modeling from Multiple Feedback Domains". In: *Proceedings of the 21st International Conference on Intelligent User Interfaces. IUI '16*. Sonoma, California, USA: ACM. 71–75. DOI: 10.1145/2856767.2856803.

Deshpande, Y. and A. Montanari. 2012. "Linear bandits in high dimension and recommendation systems". In: *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE. 1750–1754.

Dorard, L., D. Glowacka, and J. Shawe-Taylor. 2009. "Gaussian process modelling of dependencies in multi-armed bandit problems". In: *Int. Symp. Op. Res.* 77–84.

Dudık, M., K. Hofmann, R. E. Schapire, A. Slivkins, and M. Zoghi. 2015. "Contextual dueling bandits". In: *Proceedings of The 28th Conference on Learning Theory*. 563–587.

Durand, A., J.-A. Beaumont, C. Gagné, M. Lemay, and S. Paquet. 2017. "Query Completion Using Bandits for Engines Aggregation". *arXiv preprint arXiv:1709.04095*.

Garivier, A. and O. Cappé. 2011. "The KL-UCB algorithm for bounded stochastic bandits and beyond". In: *Proceedings of the 24th annual Conference On Learning Theory*. 359–376.

Garivier, A. and E. Moulines. 2011. "On upper-confidence bound policies for switching bandit problems". In: *International Conference on Algorithmic Learning Theory*. Springer. 174–188.

Gentile, C., S. Li, P. Kar, A. Karatzoglou, G. Zappella, and E. Etrue. 2017. "On context-dependent clustering of bandits". In: *International Conference on Machine Learning*. 1253–1262.

Gentile, C., S. Li, and G. Zappella. 2014. "Online Clustering of Bandits". In: *ICML*. 757–765.

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

Głowacka, D., L. Dorard, A. Medlar, and J. Shawe-Taylor. 2009. "Prior Knowledge in Learning Finite Parameter Spaces". In: *International Conference on Formal Grammar*. Springer. 199–213.

Guo, F., C. Liu, and Y. M. Wang. 2009. "Efficient multiple-click models in web search". In: *Proceedings of the second ACM international conference on web search and data mining*. ACM. 124–131.

He, J., C. Zhai, and X. Li. 2009. "Evaluation of methods for relative comparison of retrieval systems based on clickthroughs". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM. 2029–2032.

Hill, D. N., H. Nassif, Y. Liu, A. Iyer, and S. Vishwanathan. 2017. "An Efficient Bandit Algorithm for Realtime Multivariate Optimization". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '17*. Halifax, NS, Canada: ACM. 1813–1821. DOI: 10.1145/3097983.3098184.

Hofmann, K., S. Whiteson, and M. De Rijke. 2011a. "A probabilistic method for inferring preferences from clicks". In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM. 249–258.

Hofmann, K., S. Whiteson, and M. D. Rijke. 2013a. "Fidelity, soundness, and efficiency of interleaved comparison methods". *ACM Transactions on Information Systems (TOIS)*. 31(4): 17.

Hofmann, K., S. Whiteson, and M. de Rijke. 2011b. "Balancing exploration and exploitation in learning to rank online". In: *European Conference on Information Retrieval*. Springer. 251–263.

Hofmann, K., S. Whiteson, and M. de Rijke. 2013b. "Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval". *Information Retrieval*. 16(1): 63–90.

Hore, S., L. Tyrvainen, J. Pyykko, and D. Glowacka. 2015. "A reinforcement learning approach to query-less image retrieval". In: *International Workshop on Symbiotic Interaction*. Springer. 121–126.

Horvitz, D. G. and D. J. Thompson. 1952. "A generalization of sampling without replacement from a finite universe". *Journal of the American statistical Association*. 47(260): 663–685.

Hsieh, C.-C., J. Neufeld, T. King, and J. Cho. 2015. "Efficient Approximate Thompson Sampling for Search Query Recommendation". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing. SAC '15*. Salamanca, Spain: ACM. 740–746. DOI: 10.1145/2695664.2695748.

