ALGORITHMS FOR LEARNING FROM ONLINE HUMAN BEHAVIOR AND HUMAN INTERACTION WITH LEARNING ALGORITHMS

by

Gökhan Çapan

B.S., Computer Engineering, Anadolu University, 2010

M.S., Computer Engineering, Anadolu University, 2013

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

I must begin by thanking my family, my father, Seyit Çapan; my mother, Perihan; and my dear sister and brother, Birgül and Murat. I feel lucky to be part of such a family that supported me in every way possible during tough times. They gave me confidence and wholeheartedly supported all my choices, even when they seemed irrational.

I am lucky to have had exceptional advisors. Prof. Taylan Cemgil has mentored me, given me uncommon freedom in early-stage research, and supervised me through developing something I sincerely care about. I must thank Prof. Arzucan Özgür for agreeing to supervise my studies later in the thesis stage. Already has guided me throughout the thesis development, Prof. Özgür, along with Profs. Zehra Çataltepe and Fikret Gürgen provided invaluable support during my time on this work. I would also like to thank the thesis committee members, Profs. Tunga Güngor and Sinan Yıldırım, for reading and reviewing my work and providing valuable feedback.

At PiLab, I have had a great time with great people, Onur Poyraz, Melih Barsbey, Mine Öğretir, Çağlar Hızlı, Burak Kurutmaz, Umut Şimşekli, Çağatay Yıldız, and the other members. I thank each of them very much for convincing me to pursue a Ph.D. simply by being there. Among them, I must mention my chief partners-in-crime, Ali Caner Türkmen and İlker Gündoğdu, and my other collaborators in various studies contributing to this work, Semih Akbayrak, Taha Ceritli, Cağrı Sofuoğlu, and Özge Bozal. This work would not be possible without their contributions. Finally, part of my work is due to collaborations with the open-source community. In particular, I would like to thank all Apache Mahout members, especially Prof. Özgür Yılmazel, for introducing me to the community.

ABSTRACT

ALGORITHMS FOR LEARNING FROM ONLINE HUMAN BEHAVIOR AND HUMAN INTERACTION WITH LEARNING ALGORITHMS

In modern digital systems, algorithms that deliver personalized content shape the user experience and affect user satisfaction, hence long-term engagement with the system. What the system presents also influences the parties providing content to the system since visibility to the user is vital for reachability. Such algorithms learn to deliver personalized content using data on previous user behavior, *e.g.*, their choices, clicks, ratings, etc., interpreted as a proxy for user preferences. In the first part of this work, we review prevalent models for learning from user feedback on content, including our contributions to the literature. As such data is ever-growing, we discuss computational aspects of learning algorithms and focus on software libraries for scalable implementations, including our contributions. The second part is on learning from user interactions with algorithmic personalization systems. Albeit helpful, human behavior is subject to cognitive biases, and data sets comprising their item choices are subject to sampling biases, posing problems to learning algorithms that rely on such data. As users interact with the system, the problem worsens—the algorithms use biased data to compose future content. Further, the algorithms self-reinforce their inaccurate beliefs on user preferences. We review some of the biases and investigate a particular one: the user's tendency to choose from the alternatives presented by the system, putting the least effort into exploring further. To account for it, we develop a Bayesian choice model that explicitly incorporates in the inference of user preferences their limited exposure to a systematically selected subset of items by an algorithm. The model leads to an efficient online learning algorithm of user preferences through interactions.

ÖZET

İNSANLARIN ÇEVRİMİÇİ DAVRANIŞINDAN VE ÖĞRENME ALGORİTMALARIYLA ETKİLEŞİMİNDEN ÖĞRENEN ALGORİTMALAR

Modern dijital sistemlerde, kişiselleştirilmiş içerik sunan algoritmalar kullanıcı deneyimini şekillendirmekte; kullanıcı memnuniyetini ve kullanıcının sistemle uzun süreli ilişkisini etkilemektedir. Sistemin kullanıcıya gösterdikleri, erişilebilirlik için görünürlük çok önemli olduğundan, sisteme içerik sağlayan diğer partileri de etkilemektedir. Bu algoritmalar, kişiselleştirilmiş içerik sunmayı kullanıcı tercihleri için bir gösterge olarak geçmiş davranışlarından, örneğin, seçimlerinden, tıklamalarından, oylarından öğrenirler. Bu çalışmanın ilk kısmında kullanıcıların gösterilen içeriğe verdiği geribildirimden öğrenen yaygın modeller, katkılarımızı da içerecek şekilde incelenmektedir. Bu veri sürekli büyüdüğünden, öğrenme algoritmalarının hesaplama yönüne, ölçeklenebilir gerçekleşmeleri için yazılım kütüphanelerine ve bu alandaki katkımıza odaklanılmaktadır. Çalışmanın ikinci kısmı kullanıcıların algoritmik kişiselleştirme sistemleri ile etkileşimine odaklanmaktadır. Yararlı olsa da, davranış verisi, algoritmalara sorun teşkil edecek şekilde birçok bilişsel önyargı ve örnekleme yanılgısı içermektedir. Kullanıcılar sistemle ilişkide olduğu sürece sorun kötüleşir (algoritma gelecek gösterimlerini yanlı veriden yapar). Dahası algoritma, kullanıcı tercihleri ile ilgili yanlış inancını pekiştirir. Çalışmamız, kullanıcı önyargılarının bazılarını özetlemekte ve biriyle ilgilenmektedir: kullanıcının kendisine sunulan seçeneklerden birini seçme eğilimi. Bu eğilimi gözeten, kullanıcı tercihlerinin çıkarsanmasında onların tüm içeriğin sınırlı bir alt kümesine maruz kaldıklarını hesaba katan, Bayesçi bir seçim modeli geliştirilmiştir. Bu model, kullanıcı tercihlerini sistemle etkileşimlerinden öğrenen etkin bir çevrimiçi algoritmaya olanak vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$\mathcal{B}$	Beta distribution
$\mathcal{BE}$	Bernoulli distribution
$\mathcal{D}$	Dirichlet distribution
$\mathbb{E}[X]$	Mathematical expectation of a random variable X
$\mathcal{G}$	Gamma distribution
$\mathbf{I}$	The identity matrix
$[K]$	Set of positive integeres up to K, $\{1, 2, \ldots, K\}$
$\mathcal{M}$	Multinomial distribution
$\mathcal{N}$	Gaussian distribution
$\mathcal{PO}$	Poisson distribution
$\mathbb{R}^{I \times J}$	Set of real matrices of size $I \times J$
${\mathbb{R}_+}^{I \times J}$	Set of real matrices of size $I \times J$ and elementwise greater than or equal to 0
$\lvert S \rvert$	Cardinality of set S
$\mathbf{x}$	Vector $\mathbf{x}$
$\mathbf{X}$	Matrix $\mathbf{X}$
$\mathbf{x}^\top$	A row vector
$\mathbf{X}^\top$	Matrix transpose
$\mathbf{x_i}$	The i-th row of matrix $\mathbf{X}$
$x_{i,j}$	The i-th row, j-th column of matrix $\mathbf{X}$
$\lvert\lvert\mathbf{x}\rvert\rvert_\mathbf{p}$	p-norm of vector $\mathbf{x}$
Δ^{K-1}	The $K-1$ dimensional probability simplex

LIST OF ACRONYMS/ABBREVIATIONS

ARMS	Adaptive Rejection Metropolis Sampling
BTM	Beat the Mean
CCB	Copeland Confidence Bound
D-TS	Double Thompson Sampling
ECW-RMED	Efficient Copeland Winners Relative Minimum Empirical Divergence
ELBO	Evidence Lower Bound
IIA	Independence of Irrelevant Alternatives
KL	Kullback-Leibler (Divergence)
LDA	Latent Dirichlet Allocation
MF	Matrix Factorization
MM-UCB	Max-Min Upper Confidence Bound
R-UCB	Relative Upper Confidence Bound
RCS	Relative Confidence Sampling
RMED	Relative Minimum Empirical Divergence
SAVAGE	Sensitivity Analysis of Variables for Generic Exploration
SGD	Stochastic Gradient Descent
SMC	Sequential Monte Carlo
SVI	Stochastic Variational Inference
TS	Thompson Sampling
UCB	Upper Confidence Bound
VAE	Variational Autoencoder

1. INTRODUCTION

In modern digital applications, algorithms that deliver personalized content shape the user experience and affect user satisfaction, hence long-term engagement with the system. What the system presents also influences the parties providing content to the system since visibility to the user is vital for reachability. Such algorithms learn to deliver personalized content using data on previous user behavior, *e.g.*, their choices, clicks, or ratings on the earlier content/items, interpreted as a proxy for user preferences, naturally collected through user-algorithm interactions. This work is about models and algorithms for learning from such data, *online user behavior*, and *human interaction with systems* that learn from their behavior.

For the first part, we will first review prevalent models for learning from user feedback on content, including our contributions to the literature. As such data is ever-growing, a discussion on the computational aspects of learning algorithms follows the first part. We will then focus on software libraries for scalable implementations of the algorithms to learn such models and list our contributions. The models we will review are several variants of latent factor models and their extensions, where a small embedding vector represents a user. A linear (in the case of matrix factorization models such as in [1]) or non-linear (in the case of 'deep' models such as in [2]) mapping of the user embedding underlies the user preferences on all possible items. Depending on the kind of observations, these models assume user preferences to underlie observed behavior through an observation likelihood: usually, a Gaussian distribution for observed ratings [3], a Poisson for observed counts [4], or a collection of multinomials for a collection of choices [2, 5–7]. The algorithms we will review are scalable implementations of the optimization algorithms maximizing this likelihood (or a variant thereof), executed on modern distributed data processing systems, as in [8].

Albeit a helpful proxy for their preferences, human behavior is subject to cognitive biases, and data sets comprising their item choices are subject to sampling biases. Ac-

cordingly, one should interpret online human behavior carefully when designing learning algorithms, leading us to the second part of this work.

The sampling biases pose problems to learning algorithms as the observed data set does not constitute a random sample for all user-item pairs [9, 10]. Such biases might occur due to preferential inclusion of user-item pairs in the data sets, for instance, when the users selectively rate items [9]. They also occur because the user cannot give feedback to items that they did not get a chance to evaluate—due to, for instance, the item being at lower positions on a ranked list. As a result, specifically for "implicit feedback" data sets where the users do not provide explicit feedback on content [11], we cannot assume that the user did not like an item if they did not click on it—because they might be unaware of it. Otherwise, our inference of user preferences will be biased [12]. Indeed, various models exist to address such *exposure* [12] and *position* biases [13].

On the other hand, personalized user experience is shaped interactively in modern digital applications. In a typical scenario, a personalization system, *e.g.*, a recommender system, estimates user preferences based on their previous discrete choices, *e.g.*, their clicks on products, movies, or news articles. However, the options presented to the user, selected by the system in line with the preference estimates for that user, might influence their choices. The user, for instance, might trust the system's recommendations, which might result in an increase of the user's probability of click, compared to the case had the item not been recommended [14, 15]. That is, the observation does not accurately reflect the user's actual preferences. Alternatively, the user might put "the least effort" [16] into exploring all the alternatives and conveniently choose (or tend to choose) from what is recommended by the system.

In other words, the system and its users interact in a feedback loop: the system learns to make future recommendations based on user choices, influenced by the previously recommended alternatives. A consequential source of bias is ignoring the interplay between the user's choice and the system's presentation. The feedback loop,

reinforcing the system's own biased belief, may result in the "filter bubble," an unintentional form of censorship with unexpected economic and societal impact [17]. As a result, a group of users' behavior tends to "homogenize" [18] due to "algorithmic confounding" [18]. A user's interest may even "degenerate" over time due to systematic exposure, leading to "echo chambers" [19].

It should be clear from the above that to interpret user behavior and feedback correctly, recommender systems should model the interactions and not merely their feedback [18]. Furthermore, ignoring the interplay between the user's choice and the system's presentation might violate the objective of "fairness of exposure" [20]. Particularly in recommender systems that learn from implicit feedback, preferences for underrepresented options—those rarely presented by the system—might be underestimated. Effectively, the items that are shown less also receive less opportunity to be recommended and, in turn, chosen by the user [21]. Conversely, the system might develop a bias towards *initial choices*—giving them an "early-exposed advantage" [22], or towards *popular* (or *promoted*) alternatives [23], overestimate the preference for these, and keep recommending them.

The onus is on the system to sufficiently explore all the items, and stochastic bandit algorithms [24] offer a viable approach. They are widely used in recommender systems due to their dynamic and interactive nature [25], potentially alleviating some problems mentioned above. For instance, [26] reports that bandit algorithms "break" the feedback loop because they naturally explore to learn and place higher uncertainty around newly introduced items. [27] suggests that bandit algorithms are among the solutions to the filter bubble problem and argues that their *exploration* is crucial for healthier digital markets—for users, publishers, and recommender systems. In [19], the authors show that randomization due to exploration and allowing a "growing pool of items" are necessary to avoid the degeneracy of user interest.

In summary, several biases causing the non-random inclusion of user-item pairs in behavior data and the cognitive biases that influence human behavior pose problems

to learning algorithms that rely on such data sets. As users interact with systems that estimate user preferences from their previous behavior, the problem worsens as the algorithms use biased data to compose future content. Further, when they do not account for user biases and the interplay between the system and its users, the algorithms self-reinforce their inaccurate beliefs on user preferences, also harming the fairness of exposure objective.

In the second part of our work, we will review some of the biases and deeply investigate a particular one: the user's tendency to choose from the alternatives presented by the system, putting the least effort into exploring further. To account for this bias, we will develop a Bayesian choice model, the Dirichlet-Luce model, that explicitly incorporates in the inference of user preferences their limited exposure to content—systematically composed by an algorithm. The model then leads to an online (bandit) algorithm for learning user preferences through interactions.

1.1. Contributions

Throughout developing this thesis, our studies initially revolved around learning and inference algorithms from observed user feedback on content to predict the unobserved/held-out feedback. The main focus was to design matrix factorization variants, extensions, and scalable implementations of the learning algorithms. To this end, the initial set of contributions includes our contributions to open-source software for scalable machine learning [8] and a general matrix factorization variant for modeling the multinomial family of distributions for the observed feedback [28].

As discussed at the outset, our focus shifted towards to novel models for accurate interpretation of online human behavior in their interaction with algorithms—perhaps those designed to learn from online human behavior. Following a review of the biases that make online human behavior data sets unreliable for learning algorithms (thus leading to unhealthy digital platforms that are unfair for both users and content providers), we make the following contributions:

- Focusing on their tendency to choose from what is presented to them—a particular factor that might influence the individuals' choice behavior online, we develop a Bayesian choice model, the Dirichlet-Luce model, that explicitly accounts for limited exposure to alternatives. The observation likelihood we propose conforms to Luce's choice axiom [29]. We specify a full Bayesian treatment through a novel family of distributions, a generalization of the Dirichlet distribution, conjugate to the likelihood. We study our model's properties, and methods for preference estimation and full Bayesian inference. Learning user preferences under the Dirichlet-Luce model is efficient. For example, the model achieves pairwise preference aggregation upon collecting statistics for $K - 1$ distinct pairs of options (where K is the number of all items). In other words, the number of distinct pairs of options that must be presented to users to learn their preferences effectively is linear in the total number of options. Our model also ensures independence of unexplored options—marginal posterior probabilities of choosing options that were never presented are independent of other choices. That is, the model is provably fair to options that were not yet presented or were newly added (*i.e.*, cold-started). A preprint is available in [30]. The model and the algorithm that we will discuss in the next item were also the subjects of our contributed and invited talks in [25] and in [31].
- We then develop an online learning to recommend algorithm, casting the Dirichlet-Luce model as the central component of a bandit algorithm. The algorithm composes recommendations with either frequently preferred or scarcely presented items. In particular, we will first show that the Dirichlet-Luce construction yields a conjugate prior distribution to the restricted multinomial likelihood. We will then develop a sequential Monte Carlo algorithm to sample from the joint probability distribution of preference probabilities. This sequential sampler is used as a subroutine in a widely used bandit algorithm, the Thompson sampler [32], for making recommendations.
- We show by simulation experiments that the Dirichlet-Luce bandit algorithm achieves lower regret than state-of-the-art bandit algorithms in pairwise and L-wise ($L < K$) preference scenarios.

- The Dirichlet-Luce bandit algorithm is shown essential to our setup mainly due to giving enough opportunity for all items to be recommended. Other simulation experiments will mimic challenges to a healthy recommendation machinery by promoting an item or by initially hiding a favorite item, demonstrating that our setup ensures robustness to promotion and discovery of initially censored favorites. In [33], we discuss such aspects in detail. Finally, as first appeared in [34], we describe the model, the algorithm, and discuss many details.

Overall, our contributions explore and extend the literature on models and algorithms and their scalable implementations for learning from online human behavior, admitting that such data is biased and is collected in interaction with learning algorithms. We believe that they include key building blocks assisting practitioners in implementing such algorithms, for instance, when building recommender systems.

1.2. Organization of the Thesis

In Section 2, we give a general background on common applications at digital platforms that use machine learning algorithms from user behavior. Namely, we describe recommender and ranking systems. We then review common modeling approaches for recommender systems, and introduce a matrix factorization variant. We continue the chapter by reviewing learning and inference algorithms on common models for recommender systems. Finally, we review the multi-armed bandit problems for online learning, and describe two commonly used algorithms.

Section 3 focuses on the challenges the practitioner faces when designing recommender/ranking systems. We will first focus on scalability issues. There, we first give scalable variants of the algorithms we review in Section 2. We then provide a short summary on the open-source software libraries that include the scalable implementations. This chapter also highlights our contributions to open-source software. Later in Section 3.2, we review common biases in online human behavior, and that occur in their interactions with learning algorithms. We give a list of well-known biases and the

implications. A particular user bias that we will investigate further is introduced. We will also make a connection with fairness in machine learning in general, and fairness in ranking.

We will deeply investigate a particular bias in Section 4, the least effort put by the user into exploring further than what they are presented, systematically by algorithms. We first set up the notation, and summarize our contributions: a Bayesian choice model and a bandit algorithm derived out of the model. Having set up the notation, we will first review other bandit algorithms for ranking and recommendation. We will then fully specify the proposed choice model, the Dirichlet-Luce model. Later, we will present the Dirichlet-Luce bandit algorithm for online learning to recommend. We will show by simulations the role of the bandit algorithm in addressing potentially harmful biases. Later we will show, again by simulations, why our model specification is useful when, indeed, the users have a tendency to choose from what is presented to them. We then identify a number of bandit algorithms that can be used for similar purposes. We make a set of performance comparisons on simulated data sets of user choices, first based on simulated and then based on an actual set of user preferences extracted from a real-world artist listening data set. The experiments will demonstrate that the model gives superior performance. Section 5 concludes the thesis, discussing the limitations of our work and potential next steps.

2. BACKGROUND

This chapter serves as a reminder for the reader on two topics: typical applications and modeling approaches for learning from user behavior, and bandit algorithms. We do not provide a comprehensive tutorial; we merely provide a short review at a level only sufficient as a background to what follows. Books and comprehensive reviews on these topics will be given as further references.

The first main topic we review here is learning from user behavior. There, we review typical applications related to our contributions, namely, ranking and recommender systems. We then review common modeling approaches for recommender systems, with a particular focus on matrix factorization variants and recent extensions. Finally, for coherence of our presentation, we also include one of our contributions in this section, which is a matrix factorization variant.

Learning from data requires empirical loss minimization or Bayesian inference algorithms, depending on how the models are set up. We will briefly discuss gradient-based optimization algorithms for the first and approximate Bayesian inference algorithms for the second.

Later in our work, we will assert bandit algorithms, sequential decision-making algorithms based on sequential feedback on these decisions, as key to designing online recommender systems. Hence in this chapter, we will briefly describe what a bandit algorithm is. Then, we will discuss basic bandit setups, and describe two widespread bandit algorithms. One of the bandit algorithms we will present, also used later in our work, requires sequential sampling from a maintained posterior distribution updated through interactions. We will use a specific Markov chain Monte Carlo algorithm as a sequential sampler due to [35].

2.1. Ranking and Recommendation

Learning from online user behavior, *e.g.*, their ratings, clicks, or purchases online, is vital to many applications that aim to present relevant content to the users. The two perhaps most frequently deployed applications through inference from user behavior are ranking and recommender systems. In both applications, a small amount of all available content (for instance, out of a total of K documents, a small $L << K$ of them) can be presented to a user at a time. It is unreasonable to expect the everyday user to navigate through all available content, and ranking and recommender systems come to the rescue.

Ranking is central in information retrieval, as [36] puts it, an excellent review that we will follow in our summary. The term usually refers to a system—generally depending on a user query (a proxy for the user's information need), that presents a ranked shortlist of all available content. A ranking can be independent of a query, too. The famous PageRank ordering [37] is an example of websites being assigned static scores independent of a user query. The practice of using machine learning for forming a ranked list is referred to as "learning to rank" [36]. There, a model is learned to map a set of features representing query-document pairs to a relevance judgment, which might be in different forms, depending on the approach chosen.

Historically, relevance judgments are provided by *experts*, forming a training data set for a learning to rank algorithm. That is, the experts are given search queries and a list of documents to which the expert provides a relevance judgment. Relevance judgments may be in different forms. In the "pointwise" approach, they assign relevance scores (perhaps binary) to each document for a query. The expert might make "pairwise" decisions (*e.g.*, [38]), where they compare the relevance of document pairs. Finally, they might provide a "listwise" feedback [39, 40], where they specify a partial (or complete) ordering of the documents for the query. Algorithmic rankings are then learned based on the rankings the experts provide. The models are trained such that the relevance judgments from a ranking can be predicted. The task is to return a

ranked list based on the model such that a quality metric is maximized. Commonly used metrics are the mean reciprocal rank, mean average precision, and discounted cumulative gain (see [41], Chapter 8).

For the pairwise and listwise approaches, let us discuss two models, RankNet [42] and ListNet [40]. In RankNet, for each document, a score $o(i) = f(\mathbf{x_i}; \theta)$ is computed, where $\mathbf{x_i}$ represents the feature vector associated with document i, again perhaps in relation to the current query. For a pair of documents i and j, $o(i, j) = o(i) - o(j)$ are calculated, and then used in a classifier where the decision that document i should be ranked higher than document j is given by the sigmoid function: $1/(1 + e^{-o(i,j)})$. The model is trained using previously labeled pairs (by the experts). Here, $f(.; \theta)$ can be any function parameterized by θ, to be fit during training. In [42], the authors use a neural network.

ListNet [40], as the name implies, is an algorithm when listwise feedback is collected. They propose to use the top-one probability as a model to rank a list of documents, where the quantity, for document i, is computed as: $e^{o(i)} / \sum_{k=1}^{K} e^{o(k)}$ in K documents. Note that top-1 probabilities for each document forms a probability distribution. Again, $o(i) = f(\mathbf{x_i}; \theta)$. The model is trained against the judgment on documents (relative to the other items in a list of documents) associated with a query, collected by experts, or inferred from user logs. Several learning to rank models can be fit to data using optimization algorithms, and with differentiable objectives, they can be run automatically using tools that provide automatic differentiation capabilities (see a survey in [43]). *TF-ranking* [44] is an example where RankNet and ListNet, among other models, are readily available.

That said, expert judgments are not easy to collect, and they make personalized ranking impractical due to the lack of personal preferences data. Hence the recent approach is to rely on actual user interactions, for instance, the click logs in response to historical sessions, and train learning to rank models based on such a data set. Learning from user interactions is convenient, as they are readily available, directly reflect (or

indicate) user preferences, and render personalization approaches applicable. Learning from user interactions, however, has its challenges. As [45] puts it, the feedback is implicit and noisy; hence we apply probabilistic models. Perhaps a more severe problem is that human behavior is biased, and the data sets due to user interactions are subject to sampling biases. Several click models [13] and learning approaches (for instance, bandit algorithms for learning to rank) have been developed to tackle these issues. We will discuss such biases and click models in Section 3.2 and solution approaches in Section 4. Another problem is the abundance of user feedback and the need for scalable algorithms for learning from user interactions. Section 3.1 discusses the scalability issues.

Recommender systems are treated slightly differently. We typically do not have an explicit query. Instead, only a user's historical behavior is used to identify the most relevant, personalized content. The goal is then to infer user preferences accurately from their previous behavior and present future content in line with their preferences. In that sense, learning a recommender system, too, can be considered as a learning to rank problem, where we map a current user's historical behavior, perhaps combined with other information, to a ranked list of items.

2.1.1. Common Modeling Approaches for Recommender Systems

The prevalent approach to learning a recommender system is through *collaborative filtering*. There, with the assumption that similar users will have similar tastes or will act similarly, the goal is to provide recommendations to a user based on the information as to how other users act. The term "collaborative filtering" was coined in [46], and explained as "that people collaborate to help one another perform filtering by recording their reactions to documents they read", in an attempt to filter large volumes of emails into a short, personalized mailing list.

