

Bandit Algorithms in Information Retrieval

Other titles in Foundations and Trends® in Information Retrieval

Web Forum Retrieval and Text Analytics: A Survey

Doris Hoogeveen, Li Wang, Timothy Baldwin and Karin M. Verspoor

ISBN: 978-1-68083-350-8

Display Advertising with Real-Time Bidding (RTB) and Behavioural Targeting

Jun Wang, Weinan Zhang and Shuai Yuan

ISBN: 978-1-68083-310-2

Applications of Topic Models

Jordan Boyd-Graber, Yuening Hu and David Mimno

ISBN: 978-1-68083-308-9

Searching the Enterprise

Udo Kruschwitz and Charlie Hull

ISBN: 978-1-68083-304-1

Bandit Algorithms in Information Retrieval

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

now

the essence of knowledge

Boston — Delft



Documents

Document Sharing Hub

Foundations and Trends® in Information Retrieval

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

D Głowacka. *Bandit Algorithms in Information Retrieval*. Foundations and Trends® in Information Retrieval, vol. 13, no. 4, pp. 299–424, 2019.

ISBN: 978-1-68083-575-5

© 2019 D Głowacka

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Foundations and Trends[®] in Information Retrieval

Volume 13, Issue 4, 2019

Editorial Board

Editors-in-Chief

Maarten de Rijke
University of Amsterdam
The Netherlands

Yiqun Liu
Tsinghua University
China

Editors

Ben Carterette
University of Delaware

Charles L.A. Clarke
University of Waterloo

Claudia Hauff
Delft University of Technology

Diane Kelly
University of Tennessee

Doug Oard
University of Maryland

Ellen M. Voorhees
*National Institute of Standards and
Technology*

Fabrizio Sebastiani
Consiglio Nazionale delle Ricerche, Italy

Hang Li
Bytedance Technology

Ian Ruthven
University of Strathclyde, Glasgow

Jaap Kamps
University of Amsterdam

James Allan
University of Massachusetts, Amherst

Jian-Yun Nie
Université de Montreal

Jimmy Lin
University of Maryland

Leif Azzopardi
University of Glasgow

Lynda Tamine
University of Toulouse

Mark D. Smucker
University of Waterloo

Rodrygo Luis Teodoro Santos
Universidade Federal de Minas Gerais

Ryen White
Microsoft Research

Shane Culpepper
RMIT

Soumen Chakrabarti
Indian Institute of Technology

Tie-Yan Liu
Microsoft Research

Editorial Scope

Topics

Foundations and Trends® in Information Retrieval publishes survey and tutorial articles in the following topics:

- Applications of IR
- Architectures for IR
- Collaborative filtering and recommender systems
- Cross-lingual and multilingual IR
- Distributed IR and federated search
- Evaluation issues and test collections for IR
- Formal models and language models for IR
- IR on mobile platforms
- Indexing and retrieval of structured documents
- Information categorization and clustering
- Information extraction
- Information filtering and routing
- Metasearch, rank aggregation and data fusion
- Natural language processing for IR
- Performance issues for IR systems, including algorithms, data structures, optimization techniques, and scalability
- Question answering
- Summarization of single documents, multiple documents, and corpora
- Text mining
- Topic detection and tracking
- Usability, interactivity, and visualization issues in IR
- User modelling and user studies for IR
- Web search

Information for Librarians

Foundations and Trends® in Information Retrieval, 2019, Volume 13, 5 issues. ISSN paper version 1554-0669. ISSN online version 1554-0677. Also available as a combined paper and online subscription.

Contents

1	Introduction	2
2	Reinforcement Learning and Bandit Algorithms	6
2.1	Reinforcement Learning	6
2.2	What are Bandits?	8
3	Click Models and Bandit Algorithms	15
3.1	Cascade Model	15
3.2	Dependent Click Model	23
3.3	Position-Based Model	26
3.4	Summary	31
4	Ranking and Optimization	33
4.1	Diversifying Ranking with Bandits	33
4.2	Off-line Policy Evaluation	41
4.3	Query Auto-completion and Recommendation	44
4.4	Summary	46
5	Ranker Evaluation	48
5.1	Dueling Bandits and Interleave Filtering	48
5.2	Condorcet Winner	52
5.3	Copeland Dueling Bandits	57

5.4	Multi-dueling Bandits	60
5.5	Pooling Based Evaluation and Bandits	61
5.6	Summary	63
6	Recommendation	64
6.1	Personlization and the Cold Start Problem	64
6.2	Social Networks and Recommender Systems	72
6.3	Collaborative Filtering and Matrix Factorization	78
6.4	Feature Learning with Bandits	82
6.5	Recommendations with a Limited Lifespan	86
6.6	Simultaneous Multiple Arms Evaluation	91
6.7	Summary	94
7	Other Applications	96
7.1	Specialized Short Text Recommendation	96
7.2	Multimedia Retrieval	98
7.3	Web-page Layout	99
7.4	Summary	102
8	Conclusions and Future Directions	104
	Acknowledgements	106
	Appendices	107
A	Algorithms and Methods Abbreviations	108
B	Symbols	110
	References	112

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: 10.1561/15000000067.