Jie, L., S. Lamkhede, R. Sapra, E. Hsu, H. Song, and Y. Chang. 2013. "A Unified Search Federation System Based on Online User Feedback". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '13*. Chicago, Illinois, USA: ACM. 1195–1203. DOI: 10.1145/2487575.2488198.

Joachims, T. 2003. "Evaluating Retrieval Performance Using Clickthrough Data". *Text Mining*.

Kale, S., L. Reyzin, and R. E. Schapire. 2010. "Non-stochastic bandit slate problems". In: *Advances in Neural Information Processing Systems*. 1054–1062.

Katariya, S., B. Kveton, C. Szepesvari, C. Vernade, and Z. Wen. 2016a. "Stochastic rank-1 bandits". *arXiv preprint arXiv:1608.03023*.

Katariya, S., B. Kveton, C. Szepesvári, C. Vernade, and Z. Wen. 2017. "Bernoulli Rank-1 Bandits for Click Feedback". *arXiv preprint arXiv:1703.06513*.

Katariya, S., B. Kveton, C. Szepesvári, and Z. Wen. 2016b. "DCM Bandits: Learning to Rank with Multiple Clicks". In: *Proc. of ICML*.

Kaufmann, E., O. Cappé, and A. Garivier. 2012. "On Bayesian upper confidence bounds for bandit problems". In: *Artificial Intelligence and Statistics*. 592–600.

Kawale, J., H. H. Bui, B. Kveton, L. Tran-Thanh, and S. Chawla. 2015. "Efficient Thompson Sampling for Online Matrix-Factorization Recommendation". In: *Advances in Neural Information Processing Systems*. 1297–1305.

Kleinberg, R., A. Slivkins, and E. Upfal. 2008. "Multi-armed bandits in metric spaces". In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 681–690.

Kocák, T., M. Valko, R. Munos, and S. Agrawal. 2014. "Spectral Thompson Sampling". In: *AAAI*. 1911–1917.

Kocsis, L. and C. Szepesvári. 2006. "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer. 282–293.

Kohli, P., M. Salek, and G. Stoddard. 2013. "A Fast Bandit Algorithm for Recommendation to Users With Heterogenous Tastes". In: *AAAI*.

Kohonen, T. 1998. "The self-organizing map". *Neurocomputing*. 21(1-3): 1–6.

Komiyama, J., J. Honda, H. Kashima, and H. Nakagawa. 2015. "Regret lower bound and optimal algorithm in dueling bandit problem". In: *Proceedings of The 28th Conference on Learning Theory*. 1141–1154.

Komiyama, J., J. Honda, and H. Nakagawa. 2016. "Copeland dueling bandit problem: regret lower bound, optimal algorithm, and computationally efficient algorithm". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. 1235–1244.

Komiyama, J., J. Honda, and A. Takeda. 2017. "Position-based Multiple-play Bandit Problem with Unknown Position Bias". In: *Advances in Neural Information Processing Systems*. 5005–5015.

Komiyama, J. and T. Qin. 2014. "Time-decaying bandits for non-stationary systems". In: *International Conference on Web and Internet Economics*. Springer. 460–466.

Konyushkova, K. and D. Glowacka. 2013. "Content-based image retrieval with hierarchical Gaussian Process bandits with self-organizing maps". In: *ESANN*.

Koren, Y., R. Bell, and C. Volinsky. 2009. "Matrix factorization techniques for recommender systems". *Computer*. 42(8).

Kuleshov, V. and D. Precup. 2014. "Algorithms for multi-armed bandit problems". *arXiv preprint arXiv:1402.6028*.

Kveton, B., C. Szepesvari, Z. Wen, and A. Ashkan. 2015a. "Cascading Bandits: Learning to Rank in the Cascade Model". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 767–776.

Kveton, B., Z. Wen, A. Ashkan, and C. Szepesvari. 2015b. "Combinatorial cascading bandits". In: *Advances in Neural Information Processing Systems*. 1450–1458.