Historically, collaborative filtering systems were modeled based on user or item similarities (see, for instance, [47–49]). In the first case, the users are recommended

items based on a weighted combination of other users' history, where the weights are based on user-user similarities. In the second case, a user is recommended items similar to the items they previously favored, in the sense that the users that favor them largely overlap (in some similarity metric). Later, matrix factorization models were shown superior performance, and collaborative filtering systems largely used matrix factorization variants. More recently, further development was achieved due to advances in deep learning, in particular, deep Bayesian learning. Here we summarize the basic matrix factorization setup and how it varies based on the assumptions made on the observation process. We also briefly summarize a contribution of ours, sum-conditioned Poisson factorization—which first appeared in [28], a convenient factorization approach for observations assumed to follow the multinomial family of distributions. We then summarize a development that is due to variational autoencoders (VAE) [50], and it is included to demonstrate that the model is essentially a nonlinear extension to matrix factorization.

2.1.1.1. Matrix factorization for collaborative filtering. Matrix factorization generally refers to decomposing a typically incomplete matrix into multiple (usually two) low-rank matrices. Recommender systems data, *i.e.*, the user-item interactions, *e.g.*, user ratings on items, can be organized in a matrix of size $U \times I$, where U denotes the number of users, and I denotes the number of items. Naturally, the matrix is incomplete. It is also very sparse due to users rating only a small subset of all items.

Matrix factorization as a model for collaborative filtering has been made popular in a blog post [51], where the post has shown that this simple model gives comparable performance to those of top performers at the Netflix prize, an open recommender system challenge for researchers with a million-dollar award [52]. The received signal is expressed as

The idea is simple. Let $\mathbf{X}$ be a matrix of size $U \times I$, where $x_{u,i}$ contains the rating that the user u gave for the item i. $x_{u,i}$ is otherwise missing. We assume,

$$\mathbf{X} \approx \mathbf{W}\mathbf{H}^\top, \tag{2.1}$$

where $\mathbf{W}$ and $\mathbf{H}$ are $U \times R$ and $I \times R$ matrices, and $R << \min(U, I)$. That is, $\mathbf{X}$ can be approximately written as the product of two low-rank matrices. The assumption that $\mathbf{X}$ is approximately low-rank implicitly encodes the idea that a user's history can be written as a linear combination of others, following the collaborative filtering idea. Note that a row of $\mathbf{X}$, denoted with $\mathbf{x_u^\top}$, can be written as $\mathbf{Hw_u}$, where $\mathbf{w_u}$ is an embedding vector representing a user. That is, assuming an R-dimensional embedding per user, a linear mapping underlies all of their ratings.

Such a model assumes a multi-aspect collection of historical actions for an individual. These latent aspects of behavior can be understood from a collection of individuals: when the behavior of the entire community is cast as a matrix and the low-rank structure is assumed, latent decompositions can be inferred such that things typically done together are favored in a latent theme (a template), but still, an individual is allowed to combine multiple of these themes in different degrees (embeddings, or excitations).

From a probabilistic perspective, we assume that $\mathbf{WH}^\top$ parameterizes a probability distribution from which we observe the elements of $\mathbf{X}$'s. That is, we say, for instance, an $x_{u,i}$ is an independent observation that follows a Gaussian distribution with mean $\mathbf{w_u^\top h_i}$, where $\mathbf{w_u}$ and $\mathbf{h_i}$ are the uth and ith rows of the matrices $\mathbf{W}$ and $\mathbf{H}$, respectively.

$\mathbf{W}$ and $\mathbf{H}$ are parameters for the model. Assuming a normal distribution for the observation likelihood, maximizing the (logarithm of the) likelihood corresponds to the following optimization problem,

$$\text{minimize } ||\mathbf{X} - \mathbf{WH}^\top||_2^2, \tag{2.2}$$

with respect to $\mathbf{W}$ and $\mathbf{H}$. When prior belief is imposed on $\mathbf{W}$ and $\mathbf{H}$, the maximum-a-posteriori estimate is given by optimizing the regularized objective. In [3], this is how the Gaussian matrix factorization model is built. One can also be interested in the posterior probability of $\mathbf{W}$ and $\mathbf{H}$, *i.e.*, $p(\mathbf{W}, \mathbf{H} \mid \mathbf{X}, \alpha)$, where α is a set of hyperparameters. We will deal later with the solution approaches for the optimization

problem or how the posterior inference for $\mathbf{W}$ and $\mathbf{H}$ can be computed.

2.1.1.2. Other observation likelihoods. Much of our contributions are on the interpretation of observed user behavior through a likelihood. Here, we will review two alternative approaches to Gaussian matrix factorization and introduce a generic one. Our particular interest will be in discrete data sets.

Recall that in the Gaussian MF model in [3], we assume that $x_{u,i}$'s are conditionally independent, and they follow a Gaussian distribution parameterized by the dot product of $\mathbf{w_u}$ and $\mathbf{h_i}$,

$$p(\mathbf{X} \mid \mathbf{W}, \mathbf{H}, \sigma^2) = \prod_u \prod_j [(u,i) \in R] \mathcal{N}(x_{u,i}; w_u^\top h_i, \sigma^2), \tag{2.3}$$

where the variance σ^2 is fixed. $[P]$ denotes the *Iverson bracket*, *i.e.*, the function which evaluates to 1 when P is true, 0 otherwise. Here, R is a set of tuples (u,i) for which $x_{u,i}$ are observed.

On the other hand, *Poisson matrix factorization* can be used when we observe interaction counts. An example is when the data set includes how many times a user listened to a song. In [4], the following generative model is introduced, where the maximum likelihood procedure recovers the multiplicative update algorithm introduced in [53] for nonnegative matrix factorization (decomposition of a nonnegative matrix into two nonnegative matrices, *i.e.*, $\mathbf{X} \in \mathbb{R}_+^{U \times I}, \mathbf{W} \in \mathbb{R}_+^{U \times R}, \mathbf{H} \in \mathbb{R}_+^{I \times R}$),

$$\begin{aligned} w_{u,r} &\sim \mathcal{G}(\alpha_w, \beta_w), \\ h_{i,r} &\sim \mathcal{G}(\alpha_h, \beta_h), \\ s_{u,i,r} &\sim \mathcal{PO}(w_{u,r} h_{i,r}), \\ x_{u,i} &= \sum_{r=1}^{R} s_{u,i,r}. \end{aligned}$$

Here, $\mathcal{G}(\alpha, \beta)$ denotes the gamma distribution with shape and rate parameters α and β, respectively.

Note that the model is augmented (with a latent tensor $\mathbf{S}$). Due to the superposition property of the Poisson distribution [54], $x_{u,i}$'s are also Poisson with mean $\mathbf{w_u}^\top \mathbf{h_i}$. Others, for instance [55,56], directly wrote the Poisson MF model as $x_{u,i} \sim \mathcal{PO}(\mathbf{w_u}^\top \mathbf{h_i})$, where again, independent gamma priors are assumed for the entries of $\mathbf{W}$ and $\mathbf{H}$. Poisson MF has been also used for implicit feedback data when the user-item interactions are interpreted as binary signals, *e.g.*, in [56] for click data. That said, prior to the common usage of Poisson, researchers assumed some variants of Gaussian likelihood for implicit feedback data sets, too [11, 57], applying some heuristics on how missing data will be interpreted.

An implicit feedback data set can alternatively be interpreted as a collection of choices. That is, when a user consumes (or clicks, or reads) an item, we interpret the click as a choice over other alternatives, assuming a multinomial likelihood. A popular model reminiscent of matrix factorization for a data set of discrete collections is *latent Dirichlet allocation* (LDA) [6]. [6] defines the generative model of LDA as follows (adapted for modeling user-item interactions):

(i) Sample preference themes, $\beta_r \in \Delta^{I-1}$: $\beta_r \sim \mathcal{D}(\alpha_h)$ for each $r \in \{1, 2, \ldots, R\}$.

(ii) Sample mixture proportions per user, $\theta_u \in \Delta^{R-1}$: $\theta_u \sim \mathcal{D}(\alpha_w)$ for each $u \in \{1, 2, \ldots, U\}$.

(iii) At any time $n \in \{1, 2, \ldots, N_u\}$, sample the choice of user u, $x_{u,n} \in \{1, 2, \ldots, I\}$:

- $z_{u,n} \sim \mathcal{M}(1, \theta_u)$,
- $x_{u,n} \sim \mathcal{M}(1, \beta_{z_{u,n}})$.

Here, Δ^{K-1} denotes the $K-1$ dimensional probability simplex. I, as before, represents the number of items. $\mathcal{D}(\alpha)$ denotes the Dirichlet distribution with concentration parameters α, whereas $\mathcal{M}(1, p)$ denotes the multinomial distribution where the integer number of trials parameter is set to 1, and p denotes the event probabilities for different categories. N_u is the number of choice observations for user u, and sometimes included in the generative process as a random variable following Poisson distribution.

LDA makes a *mixed membership* assumption. That is, the user choices are assumed to be generated from a mixture of multinomials, each of which represents a probability distribution over all alternatives. We will call them *preference themes*. Note that mixed membership differs from clustering, which assumes that a user belongs to one of R clusters. In contrast with clustering, mixed membership is beneficial in fundamentally two ways: i) we can find sharper cluster representations (the probability distributions over all alternatives) since we do not assume every action of a user comes from the same cluster, ii) personalized representations are not restricted to a finite number of stereotypes. The items can be embedded in a low-dimensional (much lower than the number of individuals in the community) space in terms of latent preference themes. A user's behavior can be considered as a mixture whose composition is characterized by random variables over probability simplex. Finally, recently in [58], both Poisson MF and LDA have been shown to belong to a family of models, coined as allocation models.

Sum-conditioned Poisson factorization, first appeared in [28], is a generic model for multinomial family of observations. In a recommender system, it addresses the cases when the the user-item interaction is a bounded integer (*e.g.*, a star rating), a binary signal, or one of a number of categories (*e.g.*, one of add to basket/add to favorites/buy).

To achieve that, we have extended the Poisson factorization in the following way. To factorize a matrix with binary, a bounded integer, or multinomial observations, we define L-component Poisson factorizations. $L = 2$ in the binary or bounded integer case, and for multinomials, L is defined to be the number of different categories. For a component indexed by $l \in \{1, 2, \ldots, L\}$, the corresponding factorization computes $x_{u,i}^{(l)} \mid \mathbf{w_u^{(l)}}, \mathbf{h_i^{(l)}} \sim \mathcal{PO}(\mathbf{w_u^{(l)}}^\top \mathbf{h_i^{(l)}})$, where $\mathcal{PO}(\lambda)$ denotes the Poisson distribution with rate λ. We then constrain the sum $\sum_{l=1}^{L} x_{u,i}^{(l)} = 1$ for the binary case, $\sum_{l=1}^{L} x_{u,i}^{(l)} = n$ for the integer case where the maximum possible integer observation (the cardinality) is set n (for instance, $n = 5$ when we observe star ratings where the user can give at most 5 stars to an item), and $\sum_{l=1}^{L} x_{u,i}^{(l)} = 1$ for the multinomial case. Note that bounded

integers are interpreted as Binomials.

Formally, the sum-conditioned Poisson factorization defines the following generative model,

$$\begin{aligned} w_{u,r}^{(l)} &\sim \mathcal{G}(\alpha_w, \beta_w), \\ h_{i,r}^{(l)} &\sim \mathcal{G}(\alpha_h, \beta_h), \\ s_{u,i,r}^{(l)} &\sim \mathcal{PO}(w_{u,r}^{(l)} h_{i,r}^{(l)}), \\ x_{u,i}^{(l)} &= \sum_{r=1}^{R} s_{u,i,r}^{(l)}. \\ n_{u,i} &= \sum_{l=1}^{L} x_{u,i}^{(l)}. \end{aligned}$$

The model is built on well-known properties of Poisson distribution [54]. Note that $x_{u,i}^{(l)} \mid n_{u,i}$ are Bernoulli, Binomial, or multinomial when $L = 2$ and $N = 1$, when $L = 2$ and $N = n$, and when $L = c$ and $N = 1$, respectively. Here n is the cardinality of bounded integers, and c is the number of categories in the multinomial case.

As specific examples, consider first the factorization of a matrix of zero to five star ratings. We observe for certain (u, i) pairs, $x_{u,i}^{(l)} = x_{u,i}$, the actual star rating. We set $x_{u,i}^{(2)} = n - x_{u,i}^{(2)}$. The missing entries are completed analogously to a matrix factorization. Another example is the binary case. Say, we observe ones and zeros indicating likes and dislikes, and other entries are missing. Again, we set $x_{u,i}^{(1)} = 1$ for $x_{u,i} = 1$, $x_{u,i}^{(1)} = 0$ for $x_{u,i} = 0$, and complement $x_{u,i}^{(2)}$'s according to $x_{u,i}^{(1)} + x_{u,i}^{(2)} = 1$, for (u, i) pairs where $x_{u,i}$ is explicitly observed. Finally, consider the case where an observed $x_{u,i} \in \{1, 2, \ldots, C\}$, one of C categories. We set $x_{u,i}^{(c)} = 1$ whenever $x_{u,i} = c$. We then constrain the sum $\sum_{x=1}^{C} x_{u,i}^{(c)} = 1$.

In our work in [28], the experiments show that the model achieves interpretable results in binary matrix factorization compared to the *logistic matrix factorization*, it achieves superior performance for recommendation from star ratings data compared to

the "ordinal matrix factorization" [59], and the model can be extended to higher-order data sets, *i.e.*, organized in tensors.

2.1.1.3. Variational autoencoders for collaborative filtering. Finally, let us describe a nonlinear extension to matrix factorization, variational autoencoders (VAEs) for collaborative filtering.

VAE refers to a particular method for approximate Bayesian inference with stochastic gradient descent [60], for models often associated with a common architecture, resembling that of autoencoders [50]. The approach paved the way towards very powerful generative models with simpler inference routines. We will discuss variational inference [61], stochastic variational inference [62], and variational autoencoders [50,60] shortly, but let us first focus on the generative model that [2] introduces, and describe why we interpret it as an extension to matrix factorization.

Note that in matrix factorization, the parameters that underlie the observed user behavior on all items can be written as $\mathbf{Hw_u}$, that is, a linear mapping applied to $\mathbf{w_u}$—a vector of factors specific to a user u, or a user embedding. In [2], the following model is assumed to underlie the observed user behavior,

$$\begin{aligned} \mathbf{w_u} &\sim \mathcal{N}(0, I), \\ \phi_u &\propto e^{(f(\mathbf{w_u};\theta))}, \\ \mathbf{x_u} &\sim \mathcal{M}(\phi_u), \end{aligned}$$

where $f(\mathbf{w_u};\theta)$ is a nonlinear function (typically the forward function of a fully-connected, multi-layered neural network), parameterized by θ. In [2], the observation likelihood is chosen as multinomial (as in LDA), but other observation likelihoods are also possible. Hence the model is indeed an extension to matrix factorization, where the user embedding is mapped to the parameters ϕ_u that underlie their behavior through a *nonlinear* function. Although various architectures for collaborative filtering built on

neural networks were criticized in [63], the authors still identified this particular VAE architecture in [2] as the top performing one across many tasks.

2.1.2. Learning and Inference

2.1.2.1. Empirical loss minimization. A straightforward way of estimating the parameters of a model is to fit to data. That is, we first specify the observation likelihood, such as $p(\mathbf{X} \mid \mathbf{W}, \mathbf{H}, \sigma^2) = \prod_u \prod_j [(u,i) \in R] \mathcal{N}(x_{u,i}; \mathbf{w_u}^\top \mathbf{h_i}, \sigma^2)$ of Gaussian MF [3], and then search for the parameters that maximizes the logarithm of this likelihood. The objective is typically non-concave and it has multiple local maxima.

Then, maximization of the objective is usually carried out with a convex optimization algorithm, or another iterative routine that can be shown to find a local maximum, perhaps where some heuristics are applied in order to explore the search space further to converge to better solutions.

Let us first give an example of iterative routines, *alternating least squares* for the matrix factorization problem when the observation likelihood is assumed Gaussian. The loss in Equation 2.2 is non-convex due to the $\mathbf{W}$ and $\mathbf{H}$ are both unknown. However, when $\mathbf{W}$ or $\mathbf{H}$ is fixed, the problem can be treated as an ordinary least squares problem for the corresponding parameter. In alternating least squares, we simply perform these two operations in iterations in an alternating manner. This example is due to [1]. The algorithm is listed in Figure 2.1.

However, such solutions are not always available, and we resort to convex optimization algorithms, most of which are gradient descent variants. The well-known gradient descent algorithm is a an iterative convex optimization algorithm, where at each iteration t we perform,

$$\theta_t \leftarrow \theta_t - \alpha_t \nabla_{\theta_{t-1}} f(\theta), \tag{2.4}$$

to minimize the objective function $f(\theta)$ parameterized by θ. α_t is a positive real number (at iteration t) known as the learning rate. Here we take the negative of the gradient

Input: $\mathbf{X}$, R

Initialize $\mathbf{W_0} \in \mathbb{R}^{U \times R}$ and $\mathbf{H_0} \in \mathbb{R}^{I \times R}$ randomly

$t \leftarrow 1$

while not converged **do**

$\mathbf{W_t}^\top \leftarrow (\mathbf{H_{t-1}}^\top \mathbf{H_{t-1}})^{-1} \mathbf{H_{t-1}}^\top \mathbf{X}^\top$

$\mathbf{H_t}^\top \leftarrow (\mathbf{W_t}^\top \mathbf{W_t})^{-1} \mathbf{W_t}^\top \mathbf{X}$

$t \leftarrow t + 1$

end while

Figure 2.1. Alternating least squares algorithm for matrix factorization.

$\nabla f(\theta)$ as a search direction for the next value of θ. The learning rate α_t can be a small positive constant, or can be found via exact or backtracking line search. The algorithm stops when the norm of the gradient, $||\nabla f(\theta)||_2$, is below a small positive constant.

Note that the optimum θ^* is a fixed point for the iteration, as the gradient will be equal to 0 and the iteration will stop. Under certain continuity conditions, the gradient descent algorithm is guaranteed to converge to optimum of a convex optimization problem. It enjoys linear convergence when the learning rate is carefully chosen (with line search). Our brief explanation of the algorithm is due to [64], where an analysis of the convergence can also be found in Section 9.3 of the book.

Stochastic gradient descent (SGD), a variant of the gradient descent algorithm is of widespread use in machine learning. Introduced in the seminal work of Herbert Robbins and Sutton Monro in [65], it is based on a stochastic approximation algorithm for finding the root of a function. Specifically, assume that we want to find the root of a function $f(x)$, that is, we search for the x^* where $f(x^*) = 0$. A convenient way to find the root is by successive approximation, that is, we start with x_1, and then based on the value of $f(x_1)$, to take a small step in the positive (negative) direction when $f(x_1) < 0$ ($f(x_1) > 0$) to get x_2, and so on. The successive approximation takes a step in an iteration in the form: $x_{t+1} = x_t - c_t f(x_t)$. Note that x^* is a

fixed point, as $f(x^*) = 0$. Provided that $f(.)$ is monotonic, a small constant chosen for c_t would ensure convergence to x^*. Stochastic approximation to the described procedure deals with the problem where we cannot evaluate the function $f(x_t)$, but we are given samples $y_t \sim p(Y_t)$, where Y_t is a random variable whose expected value is $f(x_t)$, *i.e.*, $\mathbb{E}Y_t = f(x_t)$. [65] shows that if c_ts are chosen such that $\sum_{t=1}^{\infty} c_t = \infty$ and $\sum_{t=1}^{\infty} c_t^2 = A < \infty$, the procedure converges.

Stochastic approximation generalizes to the gradient descent algorithm, and it is known as stochastic gradient descent. Specifically, consider a random variable Y_t with $\mathbb{E}Y_t = \nabla_{\theta_t} f(\theta)$. Then we can change the gradient descent update iteration in Equation (2.4) with

$$\theta_{t+1} = \theta_t - \alpha_t y_t, \tag{2.5}$$

where $y_t \sim p(Y_t)$. That is, we can change the gradient with its *noisy* version, and the optimization procedure would still work, provided that α_t's satisfy the conditions in [65].

In machine learning, $f(\theta)$ is often a loss function, such as the sum of squared losses in Gaussian MF. Note that the total loss (say, $\mathcal{L}(\mathbf{X}; \theta)$) is usually a sum over the losses computed at n individual data points (*e.g.*, $\mathcal{L}(\mathbf{X}; \theta) = 1/n \sum_i l(\mathbf{x_i}; \theta)$), and gradient is a linear operator (*i.e.*, $\nabla_\theta \mathcal{L}(\mathbf{X}; \theta) = 1/n \nabla_\theta \sum_i l(\mathbf{x_i}; \theta) = 1/n \sum_i \nabla_\theta l(\mathbf{x_i}; \theta)$). Then the gradient of an individual term of the loss function at a randomly picked data point will serve as a sample where the expected value is the actual gradient. We can also randomly pick a few data points (called a minibatch) and take the gradient there to reduce the variance of the stochastic gradient. Let us list here the SGD updates at iteration t (where the minibatch size is 1) that correspond to maximizing the objective in Equation (2.2), the observation likelihood of Gaussian MF,

$$\begin{aligned} w_{u,r} &\leftarrow w_{u,r} + \alpha_t h_{i,r}(x_{u,i} - \mathbf{w_u}^\top \mathbf{h_i}), \\ h_{i,r} &\leftarrow h_{i,r} + \alpha_t w_{u,r}(x_{u,i} - \mathbf{w_u}^\top \mathbf{h_i}), \end{aligned}$$

where α_t is the learning rate. We suppressed the subscripts indicating the iteration number t for $w_{u,r}$ and $h_{i,r}$ to reduce the clutter in the notation. An additional $-\alpha_t \lambda w_{u,r}$ or $-\alpha_t \lambda h_{i,r}$ term is added to the update statements if we apply L_2 regularization (assuming a spherical Gaussian prior with zero mean on $\mathbf{W}$ and $\mathbf{H}$, and maximizing the posterior).

SGD does not enjoy the linear convergence of the gradient descent algorithm. However, one iteration takes significantly lower time ($O(1)$, in fact) than that of gradient descent due to large volumes of data. In fact, [66] makes an important remark. Although the convergence rate of gradient descent is better, computing the gradient is costly for the typical objective in a machine learning task. This phenomenon is largely due to data sets being large. The analysis in [66] shows that SGD is asymptotically more efficient than gradient descent for large data sets. SGD has many variants that ensures further scalability through distributed computation. We will discuss them in Section 3.1.

2.1.2.2. Variational inference. A Bayesian treats the "parameters" of a model as random variables with prior distributions, and studies their posterior distributions conditioned on the observed data. That is, we are interested in $P(\mathbf{W}, \mathbf{H} \mid \mathbf{X})$ in Gaussian MF and Poisson MF, $P(\theta, \beta \mid \mathbf{X})$ in LDA, and so on. $\mathbf{W}$ and $\mathbf{H}$ are assigned a Gaussian distribution as a prior in Gaussian MF, a gamma distribution in Poisson MF. In LDA, θ and β are assumed to follow a Dirichlet distribution.

In other words in Bayesian machine learning, learning corresponds to inference, that is, computing the posterior distributions. For many interesting problems, though, the posterior distribution is intractable, and we resort to approximate inference techniques, or approximate the posterior with samples from posterior distributions obtained via Monte Carlo methods. A great tutorial for approximate inference techniques is in [67].

Here we will focus on a specific approximate inference technique, variational inference [61]. We will follow the review in [68], and the notation therein. In general, assume that we have a model where $\mathbf{x}$ represents a set of observed variables (*i.e.*, data) and $\mathbf{z}$ represents a set of latent variables. For instance, concerning our notation for Poisson MF, the matrix $\mathbf{X}$ is observed, and the matrices $\mathbf{W}$, $\mathbf{H}$, and the tensor $\mathbf{S}$ are latent. In Bayesian inference, we are interested in $p(\mathbf{z} \mid \mathbf{x})$, the posterior distribution of the latent variables conditioned on data. Often, however, $p(\mathbf{z} \mid \mathbf{x})$ is intractable, and we resort to approximate inference or sampling methods. One approximation to the posterior is the variational approximation, where we first design a family of instrumental distributions $\mathcal{Q}$ over latent variables, and then search for the $q^*(\mathbf{z}) \in \mathcal{Q}$ that has the minimum Kullback-Leibler (KL) divergence to $p(\mathbf{z} \mid \mathbf{x})$. Specifically, we minimize the following divergence,

$$\begin{aligned} & D_{KL}(q(\mathbf{z})||p(\mathbf{z} \mid \mathbf{x})) \\ = \; & \mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{z} \mid \mathbf{x})\right] \\ = \; & \mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})\right] + \log p(\mathbf{x}). \end{aligned}$$

$D_{KL}(q||p)$ denotes the KL divergence from q to p, where the expectation is with respect to q. The equivalent objective is to minimize the quantity $\mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})\right]$, since $\log p(\mathbf{x})$ is a constant for the objective (which is an expectation under the q distribution). KL-divergence is non-negative, thus the following holds,

$$\begin{aligned} & D_{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) \\ = \; & \mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})\right] + \log p(\mathbf{x}) \\ \geq \; & 0. \end{aligned}$$

Hence $\mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})\right] \leq \log p(\mathbf{x})$, known as the evidence lower bound (the ELBO). By maximizing $\mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})\right]$, we are maximizing a lower bound to the marginal log-likelihood (or log-likelihood of data). The objective is then,

$$q^*(\mathbf{z}) = \arg\max_{q(\mathbf{z}) \in \mathcal{Q}} \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})\right]. \tag{2.6}$$

Variational inference leads to simple and efficient inference algorithms when the generative model satisfies certain conditions. In a *conjugate* model when the complete conditional distribution of a latent variable (its conditional distribution given the data and other latent variables, *i.e.*, $p(z_i \mid z_{i\not}, \mathbf{x})$) is available, in the same family with its prior $p(z_i)$, which belongs to the exponential family of distributions, and can be obtained by updating the parameters of the prior $p(z_i)$ with statistics computed from $z_{i\not}$ and $\mathbf{x}$, a specific choice of the instrumental $q(\mathbf{z})$ distribution results in a coordinate ascent algorithm to maximize the ELBO. The choice is to pick a factorized distribution $q(\mathbf{z}) = \prod_i q(z_i; \phi_i)$ in the same family with $p(z_i \mid z_{i\not}, \mathbf{x})$, known as the mean-field approximation.