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:

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

where $\mu^* = \max_{i=1,\dots,K} \mu_i$ 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:

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

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

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

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

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:

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

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 $y_t = (y_1, \dots, y_{t-1})$ is the column vector of relevance scores, or rewards, received so far, and $X_t = (x_1, \dots, x_{t-1})$ 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 regularization parameter and I is the identity matrix. The arm that maximizes $c_i \cdot y_t + \frac{\alpha}{2} \|c_i\|$ 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 $P(\theta \mid D) \propto \prod P(\rho_i \mid a_i, x_i, \theta)P(\theta)$. 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 θ_i^* . 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 $t = 1, \dots, T$ **do**
 for all $i = 1, \dots, K$ **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 $E = 1, \dots, L$, such as a list of documents. The user traverses a list of K items $A = (a_1, \dots, a_K) \in \Pi_K(E)$, 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 $\prod_{i=1}^{k-1} (1 - w(a_i))$, while the probability that the user finds at least one item attractive is $1 - \prod_{i=1}^K (1 - w(a_i))$. 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 $E = 1, \dots, L$ 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 $A_t = (a_1^t, \dots, a_K^t)$ 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:

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

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  $w_0 \sim P$ 
 $\forall_e \in E : T_0 \leftarrow 1$ 
 $\forall_e \in E : w_1(e) \leftarrow w_0(e)$ 
for all  $t = 1, \dots, n$  do
  Compute UCBs  $U_t(e)$ 
  // Recommend a list of  $K$  items and get feedback
  Let  $a_1^t, \dots, a_K^t$  be  $K$  items with largest UCBs
   $A_t \leftarrow (a_1^t, \dots, a_K^t)$ 
  Observe click  $C_t \in 1, \dots, K$ 
  //Update statistics
   $\forall_e \in E : T_t(e) \leftarrow T_{t-1}(e)$ 
  for all  $k = 1, \dots, \min\{C_t, K\}$  do
     $e \leftarrow a_k^t$ 
     $T_t(e) \leftarrow T_t(e) + 1$ 
     $\hat{w}_{T_t(e)}(e) \leftarrow \frac{T_{t-1}(e)\hat{w}_{T_{t-1}(e)}(e) + \mathbb{1}\{C_t=k\}}{T_t(e)}$ 
  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:

$$w_t(a_k^t) = \mathbb{1}\{k < O_t\} \quad k = 1, \dots, \min\{O_t, |A_t|\}$$

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 items¹.

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 $\theta^* \in \mathbb{R}^{d \times 1}$, which is shared among all items. The assumption is that there exists θ^* such that $w(e) \approx x_e^T \theta^*$ 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.

¹See 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 :

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

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 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:

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

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 $k \in \{1, \dots, K\}$ 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 $A_t = (a_1^t, \dots, a_K^t)$ 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

²Note 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 $A = (a_1, \dots, a_K)$ 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 $1 - \prod_{k=1}^K (1 - v(k)w(a_k))$. 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 $(\kappa_k)_{1 \leq k \leq K}$, 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 $r_{A(t)} = \sum_{k=1}^K Z_k(t)$. 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 $S_e(t) = \sum_{l=1}^K S_{e,k}(t)$, $S_{e,k}(t) = \sum_{s=1}^{t-1} Z_k(s) \mathbb{1}\{A_t(s) = e\}$, $N_e(t) = \sum_{l=1}^K N_{e,k}(t)$, $N_{e,k}(t) = \sum_{s=1}^{t-1} Z_k(s) \mathbb{1}\{A_t(s) = e\}$. The UCB also incorporates bias-corrected versions of the counts: $\tilde{N}_e(t) = \sum_{l=1}^K \tilde{N}_{e,k}(t)$, $\tilde{N}_{e,k}(t) = \sum_{s=1}^{t-1} Z_k(s) \kappa_k \mathbb{1}\{A_t(s) = e\}$. 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(1) 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, $k = 1, \dots, K-1$, 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 Lagr e *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, $\kappa_1 > \kappa_2 > \dots > \kappa_K > 0$ and this order is known. Additionally, $\kappa_1 = 1$. This model involves $E + K$ parameters $\{\theta_i^*\}_{i \in [E]}$ and $\{\kappa_k^*\}_{k \in [K]}$, 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 $\theta_i^* \in (0, 1)$, and each slot $k \in [K]$ is associated with a parameter $\kappa_k^* \in (0, 1]$. 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 algorithm 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 $(1 + \text{mod } E(m), 1 + \text{mod } E(1 + m), \dots, 1 + \text{mod } E(K + m - 1))$, where $\text{mod } E(x)$ 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 $\mu \rightarrow 0$. 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 to 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(1)	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 Capp  , 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).