Kveton, B., Z. Wen, A. Ashkan, and C. Szepesvari. 2015c. "Tight regret bounds for stochastic combinatorial semi-bandits". In: *Artificial Intelligence and Statistics*. 535–543.

Lacerda, A. 2017. "Multi-Objective Ranked Bandits for Recommender Systems". *Neurocomputing*. 246: 12–24.

Lagrée, P., C. Vernade, and O. Cappe. 2016. "Multiple-play bandits in the position-based model". In: *Advances in Neural Information Processing Systems*. 1597–1605.

Lai, T. L. and H. Robbins. 1985. "Asymptotically efficient adaptive allocation rules". *Advances in Applied Mathematics*. 6(1): 4–22.

Langford, J., A. Strehl, and J. Wortman. 2008. "Exploration scavenging". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 528–535.

Langford, J. and T. Zhang. 2008. "The epoch-greedy algorithm for multi-armed bandits with side information". In: *Advances in Neural Information Processing Systems*. 817–824.

Levine, N., K. Crammer, and S. Mannor. 2017. "Rotting bandits". In: *Advances in Neural Information Processing Systems*. 3077–3086.

Li, C., P. Resnick, and Q. Mei. 2016a. "Multiple Queries As Bandit Arms". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, Indiana, USA: ACM. 1089–1098. DOI: 10.1145/2983323.2983816.

Li, L., W. Chu, J. Langford, and R. E. Schapire. 2010a. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web*. 661–670.

Li, L., S. Chen, J. Kleban, and A. Gupta. 2015. "Counterfactual estimation and optimization of click metrics in search engines: A case study". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM. 929–934.

Li, L., W. Chu, J. Langford, T. Moon, and X. Wang. 2011a. "An Unbiased Offline Evaluation of Contextual Bandit Algorithms with Generalized Linear Models". In: *JMLR Workshop and Conference Proceedings vol. 26: On-line Trading of Exploration and Exploitation 2*. 19–36.

Li, L., W. Chu, J. Langford, and X. Wang. 2011b. "Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms". In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining. WSDM '11*. Hong Kong, China: ACM. 297–306. DOI: 10.1145/1935826.1935878.

Li, S., A. Karatzoglou, and C. Gentile. 2016b. "Collaborative Filtering Bandits". In: *The 39th International ACM SIGIR Conference on Information Retrieval (SIGIR)*.

Li, S., B. Wang, S. Zhang, and W. Chen. 2016c. "Contextual combinatorial cascading bandits". In: *International Conference on Machine Learning*. 1245–1253.

Li, W., X. Wang, R. Zhang, Y. Cui, J. Mao, and R. Jin. 2010b. "Exploitation and exploration in a performance based contextual advertising system". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 27–36.

Lim, Y. J. and Y. W. Teh. 2007. "Variational Bayesian approach to movie rating prediction". In: *Proceedings of KDD cup and workshop*. Vol. 7. 15–21.

Liu, T.-Y. 2009. "Learning to rank for information retrieval". *Foundations and Trends® in Information Retrieval*. 3(3): 225–331.

Liu, T.-Y., J. Xu, T. Qin, W. Xiong, and H. Li. 2007. "Letor: Benchmark dataset for research on learning to rank for information retrieval". In: *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*. Vol. 310. ACM Amsterdam, The Netherlands.

Losada, D. E., J. Parapar, and Á. Barreiro. 2016. "Feeling lucky?: multi-armed bandits for ordering judgements in pooling-based evaluation". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM. 1027–1034.

Lu, T., D. Pál, and M. Pál. 2009. "Showing relevant ads via context multi-armed bandits". In: *Proceedings of AISTATS*.

Mahajan, D. K., R. Rastogi, C. Tiwari, and A. Mitra. 2012. "LogUCB: An Explore-exploit Algorithm for Comments Recommendation". In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management. CIKM '12*. Maui, Hawaii, USA: ACM. 6–15. DOI: 10.1145/2396761.2396767.