The coordinate ascent algorithm, the iterative routine that updates each $q(z_i)$ in an alternating fashion, works by fixing $q(z_{i\not})$'s, setting the gradient of the objective with respect to the variational parameter ϕ_i to 0, and finally solving for ϕ_i. With the mean field approximation and when the model is conjugate, simple update rules can be recovered [68].

Before we write the coordinate ascent variational inference algorithm, let us change the notation for reasons that will become apparent shortly. We will closely follow the notation in [62], where the stochastic variational inference algorithm (SVI) is introduced. Assume that the variables whose posterior distributions we are interested in can be categorized into *local* and *global* variables. As an example, the mixture proportions per person in LDA, θ_u, and the user embeddings in matrix factorization models, w_u, can be treated as local variables, whereas β and $\mathbf{H}$ are treated as global. Local variables are assumed to be associated with a single data point and the global variables. As [62] describes, a generative model with local and global variables factorizes as

$$p(\mathbf{x}, \mathbf{z}, \beta \mid \alpha) = p(\beta \mid \alpha) \prod_{n=1}^{N} p(x_n, z_n \mid \beta),$$

where n is an index over all N data points (observations). β is global, z_n's are local variables. Note that z_n's are local in the sense that: $p(x_n, z_n | x_{n\not}, z_{n\not}, \beta) = p(x_n, z_n | \beta)$.

We now assume that the conditional distribution of a latent variable given other latent variables and observations is available in the exponential family form,

$$\begin{aligned} p(\beta|\mathbf{x}, \mathbf{z}, \alpha) &= h(\beta) \exp\left[\eta_g(\mathbf{x}, \mathbf{z}, \alpha)^\top t(\beta) - a_g(\mathbf{x}, \mathbf{z}, \alpha)\right], \\ p(z_{n,i}|x_n, z_{n,\not i}, \beta) &= h(z_{n,i}) \exp\left[\eta_l(x_n, z_{n,\not i}, \beta)^\top t(z_{n,i}) - a_l(x_n, z_{n,\not i}, \beta)\right]. \end{aligned}$$

Here, $\eta_g(.)$ and $\eta_l(.)$ are the natural parameter functions for global and local variables, respectively. They are functions of the other variables that we condition on. $t(.)$ is the sufficient statistics, and $a(.)$ is the log partition function. α is the prior parameter for $p(\beta; \alpha)$, implying a conjugacy relationship,

$$\begin{aligned} p(\beta; \alpha) &= h(\beta) \exp\left[\alpha^\top t(\beta) - a_g(\alpha)\right], \\ p(x_n, z_n|\beta) &= h(x_n, z_n) \exp\left[\beta^\top t(x_n, z_n) - a_l(\beta)\right]. \end{aligned}$$

We then pose a factorized approximation for $\mathbf{z}$ and β,

$$q(\mathbf{z}; \beta) = q(\beta; \lambda) \prod_n \prod_i q(z_{n,i}; \phi_{n,i}),$$

with natural local and global variational parameters ϕ_n and λ. These are going to be in the same exponential family of the complete conditionals, *i.e.*,

$$\begin{aligned} q(z_{n,i}; \phi_{n,i}) &= h(z_{n,i}) \exp\left[\phi_{n,i}^\top t(z_{n,i}) - a_l(\phi_{n,i})\right] \\ q(\beta; \lambda) &= h(\beta) \exp\left[\lambda^\top t(\beta) - a_g(\lambda)\right]. \end{aligned}$$

Recall that the coordinate ascent variational inference algorithm, at each iteration, sets the gradient of the objective for each variational parameter, one by one, to 0, and solves for that parameter. For that particular class of models, the solutions are simple. The gradient of the ELBO with respect to λ is 0 when $\lambda = \mathbb{E}_q\left[\eta_g(x, z, \alpha)\right]$. It is 0 with respect to $\phi_{n,i}$ when $\phi_{n,i} = \mathbb{E}_q\left[\eta_l(x_n, z_{n,\not i}, \beta)\right]$.

We can now write the variational inference procedure as a series of alternating updates from conjugacy and conditional independence of local variables as follows:

(i) $\lambda \leftarrow \mathbb{E}_q\left[\eta_g(x, z, \alpha)\right]$, where $\eta_g(x, z, \alpha) = \alpha + \sum_n t(x_n, z_n)$,

(ii) $\phi_{n,i} \leftarrow \mathbb{E}_q\left[\eta_l(x_n, z_{n,\not{i}}, \beta)\right]$, where $\eta_l(x_n, z_{n,\not{i}}, \beta) = t(\beta, z_n, z_{n,\not{i}})$

That is, we match the canonical parameter of the variable of interest to the expected sufficient statistics computed from the conditioning set. This is the general rule for the coordinate ascent algorithm for variational inference, on a class of models described in [62]. A derivation (also for more general cases) can be found in [68]. The reason [62] concerns with this particular class of models is that the SVI algorithm is the first step to scale up the variational inference procedure. Before we go into the details of the stochastic version, let us list the coordinate ascent variational inference algorithm for LDA and Poisson MF.

Recall that for LDA, we have the prior distributions, $\beta_r \in \Delta^{I-1}$; $\beta_r \sim \mathcal{D}(\alpha_h)$ for each $r \in \{1, 2, \ldots, R\}$, $\theta_u \in \Delta^{R-1}$; $\theta_u \sim \mathcal{D}(\alpha_u)$. We already have as the conditional for $z_{u,n} \in \{1, 2, \ldots, R\}$; $z_{u,n} \sim \mathcal{M}(\theta_u)$, and finally we assume the observations $x_{u,n} \in \{1, 2, \ldots, I\}$ follow $x_{u,n} \sim \mathcal{M}(\beta_{z_{u,n}})$. Note that for LDA, θ_u and $z_{u,n}$s are local, β is global. The complete conditionals for θ and β are available as

$$\begin{aligned} \theta_u \mid \theta_{\not{u}}, \beta, \mathbf{Z}, \mathbf{X} &\sim \mathcal{D}(\alpha_w + \mathbf{n}_{\mathbf{z_u}}), \\ \beta_r \mid \beta_{\not{r}}, \theta, \mathbf{Z}, \mathbf{X} &\sim \mathcal{D}(\alpha_h + \mathbf{n}_{\mathbf{z_r}}). \end{aligned}$$

Here $\mathbf{n}_{\mathbf{z_u}}$ is an R-dimensional vector where $n_{z_u,r} = \sum_{n=1}^{N_u}[z_{u,n} = r]$, that is, it includes the number of occurrences in z_ns of each category in the history of the user u. Slightly differently, $\mathbf{n}_{\mathbf{z_r}}$ is an I-dimensional vector where $n_{z_r,i} = \sum_{u=1}^{U}\sum_{n=1}^{N_u}[x_{u,n} = i \wedge z_{u,n} = r]$, that is, the number of times the choice of item i is associated with latent category r, over all users. We then assume a mean field approximation and posit

$$\begin{aligned} z_{u,n} \mid \phi_{u,n} \sim q(.; \phi_{u,n}) &= \mathcal{M}(\phi_{u,n}), \\ \theta_u \mid \gamma_u \sim q(.; \gamma_u) &= \mathcal{D}(\gamma_u), \\ \beta_r \mid \lambda_r \sim q(.; \lambda_r) &= \mathcal{D}(\lambda_r). \end{aligned}$$

The coordinate ascent algorithm gives the following updates at each iteration,

$$\begin{aligned}
\gamma_{u,r} &\leftarrow \mathbb{E}_q[\eta(x_n, z_n, \beta, \alpha_w)], \\
\log \phi_{u,n,r} &\leftarrow \mathbb{E}_q[\eta(x_u, z_{u,\not{n}}, \beta)], \\
\lambda_{r,i} &\leftarrow \mathbb{E}_q[\eta(\mathbf{X}, \mathbf{Z}, \alpha_h)].
\end{aligned}$$

Note that $\mathbb{E}_q[\eta(x_n, z_n, \beta, \alpha_w)] = \alpha_w + \sum_n \phi_{u,n,r}$, $\mathbb{E}_q[\eta(x_u, z_{u,\not{n}}, \beta)] = \mathbb{E}_q[\log \theta_{u,r}] + \mathbb{E}_q[\log \beta_{r,x_{u,n}}]$, and $\mathbb{E}_q[\eta(\mathbf{X}, \mathbf{Z}, \alpha_h)] = \alpha_h + \sum_u [x_{u,n} = i]\phi_{u,n,r}$ are easy to compute.

For Poisson MF, on the other hand, we have $w_{u,r} \sim \mathcal{G}(\alpha_w, \beta_w)$, $h_{i,r} \sim \mathcal{G}(\alpha_h, \beta_h)$ *a priori*. We then define $s_{u,i,r} \sim \mathcal{PO}(w_{u,r} h_{i,r})$ and $x_{u,i} = \sum_r s_{u,i,r}$. Note that further conditioned on $x_{u,i}$, $s_{u,i} \sim \mathcal{M}(x_{u,i}, (w_{u,r} h_{i,r})_{r=1}^R)$. The coordinate ascent variational inference procedure gives the following updates,

$$\begin{aligned}
\log \phi_{u,i} &\leftarrow \mathbb{E}_q[\log w_{u,r}] + \mathbb{E}_q[\log h_{i,r}], \\
a_{u,r} &\leftarrow \alpha_w + \sum_i \mathbb{E}_q[s_{u,i,r}], \\
b_{u,r} &\leftarrow \beta_w + \sum_i \mathbb{E}_q[h_{i,r}], \\
a_{i,r} &\leftarrow \alpha_h + \sum_u \mathbb{E}_q[s_{u,i,r}], \\
b_{i,r} &\leftarrow \beta_h + \sum_u \mathbb{E}_q[w_{u,r}].
\end{aligned}$$

Again, we assume a mean field approximation, where $q(s_{u,i}; x_{u,i}, \phi_{u,i}) = \mathcal{M}(x_{u,i}, \phi_{u,i})$, $q(w_{u,r}; a_{u,r}, b_{u,r}) = \mathcal{G}(a_{u,r}, b_{u,r})$, and $q(h_{i,r}; a_{i,r}, b_{i,r}) = \mathcal{G}(a_{i,r}, b_{i,r})$. For Poisson MF, a problem-specific derivation appears in [4] following the general update rules in coordinate ascent variational inference. Note that in Poisson MF, which variables are going to be considered as local depends on what we consider as an observation. If, for instance, we treat the history of a user u as a single data point, $\mathbf{w_u}$ becomes a local variable.

Let us now move to the SVI algorithm. Recall that the coordinate ascent variational inference algorithm solves for the variational parameters one by one, while fixing

the variational distribution of the others. The authors of [62] propose exactly the same coordinate ascent update for local variables. For the global ones, however, instead of solving for the variational parameter, they suggest to apply a stochastic (natural) gradient update evaluated at one or a minibatch of data points. The procedure has an advantage of updating the global variational parameter online—without a pass over all observations. [62] provides theoretical justification as to how this algorithm works, and we will list here the proposed change in the coordinate ascent variational inference. The update equations for LDA change in SVI as follows, as shown in [62]:

(i) Sample $u \in \{1, 2, \ldots, U\}$ uniformly at random.
(ii) Solve for γ_u and ϕ_u with the coordinate ascent algorithm.
(iii) Update λ: $\lambda^\tau \leftarrow (1-\rho^\tau)\lambda^{\tau-1} + \rho^\tau\,(\alpha_h + U([x_{u,n}=1]\phi_{u,n,r}))$.

Here we update λ as a weighted combination of its previous value and its solution as if we observed the same $\mathbf{x_u}$ U times. ρ^τ is the learning rate at time τ.

For Poisson MF, treating $\mathbf{w_u}$'s as local variables, SVI updates are derived in [69]. As in LDA, $q(s_{u,i,r}; x_{u,i}, \phi_{u,i})$ and $q(w_{u,r}; a_{u,r}, b_{u,r})$ are updated for a randomly picked u, and then we update $q(h_{i,r}; a_{i,r}, b_{i,r})$ as

$$\begin{aligned} a_{i,r} &\leftarrow \alpha_h + U\mathbb{E}_q[s_{u,i,r}], \\ b_{i,r} &\leftarrow \beta_h + U\mathbb{E}_q[w_{u,r}]. \end{aligned}$$

A question naturally arises in an attempt to apply SGD for optimizing the ELBO without model-specific derivations: can we find a random variable whose expected value is the gradient of the ELBO? Recall the variational objective,

$$\arg\max_q \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \phi)],$$

where $\mathbf{x}$ are observed, $\mathbf{z}$ are latent, and posterior distribution of $\mathbf{z}$ is approximated with $q(\mathbf{z}; \phi)$. To optimize this objective with SGD, we need a Monte Carlo estimate for the gradient, *i.e.*, a random variable Y with its expected value equal to the gradient of the

ELBO,

$$\mathbb{E}Y = \nabla_\phi \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \phi)].$$

Note that sampling a $\mathbf{z}^\tau \sim q(\mathbf{z}; \phi)$ and using $\nabla_\phi \left[\log p(\mathbf{x}, \mathbf{z}^\tau) - \log q(\mathbf{z}^\tau | \phi)\right]$ as a stochastic gradient *does not* work—its expectation is not equal to the gradient of the ELBO. As a result, we cannot push the gradient inside the ELBO directly. To circumvent, there are two common *tricks,* the log-derivative trick [70] and the reparameterization trick [50, 60]. We will describe the reparameterization trick. We will assume that the latent variables z_n's are local to the data points indexed by n, and they are the only latent variables we are interested in.

The reparameterization trick works as follows. We reparameterize the latent variables z_n's as deterministic transformations of auxiliary random variables whose distributions do not depend on the variational parameters ϕ. The ELBO can then be written as an expectation with respect to these auxiliary variables, and instead of sampling a z_n, we sample an auxiliary variable and transform it. That is, we first introduce $\epsilon \sim q_\epsilon(\epsilon; \nu)$ where $z = g(\epsilon; \phi)$, such that,

- $q_\epsilon(\epsilon; \nu)$ does not depend on ϕ,
- $g(\epsilon; \phi)$ is differentiable with respect to ϕ,
- $p(z, x)$ and $q(z; \phi)$ are differentiable with respect to z.

Then the ELBO can be written as an expectation with respect to the q_ϵ distribution, and a Monte Carlo estimate for the gradient can be performed by sampling ϵ and then taking the local gradient. The $g(\epsilon; \phi)$ maps a data point x_n and a random $\epsilon^{(\tau)}$ to a random sample from the approximate posterior of the latent variable associated with that data point $z_n^{(\tau)}$ through a transformation, typically a location-scale transformation (for instance, when z_n is assumed Gaussian), or an inverse cumulative distribution function transformation (for instance, when z_n is assumed to follow the exponential distribution). With the reparameterization trick, we can run stochastic gradient descent to optimize the ELBO with respect to ϕ at time τ, as follows:

(i) Sample $n \in \{1, 2, \ldots, N\}$ uniformly at random,

(ii) Sample $\epsilon^{\tau} \sim q(\epsilon; \nu)$,

(iii) Update $\phi_n^{\tau+1} \leftarrow \phi_n^{\tau} + \rho^{(\tau)} \nabla_{\phi_n} [\log p(x_n, z_n^{\tau}) - \log q(z_n^{\tau}, \phi_n)]$.

One of the most appealing properties of VAE is that if we can find a suitable reparameterization and when the differentiability conditions listed above are met, the procedure is amenable to automatic differentiation [43]. Modern numerical computing software that are commonly used for deep learning such as TensorFlow [71] can then be used to automatically learn complex generative models.

In the typical VAE architecture in [50], the variational approximation depends on data, i.e., $q(\mathbf{z_n}|\mathbf{x_n}, \phi)$. Also in [50], the authors observe that we can rewrite the ELBO as

$$\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}(q(\mathbf{z} \mid \mathbf{x}; \phi)||p(\mathbf{z})).$$

In that case what we get is an objective that maximizes $p(\mathbf{x}|\mathbf{z})$ while keeping $q(\mathbf{z}|\mathbf{x}; \phi)$ close to $p(\mathbf{z})$ (in terms of the $KL-$divergence). [50] notes the resemblance of the architecture to that of an autoencoder (see Chapter 14 in [72]), a specific neural network architecture that recovers its input at the output layer through a *bottleneck,* with compositions of functions $\mathbf{z} = \sigma(q(\mathbf{x}; \theta))$ (the encoder) and $\hat{\mathbf{x}} = y(\mathbf{z}; \nu)$ (the decoder). Here, $\mathbf{x}, \hat{\mathbf{x}} \in \mathbb{R}^I$, $\mathbf{z} \in \mathbb{R}^I$ are I and R dimensional vectors, respectively, where $R << I$ (hence the bottleneck), and the objective is to minimize $D(\mathbf{x}||\hat{\mathbf{x}})$, where D is an appropriate divergence. q and y might be neural networks or simply linear mappings, and σ is an activation function. In VAE architecture, $q(\mathbf{z} \mid \mathbf{x}; \phi)$ is known an encoder, whereas the generative model $p(\mathbf{x} \mid \mathbf{z})$ is known a decoder. Furthermore, since we learn a mapping $q(\mathbf{z}|\mathbf{x}, \phi)$, once ϕ is fit, the inference of $\mathbf{z_n}$ is a cheap operation, as it depends on the observed x_n and fit ϕ. We do not need to store an inferred $\mathbf{z_n}$, which can be computed on the fly, which allows *amortized inference* [73]. For example in VAEs for collaborative filtering, the model can make predictions for a new user after they rate a few items without a need for retraining [2].

2.2. Bandit Algorithms

Bandit algorithms are a family of algorithms that dynamically experiment competing "treatments" with a goal to commit to the best-performing one quickly, through (usually stochastic) rewards observed from the experiments. In that sense, they are decision making algorithms through an interactive sequence of previous decisions and responses to these decisions.

William R. Thompson introduced the idea as "the use of data, however meagre, as a guide to action required before more can be collected" [32]. Of course, previous data is used as a guide, as the estimates based on it would not be enough to commit a decision, especially when not much data is available. Bandit algorithms, thus, are concerned as to when to commit, or 'exploit,' an option that is estimated to give the highest reward, upon deciding sufficient 'exploration' of other options is made. [24, 74] are great tutorials that we closely follow.

Many problems can be formulated as bandit problems. In [32], an algorithm later referred to as "Thompson sampling" is proposed as a means of discovering the best treatment in clinical trials. As referred to in [24], they are used in recommender systems, ad placement systems, dynamic pricing applications, and so on. Numerous papers have been written, and here we will refrain from citing thousands of studies as examples.

Bandit problems are of theoretical interest, too. In particular, when a bandit algorithm is designed, the primary concern is that how many suboptimal decisions the algorithm would make until committing the optimal, and how much "regret" it will suffer for decisions that or not optimal. [75] first gave an asymptotic analysis of the regret, and then introduced the widely-used "upper-confidence bound" (UCB) algorithm, which we will describe in the sequel.

Let us first formalize a bandit algorithm. Bandit algorithms give a sequence of decisions indexed by rounds t. Formally, the algorithm, at round t, picks a decision $a_t \in \mathcal{A}$, a set of all possible decisions. A reward $r_{a_t,t}$ is given to the a_t by the environment. The goal of the algorithm is to maximize the reward received from the environment. A common setup is stochastic bandits, where we assume that the rewards follow a probability distribution with unknown parameters for all $a \in \mathcal{A}$.

Perhaps the simplest bandit problem is the multi-armed stochastic bandit problem, described as follows. We assume there are K arms, each $k \in [K]$ where $[K] = \{1, 2, \ldots, K\}$ revealing a stochastic reward based on a set of known univariate distributions per arm, with unknown, but fixed, mean reward parameters μ_k. One of the arms k^* is associated with the highest mean reward $\mu^* = \max_{k\in[K]} \mu_k$. A bandit algorithm, at round t, pulls $k_t \in [K]$, and receives $r_{k_t,t}$, the stochastic reward revealed by the arm k_t at round t. The goal of the algorithm is to maximize the reward, or equivalently, minimize the regret with respect to the optimal algorithm. At round T, the regret is defined as: $R_T = \max_{k\in[K]} \sum_{t=1}^{T} r_{k,t} - \sum_{t=1}^{T} r_{k_t,t}$.

One usually cares about the regret in comparison to the sequence of actions that is optimal in expectation. That is, we are interested in the difference between the total expected reward by the optimal algorithm and the algorithm at hand. [74] defines it as the pseudo-regret, whereas in [24], it is simply called the regret. Then, at round T, the regret is given by,

$$\begin{aligned} R_T &= \max_{k\in[K]} \mathbb{E} \sum_{t=1}^{T} r_{k,t} - \mathbb{E} \sum_{t=1}^{T} r_{k_t,t} \\ &= T\mu^* - \mathbb{E} \sum_{t=1}^{T} r_{k_t,t}. \end{aligned}$$

We now describe the K-armed bandit problem with Bernoulli rewards, the Bernoulli bandit, which we will use as example when describing the UCB and Thompson sampling algorithms. This is a multi-armed bandit problem with K arms. Each arm is associated with a mean reward parameter μ_k. When the arm k is pulled, it reveals

a reward according to $\mathcal{BE}(\mu_k)$, where $\mathcal{BE}$ denotes the Bernoulli distribution. That is, with probability μ_k, the arm reveals 1, with probability $1 - \mu_k$, it reveals 0. An algorithm tries to discover the arm associated with highest mean reward parameter, $k^* = \arg\max_{k \in [K]} \mu_k$, and commit to that arm. Indeed, there are many other bandit formulations for many problems, ranking and recommendation included. We will cover them in Section 4.2.

Several bandit algorithms can be devised. Among them, perhaps two most popular are the UCB algorithm and Thompson sampling, due to their appealing theoretical guarantees and empirical performance. We will now briefly describe these two algorithms on the Bernoulli bandit problem.

2.2.1. Upper-Confidence Bound Algorithm

First introduced in [75] and then simplified in [76], the UCB algorithm is based on the idea that at each round, the algorithm pulls an arm with the highest *u*pper confidence bound value, that is the estimated mean reward for that arm with an additional confidence term. The second term is typically higher for the arms that are not explored much, that is, the algorithm is more optimistic in the face of higher uncertainty.

We will follow [24] for the formalization. For the Bernoulli bandit, the decisions are made based on $UCB_k(t)$, the upper confidence bound value for arm k at time t,

$$UCB_k(t, \alpha) = \hat{\mu}_{k,t} + \sqrt{\frac{\alpha \log t}{2T_k(t)}},$$

where α is a positive real number parameter and $T_k(t)$ denotes the number of rounds the arm k has been pulled up to round t. $\hat{\mu}_{k,t}$ is the estimate for μ_k at time t. The term $\sqrt{\alpha \log t / 2T_k(t)}$ is chosen because $p(\mu_k < \hat{\mu}_k + \sqrt{\log(1/\delta)/2T_k(t)}) \geq 1 - \delta$, following from Hoeffding's inequality [77]. That is, the UCB algorithm adaptively adjusts the confidence $1 - \delta$ with the number of rounds so far. The algorithm plays each arm once, in arbitrary order for the first K rounds, and then at round t, pulls the arm with the highest UCB value, $\arg\max_{k \in [K]} UCB_k(t-1, \alpha)$.

UCB is an intuitive algorithm. Regret analysis can be found in many sources, which we will not cover here—it is out of our scope in this work as we will only show the regret empirically when proposing a bandit algorithm in the sequel.

2.2.2. Thompson Sampling Algorithm

Thompson sampling is a very simple algorithm that dates back to 1933, the seminal work of William R. Thompson [32]. The algorithm starts with a prior belief on the mean rewards of the arms, and at each round, pulls an arm with the posterior probability of being the best arm.

A simple device to action is to sample mean rewards from their posterior distributions, and then to pick the arm corresponding to the highest sample. For the Bernoulli bandit, the convenient choice of prior distribution is the beta distribution due to conjugacy, and the Thompson sampling algorithm becomes as listed in the algorithm in Figure 2.2.