Medlar, A., K. Ilves, P. Wang, W. Buntine, and D. Glowacka. 2016. "PULP: A System for Exploratory Search of Scientific Literature". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '16*. Pisa, Italy: ACM. 1133–1136. DOI: 10.1145/2911451.2911455.

Medlar, A., J. Pyykko, and D. Glowacka. 2017. "Towards Fine-Grained Adaptation of Exploration/Exploitation in Information Retrieval". In: *Proceedings of the 22Nd International Conference on Intelligent User Interfaces. IUI '17*. Limassol, Cyprus: ACM. 623–627. DOI: 10.1145/3025171.3025205.

Mnih, A. and R. R. Salakhutdinov. 2008. "Probabilistic matrix factorization". In: *Advances in Neural Information Processing Systems*. 1257–1264.

Nakamura, A. 2014. "A UCB-Like Strategy of Collaborative Filtering". In: *ACML*. Vol. 39. 315–329.

Nazerzadeh, H., R. Paes Leme, A. Rostamizadeh, and U. Syed. 2016. "Where to sell: Simulating auctions from learning algorithms". In: *Proceedings of the 2016 ACM Conference on Economics and Computation*. ACM. 597–598.

Nguyen, T. T. and H. W. Lauw. 2014. "Dynamic Clustering of Contextual Multi-Armed Bandits". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. CIKM '14*. Shanghai, China: ACM. 1959–1962. DOI: 10.1145/2661829.2662063.

Pandey, S., D. Agarwal, D. Chakrabarti, and V. Josifovski. 2007. "Bandits for Taxonomies: A Model-based Approach". In: *SDM*. SIAM. 216–227.

Qin, L., S. Chen, and X. Zhu. 2014. "Contextual combinatorial bandit and its application on diversified online recommendation". In: *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM. 461–469.

Radlinski, F., R. Kleinberg, and T. Joachims. 2008a. "Learning diverse rankings with multi-armed bandits". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 784–791.

Radlinski, F., M. Kurup, and T. Joachims. 2008b. "How does clickthrough data reflect retrieval quality?" In: *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM. 43–52.

RicciLior, F., L. Rokach, B. Shapira, and P. B. Kantor. 2001. *Recommender Systems Handbook*. Springer.

Richardson, M., E. Dominowska, and R. Ragno. 2007. "Predicting clicks: estimating the click-through rate for new ads". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 521–530.

Robbins, H. 1985. "Some aspects of the sequential design of experiments". In: *Herbert Robbins Selected Papers*. Springer. 169–177.

Ruotsalo, T., J. Peltonen, M. J. Eugster, D. Glowacka, A. Reijonen, G. Jacucci, P. Myllymäki, and S. Kaski. 2015. "SciNet: Interactive Intent Modeling for Information Discovery". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '15*. Santiago, Chile: ACM. 1043–1044. DOI: 10.1145/2766462.2767863.

Sanderson, M. 2010. "Test collection based evaluation of information retrieval systems". *Foundations and Trends® in Information Retrieval*. 4(4): 247–375.

Schuth, A., R.-J. Bruintjes, F. Buüttner, J. van Doorn, C. Groenland, H. Oosterhuis, C.-N. Tran, B. Veeling, J. van der Velde, R. Wechsler, *et al.* 2015. "Probabilistic multileave for online retrieval evaluation". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 955–958.

Shen, W., J. Wang, Y.-G. Jiang, and H. Zha. 2015. "Portfolio Choices with Orthogonal Bandit Learning". In: *IJCAI*. Vol. 15. 974–980.

Slivkins, A., F. Radlinski, and S. Gollapudi. 2010. "Learning optimally diverse rankings over large document collections". In: *Proceedings of ICML*.

Slivkins, A., F. Radlinski, and S. Gollapudi. 2013. "Ranked bandits in metric spaces: learning diverse rankings over large document collections". *Journal of Machine Learning Research*. 14(Feb): 399–436.

Srinivas, N., A. Krause, S. M. Kakade, and M. Seeger. 2010. "Gaussian process optimization in the bandit setting: No regret and experimental design". In: *Proceedings of the 27th international conference on Machine learning*. 1015–1022.