Input: T
Initialize $\alpha_k \leftarrow 1$, $\beta_k \leftarrow 1$ for all $k \in [K]$
for $t = 1$ **to** T **do**
  Sample $\mu_{k,t} \sim \mathcal{B}(\alpha_k, \beta_k)$ for all $k \in [K]$
  Pull the arm with the highest sampled mean reward, $k_t = \arg\max_{k \in [K]} \mu_{k,t}$
  Observe reward r_t to k_t
  if $r_t = 1$ **then**
    $\alpha_{k_t} \leftarrow \alpha_{k_t} + 1$
  else
    $\beta_{k_t} \leftarrow \beta_{k_t} + 1$
  end if
end for

Figure 2.2. Thompson sampling algorithm for the K-armed Bernoulli bandit problem.

A great tutorial on the Thompson sampling algorithm, including its roots and current use is in [78]. Theoretical guarantees for the Thompson sampler remained to be analyzed until 2012. First results can be found in [79] and [80]. Thompson sampling, in fact, was unpopular in the literature prior to the promising results in the empirical evaluation provided in [81]. The algorithm is now widely used for many bandit problems.

Albeit simple in the conjugate case, Thompson sampling might be difficult to implement for problems where posterior distributions for the mean rewards can not be simply updated, or where it is difficult to sample from this posterior. Indeed, several sampling techniques can be used based on the problem. Sampling from the approximate distributions to the posterior were also studied, such as in [82, 83]. That said, the algorithm we will introduce in Section 4.4 is an instance of a sequential Monte Carlo (SMC) algorithm as Chopin described in [35].

3. SOME CHALLENGES IN LEARNING FROM USER BEHAVIOR

Apart from the modeling approaches for what underlies user behavior and how the historical behavior of other users might give an idea on one, two key challenges arise when learning from behavior. The first one we will address is due to the algorithms for learning such models. The second is on the data itself.

Considering modern digital platforms have millions of active users, user behavior data is huge and grows over time. The first challenge we will discuss here is the scalability of the learning algorithms we previously provided a background for, and the implementations of such algorithms on distributed computing infrastructure. We discuss them in Section 3.1.

A second key challenge is on accurate interpretation of online human behavior. Humans have cognitive biases that shape their choices, and the data sets are collected in a feedback loop, *i.e.*, as part of their interaction with algorithms that learn from user behavior and are designed to guide the user's future choices. Section 3.2 discusses such biases and the challenges they pose to the applications that learn from online human behavior.

3.1. Scalability

Let us reiterate what kind of algorithms are used for learning from user behavior. We have focused on the matrix factorization model and its variants. We have listed alternating least squares, gradient descent, and stochastic gradient descent as algorithms for maximum likelihood or maximum a posteriori estimation. For Bayesian inference, we have discussed variational inference, stochastic variational inference, and stochastic gradient variational inference (or variational autoencoders).

A few patterns will arise in running these algorithms in a scalable fashion. Before we discuss them, let us first describe two generic patterns in processing 'big data.' We will first discuss MapReduce-like platforms designed to process large collections partitioned into multiple nodes associated with processors, ideally local to the data [84]. We will call such collections distributed data sets. For instance, the matrix $\mathbf{X}$ that previously denoted the user-item interactions can be partitioned row-wise, where we ensure that the entire history of a group of individuals resides in a node.

Examples of platforms that provide MapReduce-like processing capabilities are available within popular open-source software, such as Apache Hadoop [85] and Apache Spark [86]. Big data platforms might also be designed to process a stream of collections, such as in Apache Flink [87, 88]. Abstracting away the architectural details, big data platforms provide machinery for processing partitions of large collections in parallel, repartitioning an already partitioned collection based on a rule, broadcasting small enough data so that they are locally available to the processors that the partitions reside locally, and accumulating data from different partitions into a global data structure.

As they are designed for processing large collections, higher-order functions available in standard collection processing interfaces are usually provided. We can, for instance, run *maps* (and variants like *flatMap*) or *filters* completely in parallel to get a new and partitioned collection. Likewise, we can collect data from partitions to a node for computing aggregates (such as *sum*, *count*, etc.), repartition data for grouping and processing these groups in parallel, or repartition multiple data sets based on the same rule to join them, and so on. Common distributed processing operations, such as *all-reduce*, can also be implemented in these platforms, for instance, with a series of parallel processing, followed by aggregating the partial results in one processor and then broadcasting the result to all of the processors. These and many other parallel operations for running on partitioned and distributed data can be run using a series of MapReduce-like operations. For such functions, we refer to [89]. In summary, we deal with data $\mathbf{X}$ partitioned into m storage units: $\mathbf{X^1}, \mathbf{X^2}, \ldots, \mathbf{X^m}$. Each partition is a collection of records: $\mathbf{X^j} : \{x_1^j, x_2^j, \ldots, x_{n_j}^j\}$. We also have processors attached

to these storage units. We are allowed to run parallel operations on partitions, and move the records to other locations. The latter, however, is a costly operation (since network is usually bottleneck, and we work with very large data sets), but it is needed for aggregating local results, or repartitioning the data sets.

3.1.1. Scalable Algorithms

We will now discuss how the algorithms we have mentioned in Chapter 2 can be implemented in a scalable manner, with the capabilities of big data frameworks in mind. Let us start with the alternating least squares algorithm. To parallelize the ALS algorithm listed in Figure 2.1, we will first make some assumptions. That is, both $\mathbf{W}$ and $\mathbf{H}$ matrices are assumed to be small enough to fit into the memory of a processing node. The data matrix $\mathbf{X}$ can be partitioned. In [90], the authors observe that to compute $\mathbf{w_u}$, a processor needs to have access to only $\mathbf{x_u}$ and $\mathbf{H}$. That is, the user factors can be computed in parallel when $\mathbf{X}$ is partitioned row-wise, once $\mathbf{H}$ is made available (broadcast). Note that the local operation additionally involves inverting a small, $R \times R$ matrix. Alternating operation (solving for $\mathbf{H}$ while keeping $\mathbf{W}$ fixed) can be performed analogously, if $\mathbf{X}$ is column-wise partitioned, and $\mathbf{W}$ is broadcast to all nodes. Note that once $\mathbf{X}$ is partitioned (perhaps twice), the only communication within a cluster of nodes is to collect and broadcast $\mathbf{W}$ and $\mathbf{H}$ matrices, which we assume to be applied relatively efficiently.

For supervised learning problems in general, gradient descent is amenable to distributed computation [91]. This is due to the common optimization objective in supervised learning, and the linearity of the gradient operator. The common objective is to minimize,

$$\mathcal{L}(\mathbf{X};\theta) = \sum_{n=1}^{N} l(x_n;\theta) + \lambda||\theta||, \tag{3.1}$$

where n is an index over all data points, the first term is the total loss (a sum of individual losses), and the second term is a regularization term (that does not involve data). Since gradient is a linear operator, *i.e.*, $\nabla_\theta \sum_n l(x_n;\theta) = \sum_n \nabla_\theta l(x_n;\theta))$, the

computation of the gradient can be trivially parallelized. That is, when the data is partitioned and the parameters are broadcast, at each partition j, we compute the contribution to the gradient as $\sum_{i=1}^{n_j} \nabla_\theta l(x_i^{(j)}; \theta)$. At each iteration, the gradients from different partitions are collected to a node to perform a gradient descent update, and then the parameters θ are broadcast back to all nodes with their updated values.

Stochastic gradients, too, can be computed in parallel when a SGD step processes a large batch. However, SGD is more commonly used in a platform that allows processing data in a streaming fashion. As a sequential algorithm, each step in SGD requires the previous value of the parameters, hence direct parallelism is not applicable. That said, SGD has several variants for better utilizing the distributed processing frameworks. Let us briefly review two different approaches. In [92], the authors propose to apply SGD to batches of data in parallel, using the current value of the parameters. After local updates on parameters, all results are collected and averaged. The algorithm is efficient in network usage and MapReduce-friendly. The local process carries out the SGD algorithm on a partition, with a fixed learning rate and for T steps. Another approach, which is not MapReduce-friendly, are for environments where local processors have access to a shared memory that holds the parameters. The problem with the shared memory approach is that an update computed based on a data point by a processor requires to *lock* the parameters until the update is performed. In the approach coined as "HOGWILD!" [93], the authors suggest "running SGD in parallel without locks". Such asynchronous SGD approaches are further analyzed in their delay tolerance, *i.e.*, how late an update can be applied on a parameter [94]. A related approach is to keep the shared memory in a cluster of servers—the ParameterServer [95]. Here, again, asynchronous reads and writes to the ParameterServer can be performed by the worker processes that run the learning algorithm, commonly SGD. ParameterServer approach is more general, where the user might choose between fully synchronous or fully asynchronous approaches. Alternatively, the user can define a bound on delays to deploy an algorithm that is in between the two approaches.

The analysis in [93] makes use of the sparsity of the updates, that occurs when the updates computed for a single data point concern only a small fraction of the set of all parameters. A notable parallel implementation of SGD for matrix factorization, utilizing the sparsity of the model fully (without asynchronous updates), is due to [96, 97]. Recall the SGD updates for Gaussian MF for an arbitrary $x_{u,i}$,

$$\begin{aligned} w_{u,r} &\leftarrow w_{u,r} + \alpha_t h_{i,r}(x_{u,i} - \mathbf{w_u}^\top \mathbf{h_i}), \\ h_{i,r} &\leftarrow h_{i,r} + \alpha_t w_{u,r}(x_{u,i} - \mathbf{w_u}^\top \mathbf{h_i}). \end{aligned}$$

That is, observing $x_{u,i}$, we only update $\mathbf{w_u}$ and $\mathbf{h_i}$. When the coordinates do not overlap, such updates can be performed in parallel. This is also true for blocks of the matrix $\mathbf{X}$. Whenever the blocks do not contain user-item pairs that overlap, the SGD updates can be performed in parallel. Precisely, the distributed SGD algorithm proposed in [96] is then:

(i) Pick a set of blocks such that they do not share any rows/columns with one another. This set of blocks is called a stratum.
(ii) Run SGD in parallel on the blocks that the stratum is comprised of.
(iii) Collect resulting $\mathbf{W}$ and $\mathbf{H}$ and broadcast them.

We repeat this procedure until all of $\mathbf{X}$ is processed. [96] suggests clever ways to form a sequence of strata without a need for repartitioning the data, making the process efficient for MapReduce-like environments.

For approximate inference of the models we have discussed, similar patterns of distributed computation can be used. We will briefly cover them. In the generative models that we have discussed, we have had a concept of local and global variables. When the data set is partitioned such that a set of data points associated with corresponding local latent variables forms a partition, the MapReduce-like implementation described for gradient descent also works. The only difference is that in the coordinate ascent variational inference routines, the local nodes do not output gradient informa-

tion, but they output (expectations) of statistics that would contribute to the update of the global variables. An all-reduce pattern is then applied, that is, the global variable is accumulated in a node and then broadcast to all nodes, such that the coordinate ascent algorithm can update the local variables locally and using the global variable. Stochastic variational inference, on the other hand, is essentially a sequential algorithm. Finally, for stochastic gradient variational inference such as in VAEs, everything that we have mentioned for distributed SGD equally applies.

3.1.2. Software Libraries

It is unreasonable to expect from a machine learning scientist to be an expert in distributed computing, too. Indeed, several software libraries have been developed for large scale machine learning, running on top of modern distributed computing platforms for large data sets. Apache Mahout, a "machine learning on big data" project is one of the first examples [8], that we have also contributed to. The initial version of Mahout was designed to work on the MapReduce implementation of Apache Hadoop, which was the primary big data processing platform when the project started. Mahout included many matrix factorization variants, made available through both alternating least squares [90] and stochastic gradient descent. Older collaborative filtering algorithms were made available through distributed computation of user-user or item-item similarities [98]. LDA, too, is available with a distributed implementation of the coordinate ascent variational inference algorithm that we have previously described. Several other machine learning algorithms, based on the all-reduce pattern of gradient descent implementation or approximate inference were made available. Mahout has been used by several leading internet companies, and made available as part of some major cloud computing vendors.

Mahout has later undergone substantial changes when several distributed processing platforms have started to enjoy widespread use, and became more of a distributed matrix computation framework available for different big data platform users, without a change in the code, through a domain specific language [99]. The language, *Sam-*

sara, allows its users to declare machine learning programs in terms of common linear algebra operations, for which distributed computations are provided under the hood. The operations on distributed matrices are optimized, and a logical plan to compute them is created. The plan is then actually run on a distributed computing platform, *e.g.*, on Apache Spark or Apache Flink, depending on the developer's choice.

Other frameworks on MapReduce-like platforms, such as Apache Spark, follow a similar approach with legacy Mahout [100]. They include a limited set of distributed matrix computations and distributed implementations of gradient descent for other machine learning algorithms. The ParameterServer approach, on the other hand, is particularly useful when the amount of parameters is large, and broadcasting is not efficient. This is a common pattern for complex architectures that are widely used in modern deep learning. For instance, a recent large language model maintains 175 billion parameters [101]. Hence the frameworks like TensorFlow has extensive support for distributed machine learning with the ParameterServer approach. That said, TensorFlow also supports all-reduce type distributed machine learning.

3.2. Biases in Online Human Behavior and Their Interactions with Learning Algorithms

Much of the web is powered by algorithms that filter, rank, and finally present a small amount of abundantly available content. As discussed at the outset, such algorithms, possibly in addition to the context signals (*e.g.*, a search query), commonly rely on previous user interactions, such as in the form of star ratings or clicks. A practitioner's task is to design algorithms that compose future presentations using such data.

However, a fundamental problem arises when learning from such data sets. They are unreliable due to user/algorithmic biases. For instance, users selectively click/rate items for many reasons; they tend to click the items that are ranked higher, they trust the system's recommendations and perceive recommended items as more relevant than

they are, they tend to conveniently pick from recommended alternatives or rate as other people do.

Furthermore, the algorithmic systems might amplify these biases, as the users cannot click on things that are not shown to them, or they might tend to pick from a shortlist of items shown by the system. As a result, such systems induce the data sets they are designed to learn from [102].

Although much of the research in recommender/ranking systems focused on prediction accuracy on a held-out data set in the past, recent work has increasingly focused on listing and addressing such biases by assuming new user models, by causal reasoning, or by employing online/bandit learning algorithms, and on novel unbiased evaluation metrics. Here, we summarize some biases in interactive recommender/ranking systems due to user or algorithmic biases.

We refer to [103] for a discussion on the bias on the web. The authors categorize the biases into data and algorithmic bias, relate algorithmic bias with fairness, and finally, argue that a vicious cycle of bias occurs in a user-interactive algorithmic system, such as a recommender system. Data bias, first, includes demographic, geographical, or gender biases, causing the web's content not to match the actual characteristics of internet users. Automatically generated spam/duplicate content also contributes to this kind of bias.

As more systems rely on user interactions rather than expert judgments in learning to rank and recommend, and users interact with algorithmic systems in a (feedback) loop, biases in user-interactive systems deserve particular attention. We will discuss common user biases shortly, but let us first discuss a critical aspect of a typical interactive system, the vicious cycle of bias, as [103] puts it. First, the user feedback on the system's decisions based on its predictions (*e.g.*, recommendations) is biased. Then, based on this biased feedback, the system updates its belief on user preferences. Estimates of user preferences from such data will be biased [9, 104–106] and incon-

sistent [107]. The user, in turn, provides biased feedback to the systematic decision based on biased estimates of user preferences, reinforcing the system's initial, biased belief. This cycle continues in a series of interactions, and the bias gets more severe than initially. [103] refers to this phenomenon as "the system writing its own future." As a result, for instance, a group of users' behavior might tend to homogenize due to algorithmic confounding [18]. A user's interest may even degenerate over time due to systematic exposure, leading to *echo chambers* [19].

In addition to the users, the vicious cycle of biases threatens the other parties of the modern digital ecosystems—the platforms that host content created by various parties and the content-creating parties. For instance, we previously showed that ignoring a particular user bias—the least effort, that is the user's tendency to choose from what is presented to them, causes overestimation of user preference for promoted or initially preferred items. Conversely, the system would underestimate preference to (initially) underrepresented items, and they will not get the opportunity to be considered by the users. [33].

Before we focus on the biases that might occur in user-interactive systems, let us briefly list common scenarios of user interactions that we are interested in. A system might present to the user:

(i) an item for the user to evaluate and then rate,
(ii) an item for the user to click/skip,
(iii) a (possibly ordered) list of items for the user to click one, or opt not to click,

based on a data set of previous user interactions (click, rating, etc.) with the system. However, the existence of a click/rating or its meaning might be influenced by many factors.

3.2.1. Common Biases in Interactive Systems

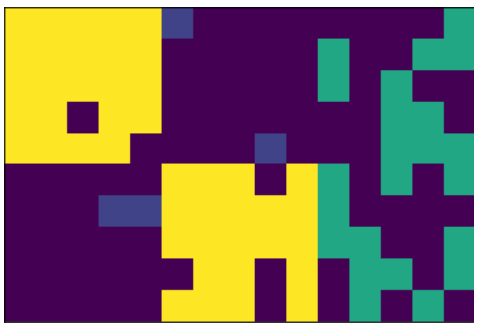
An algorithm that learns from online user behavior should interpret a user click (or rating), or absence thereof accurately for unbiased estimation of user preferences, which requires an understanding of why the users click (or rate). Our review on how users act will partly follow a recent review study by [108] and an older eye-tracking experiment by [14]. We will first review some *sampling biases* that occur when user-item pairs included in training data sets do not constitute a random sample of user-item pairs.

3.2.1.1. Self-selection in ratings. In the context of rating-based recommender systems, self-selection occurs when the users choose which items to consider, and then rate those (and only those) items. In a movie streaming service, for instance, users more likely rate the movies they would like, simply because they watch the movies they think they would like.

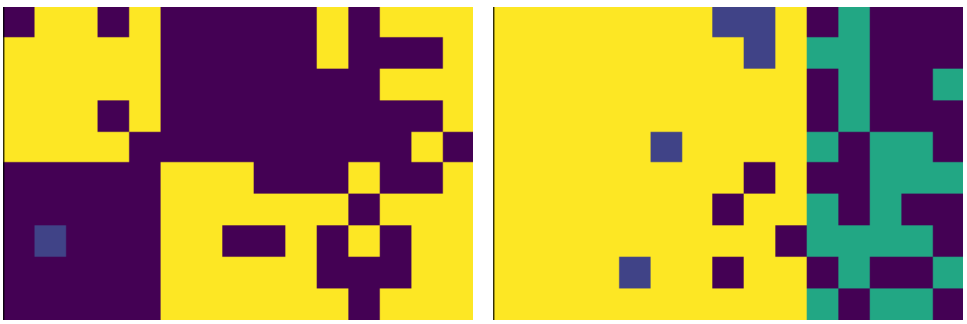
In [9], the authors give the following example that is originally due to [109], which we reiterate here. Say, we have two types of movie watchers: horror lovers, and romance lovers. Also say, we have three types of movies: horror movies, romance movies, and dramas. Horror lovers mostly rate horror movies (and the rating is high), rarely rate romance movies (and the rating is low), and moderately rate dramas (and the rating is moderate). Conversely, romance lovers mostly rate romance movies, rarely rate horror movies, and moderately rate dramas. That is, few ratings that indicate dislike are included in the data set.

For this scenario, consider two alternative recommender algorithms, π_1 and π_2. Assume π_1 predicts for horror lovers, high ratings to horror movies (good), low ratings to romance movies (good), high ratings to dramas (not very bad). Similarly for romance lovers, π_1 predicts low ratings for horror movies, high ratings to romance movies, high ratings to dramas. Alternatively, π_2 predicts for horror lovers, high ratings to horror movies (good), high ratings to romance movies (very bad), moderate ratings to

dramas (good). Similarly for romance lovers, too, π_2 predicts high ratings to horror movies (very bad), high ratings to romance movies (good), moderate ratings to dramas (good). The data set, and the rating estimations by the two algorithms are depicted in Figure 3.1



(a) Rating observations



(b) Algorithm π_1 predictions (c) Algorithm π_2 predictions

Figure 3.1. Observed ratings (a), and the rating estimates by two algorithms π_1 (b), π_2 (c). The darkest color indicates unavailability of ratings. Yellow color indicates a rating of 5, blue color a 1, and green color a 3 (at a 1-5 scale). Evaluation on the available ratings data would find algorithm π_2 more successful.

In a data set that includes the previously mentioned ratings (biased due to self-selection), π_2 would perform better, although we would expect π_1 to be the more preferable recommender algorithm. This is because the loss a 'naive estimator' minimizes, an unweighted aggregate of the deviation between the predicted rating and the true rating, is a biased evaluation of the quality of a recommender algorithm, since the user-item pairs represented in the data set are not randomly included. This kind of selection bias causes the rating matrix being missing not at random [110, 111]. The phenomenon was also shown in real-world experiments. As pointed out in [108], for

instance, [110] showed that in an experiment, when the users choose the movies to rate, they tend to choose the movies they would rate very high or low.

3.2.1.2. Exposure bias. In a data set of user clicks, and assuming they are indeed indicative of positive feedback, the absence of click is ambiguous. We do not know if the user was aware of the item and skipped it, or they were unaware of it, *i.e.*, they were never exposed to it. This phenomenon is commonly referred to as *exposure bias* [12, 57, 104, 112]. Users can only click on the items they see or they know. These might be the items their friends advise, the restaurants they see on their way, or the items shown by the system/application due to an algorithm. Of particular interest, a user might skip an item due to its low position in a ranked list, which we will presently discuss.

3.2.1.3. Position bias. Before we discuss the position bias, let us briefly review click models. In learning to rank [36] literature, the focus has been on relying on user clicks rather than expert judgments, as discussed in Chapter 2. Coming up with models that capture the true nature of the user feedback to a ranked list of items thus became a necessity. Various click models have been proposed, and we review three of them here: the document-based model, the position-based model, and the cascade model [13].

In all three, the click probability of the user can be decomposed into two components: the examination probability and the attractiveness of the item to the user. That is, we can write the probability of click on item i which is at position n in a list of length L, $p(i, n, L)$, as

$$p(i, n, L) = \alpha(i)\chi(n, L),$$

where $\alpha(i)$ is the attractiveness of item i, and $\chi(n, L)$ is solely due to the position of the item.

All three models assume that the user does not examine the items that are not shortlisted in the top-L positions. The document-based click model ignores the effect

of the position of an item within the ranked list on the user's choice. That is, if the item is listed in the first L positions, the probability of click is due to its attractiveness,

$$p(i, n, L) = \alpha(i)[n \leq L],$$

The position-based model assumes that the position of an item in a list affects its probability to be clicked,

$$p(i, n, L) = \alpha(i)\chi(n, L; \eta).$$

Here, $\chi(n, L; \eta) = 0$ if $n > L$. Otherwise, it measures the quality of position k. η is a parameter to χ that is to be learnt.

The cascade model assumes that the user examines the shortlist of items top-down, until clicking an item, where the session ends. In other words, we define,

$$\chi(n, L) = \begin{cases} 1 & n = 1, \\ 0 & n > L, \\ \prod_{j=1}^{n-1}(1 - \alpha(a(j))) & 1 < n \leq L. \end{cases}$$

That is, the examination probability of an item at position n is equal to the probability that the user did not find the first $n - 1$ items attractive. If the user does not find any of the items attractive, the user does not click. In general, the click event can be modeled with a Bernoulli random variable, with click probability decomposed into the examination probability and the attractiveness of the item to the user.

Position bias, in summary, refers to user's tendency to ignore the items ranked lower [113]. That is, regardless of its relevance, the position of an item has an effect on the click event: items ranked lower are more likely to be ignored.

We now turn into another set of biases that might cause a change in the user's actual relevance feedback.

3.2.1.4. Conformity bias. According to [108], conformity bias occurs when a user's feedback does not reflect their actual rating but is affected by other people's opin-

ions. Many examples can be given to the conformity bias. The user might adjust their original, negative feedback to an item after seeing that it is high-rated by other users [114–116], friends and social circles alike [117–120]. Conversely, for instance, in a restaurant delivery service, the user might feel uncomfortable ordering from a recently-opened restaurant. Similarly, they might avoid clicking a cold-started (newly introduced to the system) item in an online marketplace, which would be interpreted as a negative feedback.

3.2.1.5. Trust bias. Note that in common click models, the click probability is due to an item's relevance given that it is examined. In [14], a controlled experiment showed that this assumption is violated. With an eye-tracking study, the authors showed that when we switch positions of two items that are equally examined, the item in the higher position, even when it is less relevant, is more likely to be clicked. Their conclusion is that "the users have substantial trust in the search engine's ability to estimate the relevance of a page, which influences their clicking behavior." This phenomenon is referred to as "trust bias" [15], which occurs when the user's perceived relevance is higher solely because the item is found more relevant by the system.

3.2.1.6. The least effort by the user. Another form of bias where the particular recommendation by the system influences users' click behavior is when the users put little effort into exploring further and tend to conveniently choose one of the items shortlisted for them. We refer to this bias as the least effort, as we can base the assumption on the "principle of least effort" [16]. Here, a click event occurs (i) because the clicked item is recommended by the system and (ii) because the user finds the item (only) relatively more relevant than the other recommended items. Note that in click modeling assumptions, as discussed, the user's preference is not relative once the user sees an item on the list. In that case, the probability of click is equal to the item's marginal click probability.

The least effort by the user is also different from trust bias. Restricting ourselves to the case that the user trusts all of the shortlisted items, the trust bias would translate to an increment in the probability of clicking for all items that are recommended. The least effort, on the other hand, states that a click is relative, turning the interpretation of a click to a combinatorial problem.

3.2.2. Fairness in Ranking and Recommendation

A related notion is fairness in ranking and recommendation. Fairness in machine learning, in general, has gained recent traction. For instance, for supervised learning tasks where automated decisions that affect individuals or 'sensitive' groups, researchers developed several frameworks to formulate a learning task with fairness constraints, or to evaluate whether a system is fair through fairness metrics.

Fairness constraints ensure individual and group fairness, as described as follows. Individual fairness [121] suggests that similar individuals (concerning the classification task at hand) should be treated similarly. Metrics or constraints like "demographic parity" (where a decision must be independent of the protected attribute referring to a group) [122] or "equality of opportunity" (that the classification decision in favor of an individual given the actual class is indeed positive, must be independent of the value of the protected attribute) [123] were explored.

We will now consider a typical, modern digital platform. Multiple parties contribute to the platform: the users, the content providers, and the platform itself. The platform's goal is to provide the most relevant content to its users in the most convenient places across the platform interface. Examples include a streaming service's front page and a search engine's first few results to a query. The onus is on the platform to provide a healthy ecosystem where the users are satisfied and they can reach to relevant content.

How about the *items* that are being ranked? As [20] puts it, modern digital platforms "rank people, products and opinions". An immediate concern is if it would be possible for users to discover them. As argued, and coined as "fairness of exposure" in [20], the platform should concern about the items subject to ranking to be treated fairly. [124] proposes metrics to evaluate if the representation of members of a protected group at top positions and in the overall population are similar, whereas in [20], demographic parity, disparate treatment, and disparate impact constraints are formulated in their fairness of exposure framework. Here, we do not propose measures ensuring individual or group fairness directly. However, we will show that the techniques we will propose in the sequel ensure equal exposure to all items. A platform, in addition, should balance user satisfaction and item exposure, the two goals that sometimes conflict with each other. However, we believe an algorithm can make judgments on the relevance of the items only when they are given enough opportunity to be considered by the users.

There is also a lot more to fairness of exposure. In their recent, critical review in [22], the authors identify numerous fairness concerns. Among them, one that the fairness of exposure would fail to address is the "early-exposed advantage", where an item receives greater attention when presented earlier. Another concern is due to the popularity of an item (see, *e.g.* [23, 125, 126]), where items that are not popular are not given a chance by the users or are not typically favored by the algorithms. We believe that failing to accurately interpret (or model) human behavior is a partial source of such issues, and we will provide simulation experiments as to how such biases that recommender systems are known to be prone to might occur.

4. ADDRESSING THE LEAST EFFORT

Let us now discuss main solution approaches to the problems mentioned in Section 3.2, and then deeply investigate the least effort: the user's tendency to choose from what is presented by the algorithm, formed in line with their previous choices.

As discussed at the outset, any solution approach should start with modeling user behavior (or interpreting the user feedback) accurately. That is, user biases should be correctly encoded. An example is click models, *e.g.*, the position-based or the cascade model, making certain assumptions on how users click based on empirical studies. We should then address the sampling biases in the data set. Two approaches are common: (i) re-weighting the observations, (ii) ensuring exploration (hence exposure) of the items.

The first approach models the probability of evaluating an item, or exposure, and weights the observations with the inverse of this probability. This approach, *inverse propensity score* (IPS) weighting, takes inspiration from causal inference literature [127]. Specifically, because the user-item pairs in the data sets we are interested in are not randomly sampled, we cannot expect to get an idea of how an algorithm would generalize using this training data, *i.e.*, the empirical risk would not serve an unbiased estimate for the true risk. Instead, one should employ a "counterfactual risk minimization" framework [10].

Let us formalize the framework. In [10], the authors contrast empirical and counterfactual risk minimization as follows. In standard supervised learning, we construct a predictor that takes $x \in \mathcal{X}$ as input, and predicts $y \in \mathcal{Y}$, according to an hypothesis $h \in \mathcal{H}$, $h(\mathcal{Y} \mid x)$, that belongs to a hypothesis class $\mathcal{H}$ and given input x, defines a probability distribution over $\mathcal{Y}$. Then, the risk associated with a hypothesis h, $R(h)$, is defined as

$$R(h) = \mathbb{E}\mathcal{L}(h(\mathcal{Y} \mid x), y^*).$$

The expectation is with respect to the joint probability distribution $P(\mathcal{X}, \mathcal{Y})$, and $\mathcal{L}$ is a loss function measuring the discrepancy of the prediction by h with true output y^*. Our goal is to find h that minimizes $R(h)$. Since, in practice, $P(\mathcal{X}, \mathcal{Y})$ is not available, we use a training data set, $(x_n, y_n)_{n=1}^N$, and minimize the *empirical risk* [128],

$$\hat{R}(h) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(h(\mathcal{Y} \mid x_n), y_n).$$

We continue to summarize [10]. Empirical risk minimization works when the training data set includes independent and identically distributed samples according to $(x_n, y_n) \sim P(\mathcal{X}, \mathcal{Y})$, hence $\mathbb{E}\hat{R}(h) = R(h)$. The problem with the interactive setting is that the training data set is collected under a sampling policy P_0. That is, the training set might have been collected using an algorithm which selects x_n, or feedback to certain items might be systematically unavailable due the user's self-selection of what to rate, the position bias, or the exposure bias. Conversely, certain items enjoy abundant real-world feedback data, and they are overrepresented in training data. In that case, we need to 'weight' the training samples by applying importance weights. In other words, while $\mathbb{E}_{P_0}\hat{R}(h) \neq R(h)$, that is, the expected empirical risk under the sampling policy P_0 is not equal to the true risk, but with the following adjustment,

$$\tilde{R}(h) = \frac{1}{N} \sum_{n=1}^{N} \frac{\mathcal{L}(h(\mathcal{Y} \mid x_n), y_n)}{P_0(x_n, y_n)},$$

$\mathbb{E}_{P_0}\tilde{R}(h) = R(h)$. $\tilde{R}(h)$ is an unbiased estimator for $R(h)$, the true risk.

IPS-weighting is appealing due to being able to work on logged data—it does not require an interactive experimentation machinery [10]. The theoretical guarantees in [10] justify many approaches for solving sampling biases. Examples include [9] for self-selection of ratings, [12, 104] for the exposure bias, and [129] for position bias.

The gold standard, however, is by interactive experimentation, ensuring that every item is adequately represented. Such an algorithm faces an exploration-exploitation dilemma. Over the course of interactions, it should explore the alternatives to be included in a recommendation, at the same time, learn to make optimal recommendations. As discussed in Chapter 2, bandit algorithms offer a viable approach. In

ranking/recommender systems, such algorithms can be used to represent each item sufficiently, while making sure that favorable items can be discovered quickly. Section 4.2 reviews some approaches for online learning to rank or recommend.

Before we start the review, let us briefly summarize our contribution on a specific user bias, set up some notation, and list our assumptions. We address the scenario where the users have a tendency to choose one of the recommended alternatives, putting the least effort into exploring further. Thus a choice observation to a recommendation (in the form of a set of items) becomes a *relative* choice to the recommended alternatives. To model relative choice, we will introduce the Dirichlet-Luce choice model, and to make recommendations ensuring sufficient representation for all items, we will derive the Dirichlet-Luce bandit algorithm out of the model.

4.1. Problem Setup

Interpreting implicit feedback as discrete choice observations (assuming a multinomial likelihood as the observation model) has received recent interest in the recommender systems literature. See, for instance, [2, 7] and references therein. However, preference estimates obtained from such models are particularly prone to biased estimates; as all categories of a multinomial random variable are negatively correlated, the increase in the estimate of an item's probability of being preferred inevitably decreases those of others—including the ones that were never recommended. To remedy this, we use a *restricted* multinomial likelihood as the observation model.

Formally, we assume an interactive system where there is a total of K options, but K is large and the user can only be exposed to a limited number of alternatives to choose from at once. That is, the set of *choices* $\{k_t\}_{t=1}^T$ are made from systematically selected subsets of all alternatives—*presentations* $\{C_t\}_{t=1}^T$. Here, $k_t \in [K] = \{1, 2, \cdots, K\}$, and $C_t \in \mathcal{C}$, where $\mathcal{C}$ denotes the set of all non-empty subsets of $[K]$. The user conveniently picks from what was presented. Naturally, $k_t \in C_t, \forall t$.

4.1.1. Choice Model

A discrete choice model specifies the probability of a user choosing an option among K discrete alternatives (or, *items*). We study the probability $p(k \mid C)$ of choosing an item k from a presentation C. We assume a vector of *preferences* θ_k that specifies the probability of choosing an option above all other options, *i.e.*, $\theta_k = p(k|[K])$. We first assume a Bayesian choice model where each choice is multinomial restricted to a presentation. Our choice model conforms to Luce's choice axiom, hence it satisfies *independence of irrelevant alternatives*—choices are probabilistic, and the probability of choosing an option over another is independent of other items in (or, absent from) the presentation [29]. Given a presentation C, it assigns choice probability to an item $k \in C$ in proportion to those of other items in C, as first described in [130] for the case of pairwise preferences, and generalized in [29] and [131] for presentations comprising $L > 2$ options, *i.e.*, $p(k|C) = \theta_k / \left(\sum_{\kappa \in C} \theta_\kappa\right)$.

4.1.2. Online Learning to Recommend

The task of the hypothesized system is two-fold. Armed with a model of choice behavior for the *inference task*, *the discovery task*—as K is so large in practice—is to assume responsibility for finding all good items without overlooking any alternative [132, 133].

That is, the next step is to design a mechanism for online recommendation. Here, the goal is to devise an algorithm for selecting a subset $C \subset [K]$ of size $L < K$ that dynamically infers the preference probabilities of individual items, and learns to recommend the top items, *i.e.*, the most attractive (to the user) L options included in the set.

A common solution, as discussed, is the stochastic bandit algorithms. In contrast to the standard multi-armed bandits setup, where the algorithm pulls one of K arms at each round and observes a stochastic reward feedback, our setup requires

an algorithm that does not merely pull a basic (single) arm, but a subset of dependent arms. Furthermore, the learning algorithm does not receive a reward feedback for the subset presented, but only observes a *winner* (a choice indicating preference over others included in the subset). The setup is a combinatorial bandit with relative feedback [134, 135]. It has been referred to as *dueling bandits* for $L = 2$ [136], also as *battling bandits* for $L > 2$ [137].

4.1.3. Modeling Assumptions

At this point, we need to clarify some aspects of the assumed setup. We will consider a simplistic single user-system interaction scenario. But as a probabilistic building block, the setup can be reused in a more complex, collaborative recommender system. We will ignore the position bias, *i.e.*, the choice probability being dependent on the position of the option within the presentation [113]. We assume that the user is able to review the shortlisted alternatives and make a choice based on their latent preferences.

Our initial setup may seem to ignore the case that the user might *opt not to choose*. Note that we can model this case by a dummy option—assumed to be an element of any recommendation. The dummy option indexed by, say 0, would have a latent probability to be preferred, θ_0. The user could refuse to choose one of the recommended items with probability proportional to θ_0, relative to the choice probabilities of the items included in the recommendation. The model setup would not change, therefore, we will not deal with this case explicitly in our model development. That said, we highlight that assuming this dummy option does not mean that the probability of the user choosing an item in the recommended set would be independent of other items. The choice probability to an item $k \in C$ becomes $p(k \mid C) = \theta_k / \left(\theta_0 + \sum_{\kappa \in C} \theta_\kappa\right)$. This observation is important, as it states that the user tends to choose from what is presented to them unless $\theta_0 \geq \sum_{k \notin C} \theta_k$. In other words, the user would choose from what is recommended to them unless the set of recommended options are too unsatisfactory, an assumption we can base on the the principle of least effort [16].

We will ignore that the repeated and systematic exposure might actually change the users' interests, leading to an *echo chamber*. We refer to [19] for such an analysis, where sufficient conditions that lead to *interest extremes* are provided. That said, it is worth noting that the Dirichlet-Luce model with the associated online recommendation algorithm, as we will presently describe, at least matches two necessary conditions to avoid such degeneracy points, by naturally allowing a "growing pool of items," and due to the randomization inherent in the proposed bandit algorithm.

The reward distributions for the arms will be assumed stationary, *i.e.*, the expected reward of an arm does not change (in particular, decay) over time. In real-world applications, however, this might happen due to multiple reasons. We refer to "time-decaying" [138] or "rotting" bandits [139] that model the decay in the expected reward of an arm via its temporal dynamics [138], or as a function of number of times the arm has been pulled [140]. Another related setup we refer to is the "mortal bandits" [140], where the arms have a random lifetime before they expire.

4.2. A Short Review of Online Learning to Rank and Recommend

Numerous bandit algorithms have been proposed in the recommender or ranking systems literature. They fall into two broad categories. The first strand of research we review here is the family of bandit algorithms where the user provides explicit feedback—whether it is a rating or click feedback, to a recommended item. In such a setup, the development of the bandit algorithm is straightforward, as the setup translates to a multi-armed bandit problem with independent arms.

Such models clearly cannot capture relative feedback, and the user's tendency to choose from what is recommended to them. The studies focus more on modeling the collaborative context, where assuming a clustering [141], co-clustering [142], or a factorization model [143, 144] are common. [141] assumes that there is an unknown number of clusters of users with unknown parameters, which are learned through user-provided feedback to a "context vector" that the algorithm selects. [142] assumes a

clustering of both users and items based on preference behavior, where again, the user gives explicit feedback to the selected context vector. [143, 144] assume a matrix factorization model, where the user provides an explicit rating to the item that is recommended.

The Dirichlet-Luce bandit algorithm, as an *online learning to recommend* algorithm, composes a recommendation as an ordered subset of all items and receives a relative preference feedback. One can think of it as an algorithm for online *learning to rank* [36] with a document-based click (in our case, choice) model, ignoring the effect of the position of an item (within the recommended items) on the user's choice [13]. As discussed in Section 3.2, alternative click models were studied in the information retrieval literature [113]. In both position-based and cascade models, the event to be clicked can be modeled with a Bernoulli random variable, with click probability decomposed into the examination probability and the attractiveness of the item to the user. This assumption is explicit in the position-based model. In the cascade model, the user is assumed to start examining a ranked list of items from the top, and chooses the first item that they find attractive. That is, the user examines an item with the probability that they did not prefer the items that appear higher in the ranking. If the user does not find any of the recommended options attractive, the user does not click.

Many bandit algorithms (or online learning to rank algorithms) were developed assuming an underlying click model, for instance, the cascade model [145–147], the position-based model [148, 149], or a generic click model [150, 151]. The main difference of our setup from the family of bandit algorithms for click models is as described in [145] for those models; that the *weight* of any item is independent of the weights of the other items—an assumption that leads to simple and efficient algorithms. In other words, in click models contrary to our setup, once the user sees an item on the list, the user's preference is no longer relative. Such algorithms, as a result, might overestimate the preference to an item when only relative feedback is provided. As an example, consider the case that an average item is recommended together with inferior items, where the system might develop a misconception that this item is very attractive. Another

undesirable case occurs when a group of average (or inferior) items are recommended. The system, again, would overestimate the user's click when the user has a tendency to choose from what is recommended. An experiment on the performance of a prototypical bandit algorithm for click models can be found in Section 4.5.2.1. That said, the "TopRank algorithm" [151] both shows superior performance to previous online learning to rank models (including the algorithms mentioned here), and the provided algorithm interprets user feedback as relative. In particular, the algorithm maintains pairwise statistics to estimate an order relation, which are fed by the number of times an item is preferred when the other item was present in the ranking. The algorithm then shortlists a number of items created by a topological sort based on this estimate. Moreover, it subsumes various click models. We will thus use the TopRank algorithm as a baseline to measure the effectiveness of our algorithm in online learning to rank with an underlying click model (Section 4.5.2.1).

As discussed in [152], the nature of user feedback plays a major role in deciding the right family of bandit algorithms. For our relative feedback assumption where the user tends to choose from what is recommended, for $L = 2$, a very closely related family of bandit problems to ours is the "dueling bandits" problem [136]—where couples of options $C, |C| = 2$ are presented (a *duel* is set) and the *winner* $k \in C$ observed. For $L > 2$, a generalization appears in [137]. This generalization, however, allows repetition of an option in a presentation, and reduces to a dueling bandit algorithm. They, and also [153–155], found "Double Thompson Sampling (D-TS)" [153] as the best performing algorithm under various feedback models (subsuming ours), which we have also found as the closest competitor. More recently, [135] have described a related bandit problem. There, too, the instantiation of the setup with winner feedback coincides with a dueling bandit setup if the performance is measured based on the attractiveness of the option that is put in the top position. Hence a comparison in performance of the Dirichlet-Luce bandit algorithm with a set of dueling and combinatorial bandit algorithms are provided in Section 4.5.2.2.

4.3. Dirichlet-Luce Choice Model

Let us now formalize the Dirichlet-Luce choice model. Our first step is to write a Bayesian choice model that fully specifies the probability of an option being chosen given a presentation C.

We specify the likelihood of a sequence of choices $k_{1:T}$ conditioned on the sequence of presented subsets $C_{1:T}$ and an underlying vector $\theta \in \Delta$ of choice probabilities ($p$references) among $[K]$, where Δ denotes the $(K-1)$-probability simplex. Namely, we study the *restricted multinomial* likelihood,

$$\begin{aligned} p(k_{1:T} \mid \theta, C_{1:T}) &= \prod_{t=1}^{T} \frac{\theta_{k_t}}{\sum_{\kappa \in C_t} \theta_\kappa} \\ &= \prod_{C \in \mathcal{C}} \frac{\prod_k {\theta_k}^{\nu(k,C)}}{(\sum_{\kappa \in C} \theta_\kappa)^{\mu(C)}} \\ &= \frac{\prod_k {\theta_k}^{y_k}}{\prod_{C \in \mathcal{C}} (\sum_{\kappa \in C} \theta_\kappa)^{\mu(C)}}, \end{aligned} \tag{4.1}$$

where μ and y_k are statistics defined as

$$\begin{aligned} \mu(C) &= \sum_{t=1}^{T} [C_t = C], \\ y_k &= \sum_{t=1}^{T} [k_t = k]. \end{aligned}$$

That is, $\mu(C)$ is the multiplicity of a set C, and y_k the number of times an option k is chosen. These quantities are related, both are marginals of $\nu(k, C) = \sum_{t=1}^{T} [C_t = C] [k_t = k]$ that is defined to be the number of times item k was chosen when C was presented: $\mu(C) = \sum_k \nu(k, C)$ and $y_k = \sum_{C \in \mathcal{C}} \nu(k, C)$.

A Bayesian treatment models θ as a random variable, and posits a prior distribution for it. The restricted multinomial likelihood (4.1) admits the following family

of distributions as a conjugate prior distribution,

$$p(\theta \mid \alpha, \beta) \propto \prod_k \theta_k^{\alpha_k - 1} \prod_{C \in \mathcal{C}} \left(\sum_{\kappa \in C} \theta_\kappa \right)^{-\beta(C)} . \tag{4.2}$$

$\beta(C) \geq 0, \alpha_k > 0$ can be interpreted as the *pseudo-counts* for presentations and choices respectively. These parameters should be constructed as the marginals of a contingency table reflecting the prior belief, hence, $\sum_C \beta(C) = \sum_k \alpha_k$ also holds.

Note that this prior distribution appears as a generalization of the Dirichlet distribution, which is the conjugate prior for the multinomial likelihood. It is easy to see, when $\beta = \beta_0$ where

$$\beta_0(C) = [C = [K]] \sum_k \alpha_k,$$

$p(\theta \mid \alpha, \beta)$ reduces to the Dirichlet distribution $\mathcal{D}(\theta; \alpha)$, as

$$p(\theta \mid \alpha, \beta_0) = 1/B(\alpha) \prod_k \theta_k^{\alpha_k - 1},$$

where $\alpha = \alpha_{1:K}$ is the vector of concentration parameters, and $B(\alpha)$ denotes the multivariate Beta function.

One can adjust α to place prior belief on user preferences analogously to a Dirichlet prior, setting $\beta([K]) = \sum_k \alpha_k$. Prior beliefs on presented subsets can be embedded by adjusting β, together with consistent adjustments on α. The posterior distribution of preferences shares the same structure with the prior due to conjugacy, where the observed statistics y_k and $\mu(C)$ are used to update the prior parameters α_k and $\beta(C)$. Completing our specification of the *Dirichlet-Luce model*, the posterior distribution of preferences upon observing a sequence of interactions follows,

$$p(\theta \mid k_{1:T}, C_{1:T}, \alpha, \beta) \propto \prod_k \theta_k^{\alpha_k + y_k - 1} \prod_{C \in \mathcal{C}} \left(\sum_{\kappa \in C} \theta_\kappa \right)^{-\mu(C) - \beta(C)} . \tag{4.3}$$

If $C_t = [K]$, $\forall t$, the model reduces to the Dirichlet-Multinomial. At another extreme, when the user is obliged to choose from a singleton presentation $\{k\}$, the posterior is not updated, since the update factor cancels as it is θ_k / θ_k.

A special case of the posterior form in Equation (4.3), for pairwise preferences, was first introduced in [156]. Although an inference procedure was not described, the normalizing constant was recognized in a series form later referred to as a "very complicated function of factorials" by [157]. In fact, the complicated part of the normalizing constant is a special hypergeometric function, Carlson's $\mathcal{R}$ function "in bare form" [158],

$$\mathcal{R}(\alpha, \mathbf{Z}, \beta) \equiv \frac{1}{B(\alpha)} \int_{\Delta} \prod_{k} \theta_k^{\alpha_k - 1} \prod_{c} (\theta^\top \mathbf{z}_{:,\mathbf{c}})^{-\beta_c} d\theta,$$

where $\sum_k \alpha_k = \sum_c \beta_c$. $\mathbf{z}_{:,\mathbf{c}}$ denotes the c'th column of matrix $\mathbf{Z}$. We can write the normalizing constant of the Dirichlet-Luce density in Equation (4.2) in terms of Carlson's $\mathcal{R}$ function as follows: first assume an enumeration of $\mathcal{C}$, $\sigma \colon \mathcal{C} \to \{1, 2, \ldots, |\mathcal{C}|\}$, and then construct the indicator matrix $\mathbf{Z} \in \{0, 1\}^{K \times |\mathcal{C}|}$ where $z_{k,\sigma(C)} = 1$ if $k \in C$. Hence $\theta^\top \mathbf{z}_{:,\sigma(C)} = \sum_{\kappa \in C} \theta_\kappa$. Since $\sum_k \alpha_k = \sum_{C \in \mathcal{C}} \beta(C)$ holds *a priori* by construction, and the identity is preserved *a posteriori* due to conjugacy and consistency of sufficient statistics, we have the normalizing constant as $\mathcal{R}(\alpha, \mathbf{Z}, \beta) B(\alpha)$. *A posteriori*, the normalizing constant of the posterior in Equation (4.3) is $\mathcal{R}(\alpha + \mathbf{y}, \mathbf{Z}, \beta + \mu) B(\alpha + \mathbf{y})$.

The Dirichlet-Luce model specifies a distribution on user preferences, explicitly accounting for the fact that the users are exposed to and choose from only a fraction of items which are systematically selected, where the systematic selection [159, 160] acts as a confounding factor for both the future recommendations and the user choices [18]. The Bayesian treatment to preferences and the conjugacy of the model will be shown important in the following sections.

A general form of the proposed density was studied in [161] following [158]. [162] utilized this generalization for Bayesian analysis of multinomial cell probabilities under censored observations, a problem which can be thought of as the inverse of ours. Our model also appears as a special case of the "Hyperdirichlet distribution" studied by [163], where we additionally require consistency in parameterization.

4.3.1. Learning from Preferences

Having specified the likelihood for *recommendation-choice* observations and a conjugate prior distribution of preferences, we now turn into making predictions using the Dirichlet-Luce model. For the hypothesized recommender system, given a set of observations, the primary objective is to predict the choice probability of an option k if all alternatives were available to the user, $p(k_{T+1} \mid [K], k_{1:T}, C_{1:T}, \alpha, \beta)$. Indeed, such quantities of interest are written in terms of the normalizing constant, the $\mathcal{R}$ function.

The conjugacy of the model leads to a compound probability distribution as the posterior predictive distribution for the choice probability of an option k_{T+1} follows,

$$p(k|[K], k_{1:T}, C_{1:T}, \alpha, \beta) = \frac{\alpha'_k}{\sum_j \alpha'_j} \frac{\mathcal{R}(\alpha' + \mathbf{k}, \mathbf{Z}, \beta'')}{\mathcal{R}(\alpha', \mathbf{Z}, \beta')},$$

where $\mathbf{k}$ is the indicator vector of size K where all but the k-th element are 0. Here, $\alpha'_k = \alpha_k + y_k$, $\beta'(C) = \beta(C) + \mu(C)$, $\beta''(C) = \beta'(C)$ for all $C \neq [K]$, and finally, $\beta''([K]) = \beta'([K]) + 1$ (a derivation of posterior predictive distribution is provided as an appendix.)