Srinivasan, S., E. Talvitie, and M. H. Bowling. 2015. "Improving Exploration in UCT Using Local Manifolds". In: *AAAI*. 3386–3392.

Streeter, M. and D. Golovin. 2008. "An online algorithm for maximizing submodular functions". In: *Advances in Neural Information Processing Systems*. 1577–1584.

Streeter, M., D. Golovin, and A. Krause. 2009. "Online learning of assignments". In: *Advances in Neural Information Processing Systems*. 1794–1802.

Strehl, A., J. Langford, L. Li, and S. M. Kakade. 2010. "Learning from logged implicit exploration data". In: *Advances in Neural Information Processing Systems*. 2217–2225.

Sui, Y., V. Zhuang, J. W. Burdick, and Y. Yue. 2017. "Multi-dueling Bandits with Dependent Arms". *arXiv preprint arXiv:1705.00253*.

Sui, Y., M. Zoghi, K. Hofmann, and Y. Yue. 2018. "Advancements in Dueling Bandits". In: *IJCAI*. 5502–5510.

Sutton, R. S. and A. G. Barto. 1998. *Reinforcement learning: An introduction*. MIT press Cambridge.

Tang, L., Y. Jiang, L. Li, and T. Li. 2014. "Ensemble Contextual Bandits for Personalized Recommendation". In: *Proceedings of the 8th ACM Conference on Recommender Systems. RecSys '14*. Foster City, Silicon Valley, California, USA: ACM. 73–80. DOI: 10.1145/2645710.2645732.

Tang, L., Y. Jiang, L. Li, C. Zeng, and T. Li. 2015. "Personalized Recommendation via Parameter-Free Contextual Bandits". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '15*. Santiago, Chile: ACM. 323–332. DOI: 10.1145/2766462.2767707.

Tang, L., R. Rosales, A. Singh, and D. Agarwal. 2013. "Automatic ad format selection via contextual bandits". In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 1587–1594.

Teevan, J., S. T. Dumais, and E. Horvitz. 2007. "Characterizing the value of personalizing search". In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 757–758.

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

Ueda, S., Y. Yamaguchi, and H. Kitagawa. 2017. "Collecting Non-Geotagged Local Tweets via Bandit Algorithms". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM. 2331–2334.

Urvoy, T., F. Clerot, R. Féraud, and S. Naamane. 2013. "Generic Exploration and K-armed Voting Bandits". In: *ICML*. 91–99.

Valko, M., R. Munos, B. Kveton, and T. Kocak. 2014. "Spectral Bandits for Smooth Graph Functions". In: *ICML*. 46–54.

Vanchinathan, H. P., I. Nikolic, F. De Bona, and A. Krause. 2014. "Explore-exploit in top-n recommender systems via gaussian processes". In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 225–232.

Villar, S. S., J. Bowden, and J. Wason. 2015. "Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges". *Statistical Science: a review journal of the Institute of Mathematical Statistics*. 30(2): 199–215.

Vorobev, A., D. Lefortier, G. Gusev, and P. Serdyukov. 2015. "Gathering Additional Feedback on Search Results by Multi-Armed Bandits with Respect to Production Ranking". In: *Proceedings of the 24th International Conference on World Wide Web. WWW '15*. Florence, Italy: ACM. 1177–1187. DOI: 10.1145/2736277.2741104.

Wang, H., Q. Wu, and H. Wang. 2016. "Learning Hidden Features for Contextual Bandits". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16*. Indianapolis, Indiana, USA: ACM. 1633–1642. DOI: 10.1145/2983323.2983847.

Wang, H., Q. Wu, and H. Wang. 2017a. "Factorization Bandits for Interactive Recommendation". In: *AAAI*. 2695–2702.

Wang, X., Y. Wang, D. Hsu, and Y. Wang. 2014. "Exploration in Interactive Personalized Music Recommendation: A Reinforcement Learning Approach". *ACM Trans. Multimedia Comput. Commun. Appl.* 11(1): 7:1–7:22. DOI: 10.1145/2623372.