Computation of the $\mathcal{R}$ function, however, is intractable, as studied in [164]. For the general case, they propose Laplace approximation around the *maximum a posteriori* (MAP) estimate $\theta^* = \arg\max_\theta \log p(\theta \mid \alpha, \beta, k_{1:T}, C_{1:T})$. One can also compute $\mathcal{R}$-related quantities given a batch of observations via sampling by Hamiltonian Monte Carlo (HMC) [165]. The $\mathcal{R}$ function admits efficient computation in some specific conditions [164]. These include the case if the subsets of $[K]$ implied by the columns of Z imply a hierarchy of partitions on $[K]$. Here, we reiterate a key result to highlight an implication for our case.

Lemma 4.1. *[164] Let* $\mathbf{G} \in \mathbb{R}^{K \times C}$, $\mathbf{a} \in \mathbb{R}^K, \mathbf{b} \in \mathbb{R}^C$. *Assume* $\mathbf{b}$ *admits a permutation such that* $\mathbf{b} = [\tilde{\mathbf{b}}\ \mathbf{0}]$ *where* $\mathbf{0}$ *denotes a vector of zeros. Also, let* $\tilde{\mathbf{G}}$ *denote* $\mathbf{G}$ *with columns permuted conformably to* $\mathbf{b}$. *Then,* $\mathcal{R}(\mathbf{a}, \mathbf{G}, \mathbf{b}) = \mathcal{R}(\mathbf{a}, \tilde{\mathbf{G}}, \tilde{\mathbf{b}})$.

This result yields an important implication for approximating $\mathcal{R}$ functions. The dimension of the required integration depends on the sparsity of the arguments.

The efficiency of the procedures for the inference or MAP estimation of preferences are determined by the sparsity of the statistics. However, upon first inspection of the posterior (4.3), one of the first concerns is the dimensionality of the statistic μ. That is, the model informs its knowledge of the preferences θ via statistics collected on a set that scales combinatorially, *i.e.*, the dimensionality of μ can be as high as $2^K - 1$.

This raises a question on the size of the sample required for an accurate preference estimate. The key is to note that our modeling assumptions, *i.e.*, assuming a single preference vector and the Luce choice axiom, lead to the assumption that preferences are (stochastically) transitive, and independence of irrelevant alternatives holds. Furthermore, the posterior is log-concave in a reparameterized version (with one-to-one correspondence) of θ under mild conditions as shown in [166]:

Lemma 4.2. *[166] Take the reparameterization of θ, $\gamma_\kappa = \log \theta_\kappa - \log \theta_1$ for $\kappa \neq 1$ and $\gamma_1 = 0$. The logarithm of the posterior (Equation 4.3) is strictly concave in γ if and only if for each possible partition of $[K]$ into two nonempty subsets K_1 and K_2, there exists $\kappa_1 \in K_1$ and $\kappa_2 \in K_2$ such that $\mu(\{\kappa_1, \kappa_2\}) > 0$.*

For instance, a data set of pairwise choice observations from $\{i, j\}$, for all $j \in [K]$ and $j \neq i$ where i is a fixed, pivot option satisfies the condition. This data set contains only $K - 1$ distinct subsets as recommendations (hence μ is very sparse) and θ^* can easily be recovered by standard convex optimization algorithms, the minorization-maximization algorithms described by [166], or by estimating moments from posterior samples. Note that with a fixed presentation size restricted to $L \geq 2$ the gradient of the log-posterior can be computed in $O(LD + K)$ time, where D is the number of distinct presentations, which is easy to obtain when D is small.

We illustrate that the Dirichlet-Luce construction is *consistent* and is able to recover a good representation of preferences upon choice observations to a small number

of distinct presentations, demonstrated with experiments on synthetic data sets that satisfy the condition in Lemma 4.2, provided as appendices.

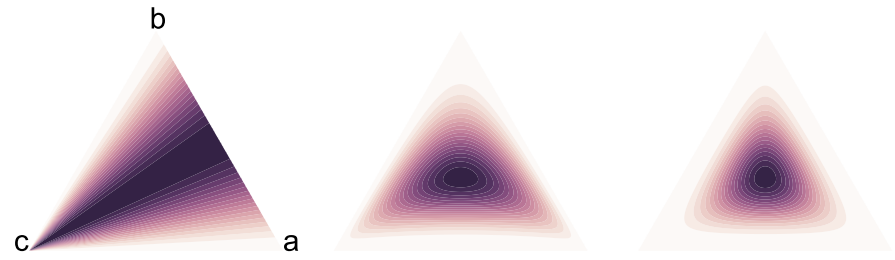
4.3.2. Conflicting Choices

Our previous argument highlights the key ingredient that our model relies on to recover preferences—stochastic transitivity. This observation raises a natural question, what if the transitivity assumption does not hold? That is, how does the model treat conflicting (cyclical) choice behavior?

It is not hard to see that the likelihood is invariant for all realizations with identical μ and y. That is, the posterior ignores the associations between choices and particular presentations. Conflicting choices are treated as *ties* by the posterior, as illustrated in Figure 4.1. These observations suggest a natural next step in utilizing our model. Cyclical choice behavior can be modeled as a mixture of preferences θ. Then, a mixture of Dirichlet-Luce densities can be used to capture different *modalities* of transitive preferences under limited subsets of alternatives, modeling complex choice behavior. There is also further evidence about the plausibility of such an approach in the psychology literature [167, 168]. While we focus on modeling a single, unimodal preference behavior in this article, this direction remains an exciting opportunity for further work.

4.3.3. Independence of Unexplored Options

We now turn into a fundamental advantage of explicitly incorporating in the inference of preferences the users' limited exposure to alternatives: its ability to keep invariant preference estimates of items that were never presented. Particularly, the posterior density keeps posterior marginals of θ_k, where k is an unexplored (never before presented) option, invariant independently of other choices. More formally, assume, without loss of generality, that option $k = 1$ is never presented, *i.e.*, $\mu(C) = 0, \forall C \ni 1$. It then follows, $p(\theta_1 \mid \alpha, \beta_0, k_{1:T}, C_{1:T}) = p(\theta_1 \mid \alpha, \beta_0)$. This is in stark contrast to the



(a) $((a < b, a > b), (b < c, b > c), (c < a, c > a))$

(b) $((a > b, a > b), (b > c, b > c), (c > a, c > a))$

Figure 4.1. Evolution of the exact posterior for $T = 6$ observations ($i > j$ denotes a preference observation i over j). Contours of the joint density are shown on a simplex plot per two preference observations. Cyclic preferences (b) are treated as ties (a).

Dirichlet-Multinomial model, in which choice observations impose negative bias on the marginals of all θ_k, regardless of if they were ever presented. We give an illustration with $K = 3$ options (Figure 4.2). In the illustration, we investigate the posterior marginals of θ_k, $k \in \{1, 2, 3\}$ when a set of choices are made from $\{1, 2\}$. The marginal of θ_3 must stay invariant as in Figure 4.2(c), the Dirichlet-Luce posterior. But if we ignore what is presented, the option 3 is unfairly penalized, as shown in Figure 4.2(b), the Dirichlet-Multinomial.

Let us prove the result by deriving the marginal distribution of posterior choice probabilities for unexplored options.

Lemma 4.3. *(Independence of unexplored options) Assume* $\mu(C) = 0, \forall C \ni \ell$. *It then follows,* $p(\theta_\ell \mid \alpha, \beta_0, k_{1:T}, C_{1:T}) = p(\theta_\ell \mid \alpha, \beta_0)$.

Proof. Assume, without loss of generality, option 1 was never presented. Since with $\beta = \beta_0$, prior θ is Dirichlet distributed with parameter α, prior marginal $\theta_1|\alpha, \beta_0$ is Beta distributed with parameters $(\alpha_1, \sum_{j=2}^K \alpha_j)$. The posterior marginal is

$$p(\theta_1 \mid \alpha, \beta_0, k_{1:T}, C_{1:T}) \propto \int_{T_1} \prod_{j=1}^K \theta_j^{\alpha_j - 1} \prod_{C \in \mathcal{C}} \frac{\prod_{j=2}^K \theta_j^{\nu(j,C)}}{(\sum_{i \in C} \theta_i)^{\mu(C)}} d\theta_{2:K-1},$$

where $T_1 = \{(\theta_2, \cdots, \theta_K) \mid \sum_{j=2}^K \theta_j = 1 - \theta_1, \theta_j > 0\}$.

With a change of variables $u_j = \frac{\theta_j}{1-\theta_1}$ for $j \in \{2, \cdots, K\}$, we obtain

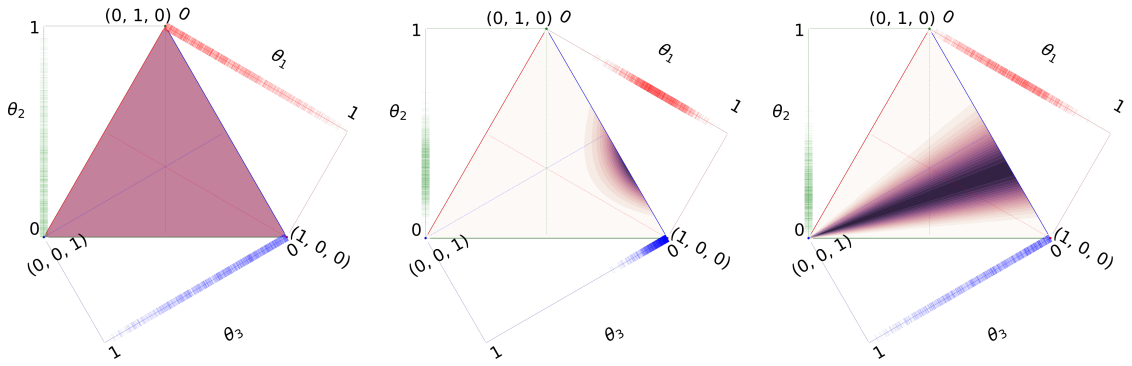
$$\begin{aligned}
p(\theta_1 \mid \quad & \alpha \quad , \beta_0, k_{1:T}, C_{1:T}) \\
& \propto \quad \theta_1^{\alpha_1 - 1}(1-\theta_1)^{\sum_{j=2}^K \alpha_j - 1} \int_\Delta \prod_{j=2}^K u_j^{\alpha_j - 1} \prod_{C \in \mathcal{C}} \frac{\prod_{j=2}^K \left((1-\theta_1)u_j\right)^{\nu(j,C)}}{\left((1-\theta_1)(\sum_{i \in C} u_i)\right)^{\mu(C)}} du \\
& = \quad \theta_1^{\alpha_1 - 1}(1-\theta_1)^{\sum_{j=2}^K \alpha_j - 1} \int_\Delta \prod_{j=2}^K u_j^{\alpha_j - 1} \prod_{C \in \mathcal{C}} \frac{\prod_{j=2}^K u_j^{\nu(j,C)}}{\left(\sum_{i \in C} u_i\right)^{\mu(C)}} du \\
& \propto \quad \theta_1^{\alpha_1 - 1}(1-\theta_1)^{\sum_{j=2}^K \alpha_j - 1}.
\end{aligned}$$

Then, the posterior $\theta_1 \mid \alpha, k_{1:T}, C_{1:T} \sim \mathcal{B}(\alpha_1, \sum_{j=2}^K \alpha_j)$ is also Beta distributed with parameters $(\alpha_1, \sum_{j=2}^K \alpha_j)$, identically to the prior $p(\theta_1|\alpha, \beta_0)$. □

This result extends trivially to *groups* of options.

4.3.4. Extending the Item Pool

Let us highlight an important implication of Lemma 4.3. Under the Dirichlet-Luce model, alternatives that are newly introduced to the system benefit from the same invariance that other options that were never presented enjoy. This property emerges as a natural way to ensure consistent inference of preferences for *cold-started* items.



(a) Prior (b) $p(\theta \mid k_{1:T}, \alpha)$ (c) $p(\theta \mid k_{1:T}, C_{1:T}, \alpha, \beta_0$

Figure 4.2. Prior choice probabilities (a), along with the posterior under Dirichlet-Multinomial (b) and Dirichlet-Luce (c) models, where options 1 and 2 were chosen from $\{1, 2\}$ 10 and 5 times, respectively. Contours of the joint density are shown on a simplex plot. Samples from posterior marginals are marked along an axis parallel to the line segment from the vertex where $\theta_k = 1$, perpendicular to the base.

4.4. Dirichlet-Luce Bandit Algorithm

Cast as the model assumption of an interactive system, the Dirichlet-Luce model provides efficient preference estimates from historical recommendation-choice observations. However, in an interactive scenario, the onus is on the recommender system to select a subset of options to present.

That is, the system needs an efficient mechanism that simultaneously explores the options which the user might like and exploits the current best alternatives, and composes the recommendations accordingly. Here, we frame this interactive recommendation scenario as a bandit problem. In a bandit setting, the Bayesian construction of the Dirichlet-Luce model serves a dual purpose. First, new choice observations can be used to inform efficient approximate inference of latent preferences. More importantly, posterior samples of the model serve as a natural means to manage the exploration-exploitation trade-off, inducing a presentation mechanism conditioned on previous choices ($k_{1:T}$) and the mechanism itself ($C_{1:T}$).

The posterior places high probability density on preferences where θ_k is high either when k was chosen frequently or presented rarely. This observation leads to the Dirichlet-Luce bandit algorithm in Figure 4.3. Presenting top-L items of the vector θ sampled from the posterior can be used as a recommendation: the system would recommend the alternatives that the user would like, or might like. Note that sampling from the posterior and picking the top options correspond to recommending an option with the posterior probability that its choice probability exceeds others that are not recommended. Then a choice feedback to the recommendation is received, and the posterior preferences are updated. The algorithm (Figure 4.3), as a result, dynamically infers the preferences and learns to recommend through a sequence of recommendation-choice observations.

Input: T

Initialize α, and set $\beta \leftarrow \beta_0$

$\mu(C) \leftarrow 0$ for all $C \in \mathcal{C}$

$y_k \leftarrow 0$ for all $k \in [K]$

for $t = 1$ **to** T **do**

  Sample $\theta \sim p_t(\theta \mid y, \mu, \alpha, \beta)$

  Form C_t with top L options from sampled θ

  Get preference feedback k_t to C_t

  $\mu(C_t) \leftarrow \mu(C_t) + 1$ {Update sufficient statistics}

  $y_{k_t} \leftarrow y_{k_t} + 1$

end for

Figure 4.3. The Dirichlet-Luce bandit algorithm.

The Dirichlet-Luce bandit algorithm is a Thompson sampling instance, an algorithm for decision making through posterior sampling [32], widely used for bandit problems as discussed before. To reiterate, in the standard multi-armed bandit setup where each arm reveals a binary reward, Thompson sampling corresponds to independently sampling from the marginal posterior distributions of the reward probabilities, and pulling the arm associated with the largest sample. In our case, conditioned on

all previous interactions, the posterior sampling is from the joint distribution of choice probabilities. It affords fast exploration of the space of subsets for reasons analogous to the fast convergence of posterior preferences under Dirichlet-Luce model.

4.4.1. Sampling Subroutine

The main ingredients of the Dirichlet-Luce bandit algorithm are sampling from the posterior distribution of choice probabilities, gathering an observation (a recommendation and the user choice) and updating the distribution of choice probabilities. Dirichlet-Luce model then becomes a natural choice due to its conjugacy properties; since it is convenient to update the distribution upon an observation, and it is possible to design an efficient sequential sampler that dynamically updates an initial set of weighted samples.

In particular, the model leads to a sequential Monte Carlo (SMC) algorithm as described in [35], which we use to implement the actual sampling subroutine in Figure 4.3. Specifically, the initial set of samples (particles), indexed by i, are drawn from $\theta^{(i)} \sim \mathcal{D}(\mathbf{1})$ and assigned unit weights. After each interaction, the weight of a particle ($w^{(i)}$) is updated recursively based on the recommendation-choice pair (C_t, k_t),

$$\begin{aligned} w_t^{(i)} &= w_{t-1}^{(i)} \frac{p_t(\theta^{(i)})}{p_{t-1}(\theta^{(i)})} \\ &= w_{t-1}^{(i)} \frac{\theta_{k_t}^{(i)}}{\sum_{\kappa \in C_t} \theta_\kappa^{(i)}}. \end{aligned}$$

Here, $p_t(\theta^{(i)})$ denotes the posterior density at time t evaluated at the point $\theta^{(i)}$, short for $p_t(\theta^{(i)} \mid C_{1:t}, k_{1:t}, \alpha, \beta)$.

Whenever the *effective sample size* drops below a set threshold, $N/2$, we perform multinomial resampling followed by a move step. $N/2$ is a typical choice for particle sample implementations [169], and it is a trade-off between accuracy and computation time. In our setup, we cannot directly sample from the full conditional densities

$f_j = p(\theta_j \mid \{\theta_l\}_{l \notin \{j,K\}})$. Hence the move step is performed with a *Metropolis-within-Gibbs* transition kernel which targets the posterior at t, $p_t(\theta)$, *i.e.*, particles undergo a *resample-and-move* update. Metropolis-within-Gibbs sampling can be performed by various alternatives for sampling from unnormalized densities, *e.g.*, *slice sampling* [170], *adaptive rejection Metropolis sampling* (ARMS) [171]. However, they require multiple evaluations of the full-conditional densities, which are prohibitive in our case since their evaluation at any given point has a considerably high computational cost. We use a simpler but effective Metropolis scheme which requires only two evaluations of f_j for each coordinate, as follows: The coordinates except j constrain the jth coordinate to an interval, $\mathcal{I} = \left(0, 1 - \sum_{j' \neq j} \theta_{j'}\right)$. We sample $\hat{\theta}_j \sim \mathcal{U}(\mathcal{I})$ (uniformly in the interval $\mathcal{I}$), and accept it with probability $\min\{1, f_j(\hat{\theta}_j)/f_j(\theta_j)\}$.

It is worth noting that the conjugacy in our model leads to particle weight updates that take a negligible $O(L)$ time. Unnormalized posterior evaluations, which are expensive, are only needed during the resample-and-move step. However, as the number of observations T grows, the posterior peaks around the latent preference vector and the algorithm requires resampling with decreasing frequency. That is, expensive resampling steps are dominant as the algorithm tends towards exploration, and very rare when the posterior is peaked and the algorithm "commits" a preference representation. The complete algorithm with SMC sampling procedure is part of [172], and is listed as an appendix. A reference implementation of the sampler along with an infrastructure to conduct simulations is made publicly available under MIT license in [172].

4.4.2. The Role of Exploration

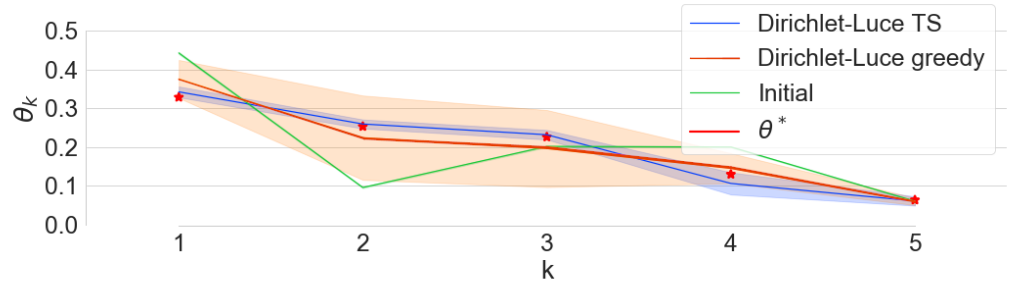
The Dirichlet-Luce bandit algorithm described here completes the interactive system setup. We now turn into the essential role that a bandit algorithm plays for recommendation—in addition to choice modeling that explicitly accounts for the user's limited exposure to the items. Contrary to a greedy mechanism that presents top-L options to the user based on the *posterior preference estimate* of the Dirichlet-Luce

model, the Dirichlet-Luce bandit algorithm composes a recommendation from a *posterior preference sample*, thereby achieves exploration of under-presented alternatives. The following two examples illustrate this role. In both examples, we assume there are five options, with true choice probabilities $\theta_i^* > \theta_j^*$ whenever $i < j$, without loss of generality.

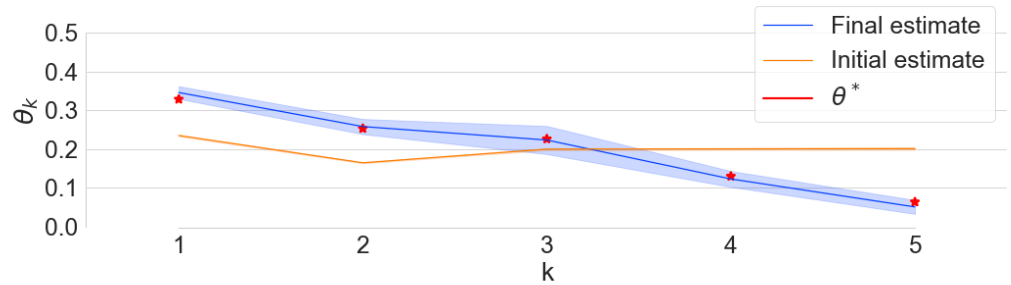
4.4.2.1. Second chance. A side effect of the stochastic transitivity is that once an already inferior option is preferred to a relatively favorable one, the preference estimate for this option will decrease, even it is not further presented, as others will be preferred to the former. However, exploration due to the algorithm ensures that it will be presented, *i.e.*, it will get a *second chance* (Figure 4.4(a)). We consider the case, where, initially, option 1 was preferred to option 5 (10 times), and option 5 to option 2. In the remaining rounds, as option 1 is preferred to other options and option 5 is usually ignored, stochastic transitivity enforces a decrease in the mean preference estimate of θ_2 as it does not appear in recommendations. However, exploration due to Thompson sampling ensures that options that are underrepresented in previous interactions will be recommended.

4.4.2.2. Robustness to unfair comparisons. Due to the independence of irrelevant alternatives, repetitive presentation of a relatively favorable option together with a more superior one might lead the system to develop a misconception on this option that it is overall inferior. By allowing sufficient comparison of options thanks to the recommendation algorithm, as demonstrated in Figure 4.4(b), the system would correct the preference estimates over the course of interactions. There, when an initial 100 choices are made from the alternatives $\{1, 2\}$, Dirichlet-Luce initially imposes negative bias towards option 2. Over the course of interactions, the system captures that option 2 is still preferable to other, originally inferior options.

That said, under natural conditions stated as "exploit" and "independence of irrelevant options" properties, [173] showed that collecting data in an adaptive manner



(a) Second chance



(b) Robustness to unfair comparisons

Figure 4.4. The role of exploration in recommendation. x-axis indexes the items. The y coordinate corresponds to preference probability θ_k—its true and estimated values by the algorithms after $T = 1000$ interactions. Exploration ensures that option 2 will get a second chance (a), also corrects the initially negative bias towards option 2 (b). Shaded regions denote the standard deviation in ten simulations.

(as in bandit algorithms) imposes negative bias on the mean estimate of the reward corresponding to an arm, due to an asymmetry in data collection. The mean estimate of the reward of a 'lucky' arm (with a current empirical mean larger than its true mean) would revert back to its true mean over time, but if the arm is 'unlucky' it might stuck with the negative bias, hence the overall bias is negative. We think the same phenomenon would be observed for our bandit algorithm, too. While we focus on the biases that arise from inaccurate modeling of user behavior and the properties of the model itself, whether a particular family of bandit setups would correct for this undesirable phenomenon remains another direction for further work.

4.5. Simulation Experiments

The proposed framework includes two main components for interactive recommender systems. The Dirichlet-Luce model explicitly takes into account the user's limited and systematic exposure to the alternatives in the inference of user preferences. The Dirichlet-Luce bandit algorithm aims to quickly discover and commit to the recommendation that includes the most preferable items without overlooking any alternatives.

The question of why we would need the second component, *i.e.*, the Dirichlet-Luce bandit algorithm in addition to the Dirichlet-Luce choice model, is addressed in Section 4.4. In particular, we have discussed two cases. First, what happens to an unexplored but favorable item, when a known inferior item is preferred to it? In this case, even the favorable item is not further presented, stochastic transitivity enforces a decrease in the mean preference estimate for it—as the inferior item will not be preferred much in the future recommendations. But exploration due to the bandit algorithm ensures that it will get a second chance. Another deficit of the choice model occurs when the system makes repetitive comparisons of a relatively favorable item with a superior one. In this case, the preference for a relatively favorable item would be underestimated, which is corrected by the algorithm as it will compose recommendations including that item and originally inferior items.

In this section, our focus will be on the potential biases of bandit algorithms assuming incorrect underlying models, and the performance of the bandit algorithms that have a similar setup to that of ours. In particular, we describe various simulation experiments that address *two questions*. First, how does the Dirichlet-Luce model, in interactive setups that would potentially lead to biased preference estimates, compare to its counterpart that naively ignores the user's exposure to the alternatives? Second, how does the Dirichlet-Luce bandit algorithm compare to the existing bandit algorithms in its performance of interactively learning to recommend the top options?

4.5.1. Demonstration of Biases in Preference Estimation

We first give synthetic examples that mimic challenges to healthy recommendation machinery by potentially leading to biased estimates of user preferences. We design two examples to investigate whether explicitly conditioning on systematic exposure ensures *robustness to the promotion* of an item and *discovery of initially censored* items. Our examples highlight the potential biases that might occur due to incorrect modeling of the user's choice behavior.

Recall that when the systematic exposure to the alternatives is ignored, *i.e.*, the user is assumed to pick an option k among $[K]$ regardless of the recommendation C, the Dirichlet-Luce model reduces to the Dirichlet-Multinomial model. A Thompson sampler based on this model assumes a Dirichlet prior on preference probabilities, where a choice observation leads to a Dirichlet posterior due to conjugacy. In addition, interpretation of user choice as a multinomial observation is common, as discussed in Section 4.1. Hence the Dirichlet-Multinomial model provides a test bed to assess the contribution of the model.

4.5.1.1. Simulation setup. In the two examples we will presently describe, we simulate an interactive system where the user's preferences, represented as choice probabilities $\theta^*_k = p(k \mid [K])\ \forall k \in [K]$, are latent. Over the course of interactions, the system recommends $\{C_t \mid t = 1, \ldots, T\}$ and observes choice feedback $k_{1:T}$ (where $k_t \in C_t$). We assume for each interaction, $L = 2$ of $K = 5$ alternatives, say, $\{i, j\}$, are recommended to the user, and the user picks one of them proportional to θ^*_i and θ^*_j. To ensure the exploration of all alternatives, we simulate Thompson sampling-based bandit algorithms as a recommender for both models, for reasons described at the outset of this section. In the following two sets of simulations, the latent choice probabilities (θ^*) are sampled from a Dirichlet distribution with parameter $\alpha_k = 1$ for all $k \in [K]$. K is small and a large number of interactions are simulated, and our goal is to show the biases that might occur due to incorrect modeling. For ease of exposition, we assume $\theta_1 > \theta_2 > \cdots > \theta_K$ without loss of generality.

4.5.1.2. Robustness to promotion. In this scenario, we simulate the case where some items are promoted. In particular, we assume that the third item, which would not be included in the optimal recommendation $\{1, 2\}$, is promoted, *i.e.*, included in every recommended subset.

Both the Dirichlet-Luce bandit algorithm and a Thompson sampler for the Dirichlet-Multinomial model are simulated as they are interacting with the user. Figure 4.5 depicts the resulting preference estimates by the corresponding models. We observe that Dirichlet-Luce, as illustrated in Figure 4.5, can cope with the overestimation of preferences for overrepresented items. Dirichlet-Multinomial on the other hand, ignores that the promoted item is overrepresented, and overestimates the preference to it simply because it is more frequently chosen.

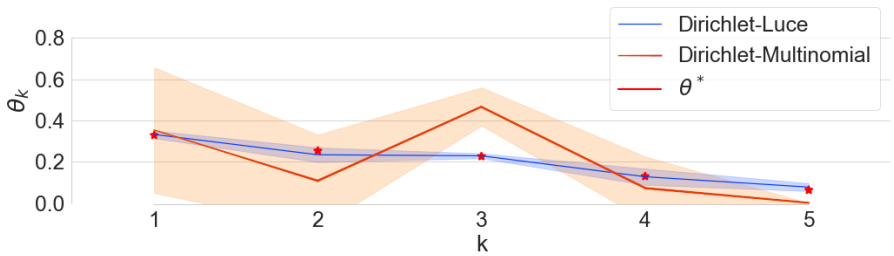


Figure 4.5. Final (after $T = 10000$ interactions) preference estimates averaged over 10 runs when option 3 is *promoted*, *i.e.*, it is included in every recommendation. Preference to overrepresented item 3 is overestimated by Dirichlet-Multinomial, whereas conditioning on presentations fixes this bias. Shaded regions denote the standard deviation.

4.5.1.3. Discovery of censored favorites. The same two algorithms are now compared in a different scenario, where some items are initially *censored*. In the case of originally favorable items are initially not included in recommendations, we would like the recommender system not to grow bias towards the user's initial choices, on the contrary, to give each option an equal opportunity to be presented. Initially censored items can be discovered by first not underestimating the choice probabilities to them, and then by

eventually learning to present the "best" items. Figure 4.6 demonstrates the described scenario and the resulting preference estimates.

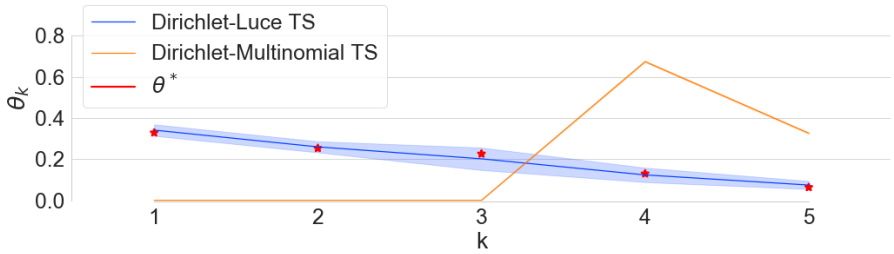


Figure 4.6. Followed by 100 choices made from originally inferior options $\{4, 5\}$, final preference estimates after $T = 10000$ interactions. Dirichlet-Luce does not impose negative bias towards censored options, and the implied recommender system eventually learns to recommend the best two items. They are underestimated, and cannot be discovered by the Dirichlet-Multinomial model.

These two examples illustrate that the Dirichlet-Luce model neither overestimates the preference to an overrepresented item nor overlooks initially underrepresented items.

4.5.2. Performance Comparisons

We now demonstrate the contribution of the Dirichlet-Luce bandit algorithm with another set of simulation experiments. Our goal is to show the efficacy of the algorithm in dynamically learning the top options through pairwise and L-wise preference feedback. Several dueling bandit algorithms, a combinatorial bandit algorithm, and an online learning to rank algorithm are considered. In a simulation, as before, we assume that a latent θ^* underlies the user choice given a presentation C. That is, the user chooses an option $k \in C$ with probability $\frac{\theta^*_k}{\sum_{\kappa in C} \theta^*_\kappa}$.

Here, an online learning to recommend algorithm's goal is to find out the most preferable items in all $[K]$, answering the query; which items the user would most likely choose if they had a chance to evaluate all possible alternatives. The following sets of

simulations compare a number of algorithms in their performance to achieve this goal.

4.5.2.1. Comparison with bandit algorithms with underlying click models. We now report a comparison of the Dirichlet-Luce bandit to the *TopRank algorithm* [151]. The TopRank algorithm maintains a statistic to estimate the pairwise order relations, and computes the ranking based on a topological sort of the items, informed by the confidence on the pairwise ordering estimate.

In addition to TopRank, we introduce another Thompson sampling variant as a baseline that mimics the bandit algorithms for click models when the position bias is ignored, the Beta-Bernoulli. When a recommendation C_t of L items is made at time t, the user is assumed to see all L items and make a choice. The independence assumption (see Section 4.2) of the click models renders a convenient bandit algorithm, as follows. The reward of an arm is assumed a Bernoulli random variable with an unknown mean parameter. The mean reward of an arm is assigned a uniform prior following beta distribution with parameters $\alpha = 1$ and $\beta = 1$, *i.e.*, $\mathcal{B}(1, 1)$, and a click (choice) increments α parameter of the clicked item by 1, whereas the β parameters of the items that are recommended by the system but not clicked by the user are incremented by 1. If an item is not recommended, the corresponding beta distribution stays invariant. The recommendations are made according to the samples from the posterior beta distributions of every arm at each round.

As a list of the estimated top items, we use the ranking TopRank outputs at each time of interaction, and in the case of the Dirichlet-Luce and Beta-Bernoulli bandit algorithms, the ranking implied by the sampled preference probabilities. We compare the performances of the three algorithms in terms of the cumulative regret at the top-N options included in the presentation, that is,

$$R_T = T \left(\max_{i_1, i_2, \cdots, i_N} \sum_{n=1}^{N} \theta^*_{i_n} \right) - \sum_{t=1}^{T} \sum_{n=1}^{N} \theta^*_{\kappa_n^{(t)}}.$$

At round t, the value of a recommended alternative $\kappa_n^{(t)}$ at position $n \leq N$ to the user is defined as the expected value of the multinomial choice variable if all alternatives

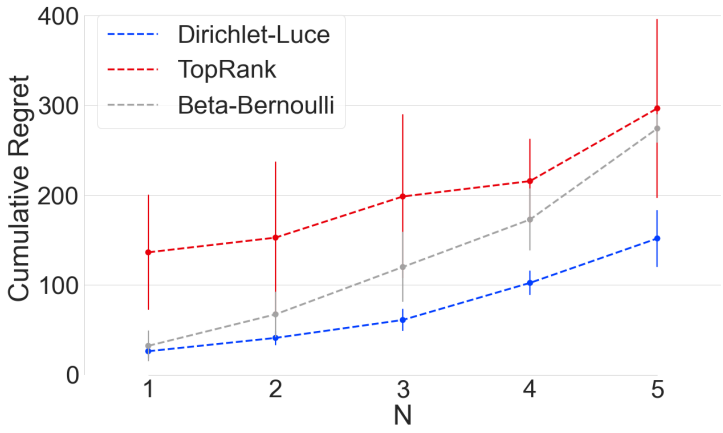
$[K]$ were to be considered, hence $\theta^*_{\kappa_n^{(t)}}$. That is, over the course of interactions, the gap between the collective preference probabilities of the optimal ranking at top-N positions and those proposed by the algorithm contributes to the cumulative regret. Presenting N most attractive items in the first N positions in any order incurs no regret at top-N.

In the following simulations, we consider two cases: (i) when θ^* is sampled from the Dirichlet distribution, (ii) when θ^* is estimated from Last.FM artist listening data set released as part of [174]. Dirichlet-Luce bandit, Beta-Bernoulli bandit, and TopRank algorithms are used to estimate the click model, and to recommend a list of $L = 5$ items. We evaluate the algorithms in terms of their cumulative regret, measured at top-N positions of the ranked lists recommended by both algorithms, for $N = 1, 2, \ldots, L$.

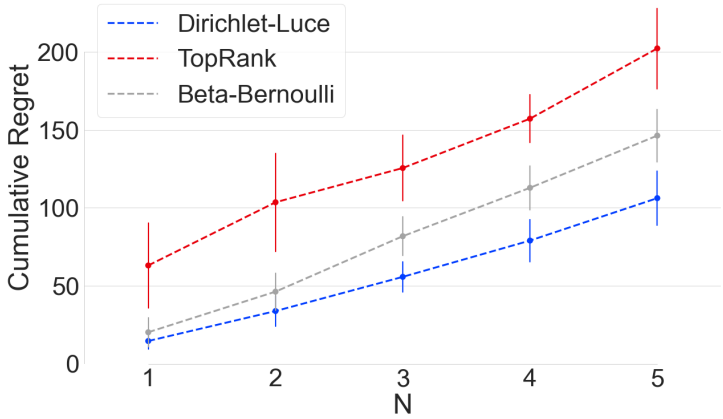
In the first case, we work with a sampled θ^*. We run two separate set of simulations for $K = 50$, using a sparse and a dense θ^*. θ^* is sampled from the Dirichlet distribution with concentration parameter $\alpha_\kappa = 0.1$ for all $\kappa \in [K]$ when it is sparse, and with $\alpha_\kappa = 5$ when it is dense. Across varying N, the Dirichlet-Luce bandit categorically achieves lower cumulative regret (Figure 4.7).

In the second case, θ^* is estimated from Last.FM artist listening data set. In a more realistic simulation, we first estimate the latent preference probabilities θ^* from the publicly available Last.FM data set [174]. The data set contains listening information of a set of 17632 music artists by a set of 1892 users. It additionally includes social networking and tagging data, which we do not use.

We first learn an LDA model from the listening logs. LDA, as described in Chapter 2 is a mixed membership model where collections of discrete random variables (in our case, collections of artists that users listen to) are assumed to be generated as follows. First, we assume a small set of population-level latent preference profiles, represented as discrete probability distributions over all artists. Then, a user's listening



(a) Sparse θ^*



(b) Dense θ^*

Figure 4.7. Average cumulative regret (lower is better) at top-N positions included in a recommendation/ranking in online learning to rank scenario after round $T = 10000$, where θ^* is sampled from the Dirichlet distribution with parameter $\alpha_k = 0.5$ (a), and $\alpha_k = 5$ (b). Error bars denote the standard deviation. TopRank hyperparameter δ was optimized in a held-out experiment.

behavior is assumed to follow a mixture of these preference profiles. That is, for every listening record in their history, the user first picks a particular preference profile, and then listens to an artist according to the probability distribution representing the preference profile. Latent preference profiles and the mixture proportions per user are assumed to be Dirichlet random variables. We fit LDA to the artist listening data with

20 latent preference profiles, assuming sparse priors on preference probability vectors and flat priors for per-user mixture components. We took the mean of the posterior Dirichlet corresponding to one, arbitrarily selected preference profile, which gives a probability vector over all artists, representing a typical preference probability vector. We then took the top 1000 artists (hence $K = 1000$ for this set of simulations), and normalized it such that the probabilities add up to one. The resulting vector is used as θ^*, from which user feedback is simulated. Top 50 elements of the selected θ^*, along with the artist names corresponding them, are included as an appendix.

Across varying N, the Dirichlet-Luce bandit categorically achieves lower cumulative regret than TopRank and Beta-Bernoulli bandit algorithms (Figure 4.8). That said, the Beta-Bernoulli bandit algorithm results in substantially lower variance of regret across simulations.

One might be concerned about the experiments being conducted using a single preference probability vector obtained through LDA. We understand that this setup might seem limited. We note, however, that the preference probability vector used for the experiment is at a population level. That is, it is not for a single user found in the data set, but it underlies parts of each user's entire listening behavior (in different proportions for different users). Indeed, there are 20 such preference probability vectors. We believe, however, combining the regret distributions over several simulations over 20 preference probability vectors would not give a clear idea to the reader on the variance of regret across experiments.

4.5.2.2. Comparison with dueling and combinatorial bandit algorithms. We now focus on the case where $L = 2$, of particular interest since this specific instantiation of the problem can be viewed as an instance of *dueling bandits* [136]. In this set of experiments, user feedback is simulated based on the latent θ^* vectors that are used in online click modeling experiments. The simulated user feedback observations to a pair of items, *i.e.*, a *duel* $\{i, j\}$, are sampled from $p(i \mid \{i, j\}) = \frac{\theta^*_i}{\theta^*_i + \theta^*_j}$.

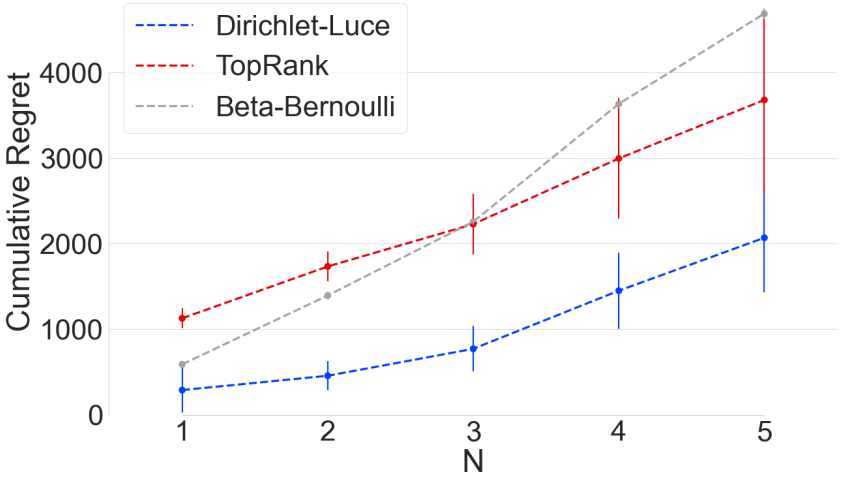


Figure 4.8. Average cumulative regret (lower is better) at top-N positions included in a ranking after round $T = 100000$, in an online learning to rank scenario, where θ^* is a 1000 dimensional preference probabilities vector estimated from Last.FM artist listening data set. Error bars denote the standard deviation across ten simulations assuming the same θ^*. TopRank hyperparameter δ is adaptively set.

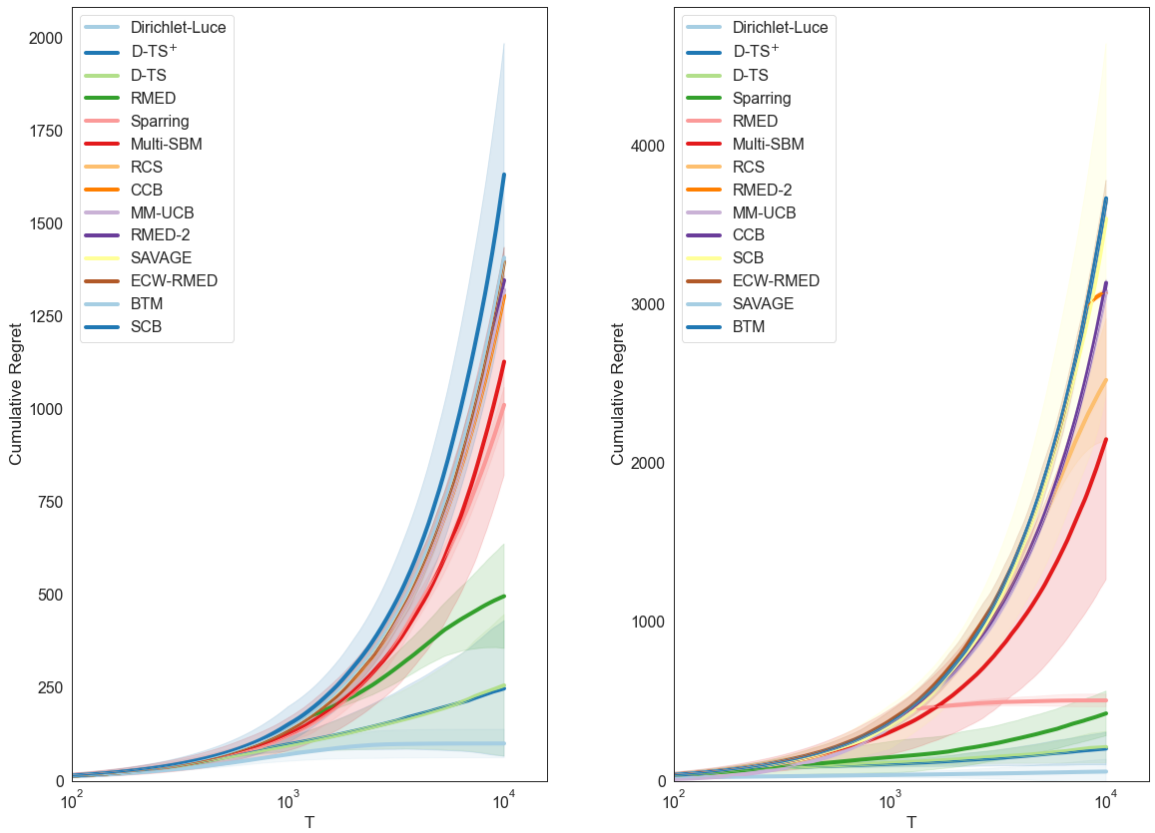
We make a set of comparisons of *Dirichlet-Luce bandit* with a set of dueling bandit algorithms as released in [153], namely, "double Thompson sampling" (*D-TS*) and its variant *D-TS+* [153], "relative minimum empirical divergence" (*RMED*) [175], "relative upper confidence bound" (*R-UCB*) [176], "beat the mean" (*BTM*) [177], "sensitivity analysis of variables for generic exploration" (*SAVAGE*) [178], "Sparring" [179], "relative confidence sampling" (*RCS*) [180], "Copeland confidence bound" (*CCB*) [181], and "efficient Copeland winners RMED" (*ECW-RMED*) [182], and a combinatorial bandit with relative feedback algorithm [135], "MaxMin UCB" (*MM-UCB*) for $L = 2$. Since a dueling bandits algorithm is allowed to present multisets (and ours never presents a multiset), we make the evaluation in terms of standard cumulative *weak* dueling regret (in T rounds),

$$R_T = \sum_{t=1}^{T} \min \left[\tilde{p} \left(i^* \mid \{i^*, \kappa_1^{(t)}\} \right), \tilde{p} \left(i^* \mid \{i^*, \kappa_2^{(t)}\} \right) \right],$$

where i^* is the user's favorite option, and $C_t = \{\kappa_1^{(t)}, \kappa_2^{(t)}\}$ is the presentation by the

learner at round t. $\tilde{p}(i^* \mid \{i^*, \kappa\}) = p(i^* \mid \{i^*, \kappa\}) - \frac{1}{2}$, where $p(i \mid \{i, j\})$ is the probability of preferring the option i to j. Including i^* in a presentation incurs no regret.

The results are reported in Figure 4.9. We find that the Dirichlet-Luce bandit for online recommendation results in substantially lower regret when stochastic transitivity is assumed to hold.



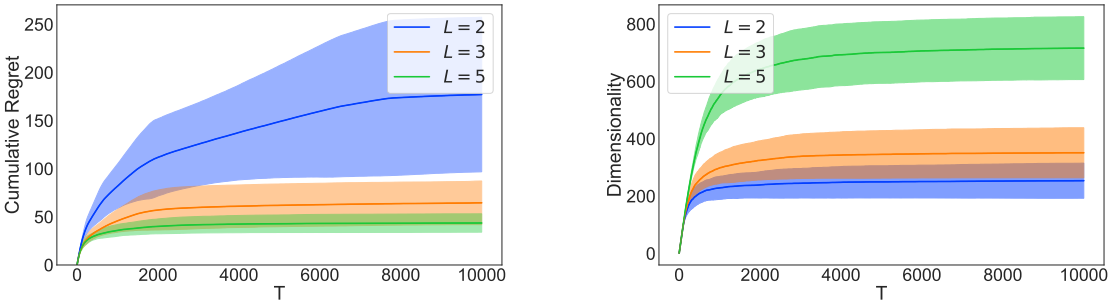
(a) Dense θ^*

(b) Sparse θ^*

Figure 4.9. Average cumulative weak dueling regret (lower is better) with respect to time (shown at log-scale) in a dueling bandits setup with simulated transitive preference feedback. The lists of algorithms in the legends are ordered by their average performance in the simulations. Shaded regions denote the standard deviation.

The closest competitor, D-TS (and its variant), puts a Beta prior to pairwise preference probabilities and proceeds by presenting subsets with two options from the posterior probability of an item being preferable to every other item. In contrast, we assume the prior in Equation 4.2 for the joint distribution of choice probabilities. Note, however, that the modeling assumptions of most dueling bandit algorithms can capture non-transitive choice behavior, although TopRank [151] and MM-UCB [135] cannot. The combinatorial bandit algorithm included in the experiment, MM-UCB, makes the same assumption as ours on the user behavior. Finally, we highlight that the Dirichlet-Luce bandit algorithm does not need to maintain pairwise preference statistics.

4.5.2.3. The effect of the number of items in recommendation. Our bandit algorithm outperforms the baseline dueling bandits, combinatorial bandits and online learning to rank algorithms in both pairwise ($L = 2$) and subset-wise ($L = 5$) selection tasks. In Figure 4.10(a), fixing $N = 2$, we explore how learning speed improves as the system is allowed to make larger presentations. As expected, growing presentation sizes leads to lower cumulative regret, *i.e.*, the algorithm learns to present the top-2 options sooner. We also report the dimensionality of the statistic μ—the number of unique presentations explored before converging to a preference estimate. Despite the potentially high complexity, the algorithm maintains manageably low-dimensional statistics (Figure 4.10(b)). For these simulations, we used a 100-dimensional sparse θ^* simulated from the Dirichlet distribution.



(a) Performance at top-2

(b) Number of unique presentations

Figure 4.10. Average cumulative regret (over 10 runs) at top-2 options (out of 100) included in presentations with different sizes (a), and the effective dimensionality of the statistic μ (b) over the course of interactions. Shaded regions denote the standard deviation.

5. CONCLUSION

Machine learning from online human behavior is central to many modern digital platforms. It is used for numerous applications, primarily to present personalized, relevant content—due to the abundance of available content. The goal is to ensure user satisfaction and long-term engagement with the platform. Indeed, models and algorithms that learn from historical user behavior to understand their preferences have gained much traction. Several models have shown success in relating the users with each other and how personalized inferences can be made from a collection of users. For a successful platform, however, a large number of users continuously generate data for algorithms to learn from, such as by clicking on items, choosing from alternatives, reading articles, or watching streaming content, ending up with massive data sets. Therefore, the algorithms for learning applicable models from such data sets should scale well to be helpful.

On the other hand, online human behavior should be interpreted with great care, as human behavior is subject to cognitive biases. To name a few, the user might trust the algorithmic recommendations and act accordingly, popular opinion might influence user behavior, or the user might have a tendency to choose from what is recommended to them, showing the least effort into exploring further.

In addition, the platforms collect data sets comprising user behavior in their interactions with the algorithm, causing them to learn from the data sets that they induce. Specifically, learning from user interactions with a recommender system is challenging due to the inherent feedback loop in the interaction dynamics. The learning algorithm's task is to learn from its previous interactions with the users to design future interactions. However, the algorithm influences the interaction data input to itself. A simple example is that the user cannot click on an item if the algorithm does not show it on the platform. As a result, the platform's failure to ensure adequate representation of different content harms content providers—other vital stakeholders

of modern digital platforms. They contribute to the platform, and it is on the platform to ensure they attract adequate attention and are not buried at the end of endless lists of alternatives. Consequently, a recommender system should explicitly account for the interaction dynamics, not merely user actions alone.