Wang, Y., H. Ouyang, H. Deng, and Y. Chang. 2017b. "Learning Online Trends for Interactive Query Auto-Completion". *IEEE Transactions on Knowledge and Data Engineering*. 29(11): 2442–2454.

Wang, Y., H. Ouyang, C. Wang, J. Chen, T. Asamov, and Y. Chang. 2017c. "Efficient Ordered Combinatorial Semi-Bandits for Whole-Page Recommendation". In: *AAAI*. 2746–2753.

Wen, Z., A. Ashkan, H. Eydgahi, and B. Kveton. 2015. "Efficient learning in large-scale combinatorial semi-bandits". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. 1113–1122.

Williamson, S. F., P. Jacko, S. S. Villar, and T. Jaki. 2017. "A Bayesian adaptive design for clinical trials in rare diseases". *Computational Statistics & Data Analysis*. 113: 136–153.

Wu, H. and X. Liu. 2016. "Double thompson sampling for dueling bandits". In: *Advances in Neural Information Processing Systems*. 649–657.

Wu, Q., H. Wang, Q. Gu, and H. Wang. 2016. "Contextual Bandits in a Collaborative Environment". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '16*. Pisa, Italy: ACM. 529–538. DOI: 10.1145/2911451.2911528.

Xing, Z., X. Wang, and Y. Wang. 2014. "Enhancing Collaborative Filtering Music Recommendation by Balancing Exploration and Exploitation". In: *Ismir*. 445–450.

Yue, Y., J. Broder, R. Kleinberg, and T. Joachims. 2012a. "The k-armed dueling bandits problem". *Journal of Computer and System Sciences*. 78(5): 1538–1556.

Yue, Y. and C. Guestrin. 2011. "Linear submodular bandits and their application to diversified retrieval". In: *Advances in Neural Information Processing Systems*. 2483–2491.

Yue, Y., S. A. Hong, and C. Guestrin. 2012b. "Hierarchical exploration for accelerating contextual bandits". In: *Proceedings of ICML*.

Yue, Y. and T. Joachims. 2009. "Interactively optimizing information retrieval systems as a dueling bandits problem". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 1201–1208.

Yue, Y. and T. Joachims. 2011. "Beat the mean bandit". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 241–248.

Zhao, X., W. Zhang, and J. Wang. 2013. "Interactive collaborative filtering". In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 1411–1420.

Zhou, L. 2015. "A survey on contextual multi-armed bandits". *arXiv preprint arXiv:1508.03326*.

Zhou, L. and E. Brunskill. 2016. "Latent contextual bandits and their application to personalized recommendations for new users". *arXiv preprint arXiv:1604.06743*.

Zoghi, M., Z. S. Karnin, S. Whiteson, and M. De Rijke. 2015a. "Copeland dueling bandits". In: *Advances in Neural Information Processing Systems*. 307–315.

Zoghi, M., T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, and Z. Wen. 2017. “Online learning to rank in stochastic click models”. In: *International Conference on Machine Learning*. 4199–4208.

Zoghi, M., S. A. Whiteson, M. De Rijke, and R. Munos. 2014a. “Relative confidence sampling for efficient on-line ranker evaluation”. In: *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM. 73–82.

Zoghi, M., S. Whiteson, R. Munos, and M. De Rijke. 2014b. “Relative upper confidence bound for the k-armed dueling bandit problem”. In: *Proceedings of ICML*. 10–18.

Zoghi, M., S. Whiteson, and M. de Rijke. 2015b. “MergeRUCB: A Method for Large-Scale Online Ranker Evaluation”. In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining. WSDM ’15*. Shanghai, China: ACM. 17–26. DOI: 10.1145/2684822.2685290.

Zong, S., H. Ni, K. Sung, N. R. Ke, Z. Wen, and B. Kveton. 2016. “Cascading Bandits for Large-Scale Recommendation Problems”. *arXiv preprint arXiv:1603.05359 - Proc. UAI*.