This dissertation has reviewed the progress and reported our contributions in modeling observations involving user-item pairs, implementing learning algorithms in a scalable manner, and addressing biases that arise in data sets of online human behavior in their interaction with algorithms that learn from human behavior.

The first part of the thesis is concerned with extending a widely-used class of models, matrix factorization, for collaborative filtering. We proposed an extension to Poisson matrix factorization to model a multinomial family of observations, which rendered a generic factorization algorithm with more interpretable parameters and higher performance than its counterparts across relevant tasks. We then discussed common approaches for learning matrix factorization variants via different algorithms but in a scalable manner. We concluded the part with a brief review of software libraries—one of which we have contributed to—that implement those scalable algorithms in modern distributed processing platforms.

An obvious next step was to design more complex models that give higher accuracy on benchmark data sets or to design algorithms that scale even better. However, we focused on a more fundamental problem of user modeling: biases in human behavior and their effect on the data sets that include human behavior. In particular, we studied choice modeling for interactive recommender systems, where the users choose one among a limited number of systematically presented options.

The choice model differs from the models designed to learn from explicit user feedback to recommendations, or common click models, in an important way. Contrary to such models, where the user choice is assumed independent of other options (items) given that the user examines the item, in choice modeling, user choice is relative. The

user is assumed to have a tendency to choose from what is recommended by the system; thereby, their choice depends on other items in the recommended set.

These observations led us to introduce the Dirichlet-Luce model, a Bayesian choice model that is aware of systematic and limited exposure: only a systematically-selected small fraction of the options are presented to the users. We proposed a conjugate family of distributions, a generalization of the Dirichlet distribution, to obtain a density where posterior samples place high mass on frequently preferred options and those rarely presented. We studied the Dirichlet-Luce model in detail and showed a critical practical aspect: $O(K)$ distinct pairwise presentations are sufficient to recover user preferences.

Bayesian treatment of the model paved the way for a bandit algorithm for online learning to recommend—dynamically estimating user preferences and making recommendations based on Thompson sampling. This algorithm, the Dirichlet-Luce bandit, addresses sampling biases by balancing exploring novel items and exploiting the alternatives found to be preferable.

We found that both the Dirichlet-Luce model and the Dirichlet-Luce bandit are crucial in eliminating some biases that recommender systems are prone to. We also showed that the proposed framework outperforms several related bandit algorithms previously proposed to be used in recommender and online learning to rank systems, as well as a generic combinatorial bandit algorithm.

Overall, we have contributed to multiple aspects of modeling and learning from online human behavior. We believe the report would be helpful for practitioners when designing applications that learn from online human behavior.

That said, our contributions are not complementary. The first set of contributions ignores human biases, and the second one is not readily available for use in a real-world digital platform. Learning complex models and addressing user biases might

seem orthogonal, but an exciting next step is to design complex models with great predictive power, while acknowledging biases in human behavior and in the behavior data sets. A starting point would be to adapt Dirichlet-Luce model (and bandit) to a contextual setting, where there are many users and items but few (perhaps latent) contexts, each associated with a composition of choice probabilities. This extension serves two purposes. First, it addresses the shortcoming of the Dirichlet-Luce choice model that it does not model cyclical preferences. It also extends the model so that multiple users' complex behavior [167, 168] can be captured. This setup is similar to bandit algorithms assuming a factorization model. [143] discuss such a model where posterior samples for the user and item factors are used for recommendation, leading to a Thompson sampler. [144], slightly differently, use an estimate of the item factors. As a result, choice probabilities can be inferred based on collaborative feedback as in collaborative filtering-based recommender systems.

In addition, to address sampling biases, one has two main alternatives: "to model or to intervene," as [183] puts it. Indeed, there are other solution approaches (such as off-policy or counterfactual learning instead of online learning) that we reviewed but did not apply. Another future direction would be to provide a counterfactual learning to recommend framework, possibly combined with some online learning when required, as described in [184]. That said, addressing several user biases with a unified, causal user model is another exciting direction to be explored. The Dirichlet-Luce model already addresses the exposure bias and the user's choice behavior due to their trust in the system, the *trust bias* as discussed by [15], to an extent. A further extension that combines the relative choice assumption with the position bias in user choice would lead to a more viable approach for recommender system designers.

REFERENCES

1. Koren, Y., R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems", *Computer*, Vol. 42, No. 8, pp. 30–37, 2009.

2. Liang, D., R. G. Krishnan, M. D. Hoffman and T. Jebara, "Variational Autoencoders for Collaborative Filtering", *World Wide Web Conference*, pp. 689–698, Lyon, France, 2018.

3. Mnih, A. and R. R. Salakhutdinov, "Probabilistic Matrix Factorization", *Conference on Neural Information Processing Systems*, pp. 1257–1264, Vancouver, Canada, 2007.

4. Cemgil, A. T., "Bayesian Inference for Nonnegative Matrix Factorisation Models", *Computational Intelligence and Neuroscience*, 2008.

5. Yang, S.-H., B. Long, A. J. Smola, H. Zha and Z. Zheng, "Collaborative Competitive Filtering: Learning Recommender using Context of User Choice", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 295–304, New York City, New York, United States, 2011.

6. Blei, D. M., A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation", *The Journal of Machine Learning Research*, Vol. 3, pp. 993–1022, 2003.

7. Covington, P., J. Adams and E. Sargin, "Deep Neural Networks for YouTube Recommendations", *ACM Conference on Recommender Systems*, pp. 191–198, Boston, Massachusetts, United States, 2016.

8. Anil, R., G. Çapan, I. Drost-Fromm, T. Dunning, E. Friedman, T. Grant, S. Quinn, P. Ranjan, S. Schelter and Ö. Yilmazel, "Apache Mahout: Machine Learning on Distributed Dataflow Systems.", *The Journal of Machine Learning Research*, Vol. 21, No. 127, pp. 1–6, 2020.

9. Schnabel, T., A. Swaminathan, A. Singh, N. Chandak and T. Joachims, "Recommendations as Treatments: Debiasing Learning and Evaluation", *International Conference on Machine Learning*, pp. 1670–1679, New York City, New York, United States, 2016.

10. Swaminathan, A. and T. Joachims, "Counterfactual Risk Minimization: Learning from Logged Bandit Feedback", *International Conference on Machine Learning*, pp. 814–823, Lille, France, 2015.

11. Hu, Y., Y. Koren and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets", *IEEE International Conference on Data Mining*, pp. 263–272, Pisa, Italy, 2008.

12. Liang, D., L. Charlin, J. McInerney and D. M. Blei, "Modeling User Exposure in Recommendation", *World Wide Web Conference*, p. 951–961, Montréal, Canada, 2016.

13. Chuklin, A., I. Markov and M. d. Rijke, "Click Models for Web Search", *Synthesis Lectures on Information Concepts, Retrieval, and Services*, Vol. 7, No. 3, pp. 1–115, 2015.

14. Joachims, T., L. Granka, B. Pan, H. Hembrooke and G. Gay, "Accurately Interpreting Clickthrough Data as Implicit Feedback", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 154–161, Salvador, Brazil, 2005.

15. Agarwal, A., X. Wang, C. Li, M. Bendersky and M. Najork, "Addressing Trust Bias for Unbiased Learning-to-Rank", *World Wide Web Conference*, pp. 4–14, San Francisco, California, United States, 2019.

16. Zipf, G. K., *Human Behavior and the Principle of Least Effort*, Addison Wesley Press, Inc., 1949.

17. Pariser, E., *The Filter Bubble: What the Internet is hiding from you*, Penguin UK, 2011.

18. Chaney, A. J., B. M. Stewart and B. E. Engelhardt, "How Algorithmic Confounding in Recommendation Systems Increases Homogeneity and Decreases Utility", *ACM Conference on Recommender Systems*, pp. 224–232, Vancouver, Canada, 2018.

19. Jiang, R., S. Chiappa, T. Lattimore, A. Agyorgy and P. Kohli, "Degenerate Feedback Loops in Recommender Systems", *AAAI/ACM Conference on AI, Ethics, and Society*, Honolulu, Hawaii, United States, 2019.

20. Singh, A. and T. Joachims, "Fairness of Exposure in Rankings", *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2219–2228, London, United Kingdom, 2018.

21. Mehrotra, R., J. McInerney, H. Bouchard, M. Lalmas and F. Diaz, "Towards a Fair Marketplace: Counterfactual Evaluation of the Trade-Off Between Relevance, Fairness & Satisfaction in Recommendation Systems", *ACM International Conference on Information and Knowledge Management*, pp. 2243–2251, Turin, Italy, 2018.

22. Patro, G. K., L. Porcaro, L. Mitchell, Q. Zhang, M. Zehlike and N. Garg, "Fair Ranking: A Critical Review, Challenges, and Future Directions", *arXiv preprint arXiv:2201.12662*, 2022.

23. Abdollahpouri, H., "Popularity Bias in Ranking and Recommendation", *AAAI/ACM Conference on AI, Ethics, and Society*, pp. 27–28, Honolulu, Hawaii, United States, 2019.

24. Lattimore, T. and C. Szepesvári, *Bandit Algorithms*, Cambridge University Press, 2020.

25. Joachims, T., Y. Raimond, O. Koch, M. Dimakopoulou, F. Vasile and A. Swaminathan, "REVEAL 2020: Bandit and Reinforcement Learning From User Interactions", *ACM Conference on Recommender Systems*, p. 628–629, Virtual Event, 2020.

26. Basilico, J., "Recent Trends in Personalization: A Netflix Perspective", *Workshop on Adaptive & Multitask Learning in Conjunction with the International Conference on Machine Learning*, Long Beach, California, United States, 2019.

27. Baeza-Yates, R., "Bias in Search and Recommender Systems", *ACM Conference on Recommender Systems*, p. 2, Virtual Event, 2020.

28. Çapan, G., S. Akbayrak, T. Y. Ceritli and A. T. Cemgil, "Sum Conditioned Poisson Factorization", *International Conference on Latent Variable Analysis and Signal Separation*, pp. 24–35, Surrey, United Kingdom, 2018.

29. Luce, R. D., *Individual Choice Behavior*, Wiley, 1959.

30. Çapan, G., İ. Gündoğdu, A. C. Türkmen, Ç. Sofuoğlu and A. T. Cemgil, "A Bayesian Choice Model for Eliminating Feedback Loops", *arXiv Preprint arXiv:1908.05640*, 2019.

31. Vinagre, J., A. M. Jorge, M. Al-Ghossein and A. Bifet, "Proceedings of the 4th Workshop on Workshop on Online Recommender Systems and User Modeling", *arXiv Preprint arXiv:2201.05156*, 2022.

32. Thompson, W. R., "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples", *Biometrika*, Vol. 25, No. 3, pp. 285–294, 1933.

33. Çapan, G., O. Bozal, I. Gündoğdu and A. T. Cemgil, "Towards Fair Personalization by Avoiding Feedback Loops", *Workshop on Human-Centric Machine Learning in Conjunction with the Conference on Neural Information Processing*

Systems, Vancouver, Canada, 2019.

34. Çapan, G., İ. Gündoğdu, A. C. Türkmen and A. T. Cemgil, "Dirichlet–Luce Choice Model for Learning From Interactions", *User Modeling and User-Adapted Interaction*, pp. 1–38, 2022.

35. Chopin, N., "A Sequential Particle Filter Method for Static Models", *Biometrika*, Vol. 89, No. 3, pp. 539–552, 2002.

36. Liu, T.-Y., "Learning to Rank for Information Retrieval", *Foundations and Trends® in Information Retrieval*, Vol. 3, No. 3, pp. 225–331, 2009.

37. Page, L., S. Brin, R. Motwani and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web.*, Tech. rep., Stanford InfoLab, 1999.

38. Freund, Y., R. Iyer, R. E. Schapire and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences", *The Journal of Machine Learning Research*, Vol. 4, pp. 933–969, 2003.

39. Xia, F., T.-Y. Liu, J. Wang, W. Zhang and H. Li, "Listwise Approach to Learning to Rank: Theory and Algorithm", *International Conference on Machine Learning*, pp. 1192–1199, Helsinki, Finland, 2008.

40. Cao, Z., T. Qin, T.-Y. Liu, M.-F. Tsai and H. Li, "Learning to Rank: From Pairwise Approach to Listwise Approach", *International Conference on Machine Learning*, pp. 129–136, Corvallis, Oregon, United States, 2007.

41. Manning, C. D., P. Raghavan and H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

42. Burges, C., T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender, "Learning to Rank Using Gradient Descent", *International Conference on Machine Learning*, pp. 89–96, Bonn, Germany, 2005.

43. Baydin, A. G., B. A. Pearlmutter, A. A. Radul and J. M. Siskind, "Automatic Differentiation in Machine Learning: A Survey", *The Journal of Marchine Learning Research*, Vol. 18, pp. 1–43, 2018.

44. Pasumarthi, R. K., S. Bruch, X. Wang, C. Li, M. Bendersky, M. Najork, J. Pfeifer, N. Golbandi, R. Anil and S. Wolf, "TF-Ranking", *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2970–2978, Anchorage, Alaska, United States, 2019.

45. Oosterhuis, H. and M. de Rijke, "Unifying Online and Counterfactual Learning to Rank: A Novel Counterfactual Estimator That Effectively Utilizes Online Interventions", *ACM International Conference on Web Search and Data Mining*, pp. 463–471, Virtual Event, 2021.

46. Goldberg, D., D. Nichols, B. M. Oki and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM*, Vol. 35, No. 12, pp. 61–70, 1992.

47. Resnick, P., N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl, "Grouplens: An Open Architecture for Collaborative Filtering of Netnews", *ACM Conference on Computer Supported Cooperative Work*, pp. 175–186, New York City, New York, United States, 1994.

48. Linden, G., B. Smith and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering", *IEEE Internet Computing*, Vol. 7, No. 1, pp. 76–80, 2003.

49. Herlocker, J. L., J. A. Konstan, A. Borchers and J. Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 230–237, Berkeley, California, United States, 1999.

50. Kingma, D. P. and M. Welling, "Auto-Encoding Variational Bayes", *arXiv Preprint arXiv:1312.6114*, 2013.

51. Funk, S., *Netflix Update: Try This at Home*, 2006, `https://sifter.org/simon/journal/20061211.html`, accessed on 5-March-2022.

52. Bennett, J. and S. Lanning, "The Netflix Prize", *In KDD Cup and Workshop in Conjunction with the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 3–6, San Jose, California, United States, 2007.

53. Lee, D. D. and H. S. Seung, "Learning the Parts of Objects by Non-Negative Matrix Factorization", *Nature*, Vol. 401, No. 6755, pp. 788–791, 1999.

54. Kingman, J. F. C., *Poisson Processes*, Vol. 3, Clarendon Press, 1992.

55. Gopalan, P., J. M. Hofman and D. M. Blei, "Scalable Recommendation With Poisson Factorization", *arXiv Preprint arXiv:1311.1704*, 2013.

56. Gopalan, P., J. M. Hofman and D. M. Blei, "Scalable Recommendation With Hierarchical Poisson Factorization", *Conference on Uncertainty in Artificial Intelligence*, pp. 326–335, Amsterdam, Netherlands, 2015.

57. Pan, R., Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz and Q. Yang, "One-class Collaborative Filtering", *IEEE International Conference on Data Mining*, pp. 502–511, Pisa, Italy, 2008.

58. Yıldırım, S., M. B. Kurutmaz, M. Barsbey, U. Şimşekli and A. T. Cemgil, "Bayesian Allocation Model: Marginal Likelihood-Based Model Selection for Count Tensors", *IEEE Journal of Selected Topics in Signal Processing*, Vol. 15, No. 3, pp. 560–573, 2020.

59. Paquet, U., B. Thomson and O. Winther, "A Hierarchical Model for Ordinal

Matrix Factorization", *Statistics and Computing*, Vol. 22, No. 4, pp. 945–957, 2012.

60. Rezende, D. J., S. Mohamed and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models", *International Conference on Machine Learning*, pp. 1278–1286, Beijing, China, 2014.

61. Jaakkola, T. S. and M. I. Jordan, "Variational Probabilistic Inference and the QMR-DT Network", *Journal of Artificial Intelligence Research*, Vol. 10, pp. 291–322, 1999.

62. Hoffman, M. D., D. M. Blei, C. Wang and J. Paisley, "Stochastic Variational Inference", *The Journal of Machine Learning Research*, 2013.

63. Dacrema, M. F., P. Cremonesi and D. Jannach, "Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches", *ACM Conference on Recommender Systems*, pp. 101–109, Copenhagen, Denmark, 2019.

64. Boyd, S., S. P. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

65. Robbins, H. and S. Monro, "A Stochastic Approximation Method", *The Annals of Mathematical Statistics*, Vol. 22, No. 3, pp. 400–407, 1951.

66. Bottou, L., "Large-Scale Machine Learning with Stochastic Gradient Descent", *International Conference on Computational Statistics*, pp. 177–186, Paris, France, 2010.

67. Wainwright, M. J., M. I. Jordan *et al.*, "Graphical Models, Exponential Families, and Variational Inference", *Foundations and Trends® in Machine Learning*, Vol. 1, No. 1–2, pp. 1–305, 2008.

68. Blei, D. M., A. Kucukelbir and J. D. McAuliffe, "Variational Inference: A Review for Statisticians", *Journal of the American Statistical Association*, Vol. 112, No. 518, pp. 859–877, 2017.

69. Paisley, J., D. M. Blei and M. I. Jordan, "Bayesian Nonnegative Matrix Factorization With Stochastic Variational Inference", *Handbook of Mixed Membership Models and Their Applications*, pp. 239–258, Chapman and Hall/CRC, 2014.

70. Ranganath, R., S. Gerrish and D. Blei, "Black Box Variational Inference", *Artificial Intelligence and Statistics*, pp. 814–822, Reykjavic, Iceland, 2014.

71. Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Tech. rep., 2015, `https://www.tensorflow.org/`.

72. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.

73. Gershman, S. and N. Goodman, "Amortized Inference in Probabilistic Reasoning", *Annual Meeting of the Cognitive Science Society*, Vol. 36, Quebec, Canada, 2014.

74. Bubeck, S., N. Cesa-Bianchi *et al.*, "Regret Analysis of Stochastic and Nonstochastic Multi-Armed Bandit Problems", *Foundations and Trends® in Machine Learning*, Vol. 5, No. 1, pp. 1–122, 2012.

75. Lai, T. L. and H. Robbins, "Asymptotically Efficient Adaptive Allocation Rules", *Advances in Applied Mathematics*, Vol. 6, No. 1, pp. 4–22, 1985.

76. Katehakis, M. N. and H. Robbins, "Sequential Choice From Several Populations.", *Proceedings of the National Academy of Sciences*, Vol. 92, No. 19, pp. 8584–8585, 1995.

77. Hoeffding, W., "Probability Inequalities for Sums of Bounded Random Variables", *Journal of the American Statistical Association*, Vol. 58, No. 301, pp. 13–30, 1963.

78. Russo, D. J., B. Van Roy, A. Kazerouni, I. Osband and Z. Wen, "A Tutorial on Thompson Sampling", *Foundations and Trends® in Machine Learning*, Vol. 11, No. 1, pp. 1–96, 2018.

79. Agrawal, S. and N. Goyal, "Analysis of Thompson Sampling for the Multi-Armed Bandit Problem", *Conference on Learning Theory*, pp. 39–1, Edinburgh, Scotland, 2012.

80. Agrawal, S. and N. Goyal, "Further Optimal Regret Bounds for Thompson Sampling", *Artificial Intelligence and Statistics*, pp. 99–107, Scottsdale, Arizona, United States, 2013.

81. Chapelle, O. and L. Li, "An Empirical Evaluation of Thompson Sampling", *Conference on Neural Information Processing Systems*, pp. 2249–2257, Granada, Spain, 2011.

82. Yu, T., B. Kveton, Z. Wen, R. Zhang and O. J. Mengshoel, "Graphical Models Meet Bandits: A Variational Thompson Sampling Approach", *International Conference on Machine Learning*, pp. 10902–10912, Virtual Event, 2020.

83. Phan, M., Y. Abbasi Yadkori and J. Domke, "Thompson Sampling and Approximate Inference", *Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019.

84. Dean, J. and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Communications of the ACM*, Vol. 51, No. 1, pp. 107–113, 2008.

85. *Apache Hadoop*, `https://hadoop.apache.org`, accessed on 5-March-2022.

86. Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster Computing With Working Sets", *USENIX Workshop on Hot Topics in Cloud Computing*, Boston, Massachusetts, 2010.

87. Alexandrov, A., R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl *et al.*, "The Stratosphere Platform for Big Data Analytics", *The International Journal on Very Large Databases*, Vol. 23, No. 6, pp. 939–964, 2014.

88. Carbone, P., A. Katsifodimos, S. Ewen, V. Markl, S. Haridi and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine", *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Vol. 36, No. 4, 2015.

89. Sanders, P., K. Mehlhorn, M. Dietzfelbinger and R. Dementiev, *Sequential and Parallel Algorithms and Data Structures*, Springer, 2019.

90. Schelter, S., C. Boden, M. Schenck, A. Alexandrov and V. Markl, "Distributed Matrix Factorization with Mapreduce Using a Series of Broadcast-Joins", *ACM Conference on Recommender Systems*, pp. 281–284, Hong Kong, China, 2013.

91. Chu, C.-T., S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun and A. Ng, "Map-Reduce for Machine Learning on Multicore", *Conference on Neural Information Processing Systems*, pp. 281–288, Vancouver, Canada, 2006.

92. Zinkevich, M., M. Weimer, L. Li and A. Smola, "Parallelized Stochastic Gradient Descent", *Conference on Neural Information Processing Systems*, pp. 2595–2603, Vancouver, Canada, 2010.

93. Recht, B., C. Re, S. Wright and F. Niu, "Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent", *Conference on Neural Information*

Processing Systems, pp. 693–701, Granada, Spain, 2011.

94. Agarwal, A. and J. C. Duchi, "Distributed Delayed Stochastic Optimization", *Conference on Neural Information Processing Systems*, pp. 873–881, Granada, Spain, 2011.

95. Li, M., D. G. Andersen, A. J. Smola and K. Yu, "Communication Efficient Distributed Machine Learning With the Parameter Server", *Conference on Neural Information Processing Systems*, pp. 19–27, Montréal, Canada, 2014.

96. Gemulla, R., E. Nijkamp, P. J. Haas and Y. Sismanis, "Large-Scale Matrix Factorization With Distributed Stochastic Gradient Descent", *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 69–77, San Diego, California, United States, 2011.

97. Teflioudi, C., F. Makari and R. Gemulla, "Distributed Matrix Completion", *IEEE International Conference on Data Mining*, pp. 655–664, Brussels, Belgium, 2012.

98. Schelter, S., C. Boden and V. Markl, "Scalable Similarity-Based Neighborhood Methods With Mapreduce", *ACM Conference on Recommender Systems*, pp. 163–170, Dublin, Ireland, 2012.

99. Lyubimov, D. and A. Palumbo, *Apache Mahout: Beyond MapReduce*, CreateSpace Independent Publishing Platform, 2016.

100. Meng, X., J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde and S. Owen, "MLlib: Machine Learning in Apache Spark", *The Journal of Machine Learning Research*, Vol. 17, No. 1, pp. 1235–1241, 2016.

101. Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models Are Few-Shot Learners", *Conference on Neural Information Processing Systems*, pp. 1877–1901, Virtual Event, 2020.

102. Liu, T.-Y., W. Chen and T. Qin, "Mechanism Learning with Mechanism Induced Data", *AAAI Conference on Artificial Intelligence*, pp. 4037–4041, Austin, Texas, United States, 2015.

103. Baeza-Yates, R., "Bias on the Web", *Communications of the ACM*, Vol. 61, No. 6, pp. 54–61, 2018.

104. Liang, D., L. Charlin and D. M. Blei, "Causal Inference for Recommendation", *Causation: Foundation to Application Workshop in Conjunction with the Conference on Uncertainty in Artifical Intelligence*, New York City, New York, United States, 2016.

105. Sinha, A., D. F. Gleich and K. Ramani, "Deconvolving Feedback Loops in Recommender Systems", *Conference on Neural Information Processing Systems*, pp. 3243–3251, Barcelona, Spain, 2016.

106. Sun, W., S. Khenissi, O. Nasraoui and P. Shafto, "Debiasing the Human-Recommender System Feedback Loop in Collaborative Filtering", *World Wide Web Conference*, pp. 645–651, San Francisco, California, United States, 2019.

107. Schmit, S. and C. Riquelme, "Human Interaction with Recommendation Systems", *International Conference on Artificial Intelligence and Statistics*, Lanzarote, Canary Islands, 2018.

108. Chen, J., H. Dong, X. Wang, F. Feng, M. Wang and X. He, "Bias and Debias in Recommender System: A Survey and Future Directions", *arXiv Preprint arXiv:2010.03240*, 2020.

109. Steck, H., "Training and Testing of Recommender Systems on Data Missing Not at Random", *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 713–722, Washington, DC, United States, 2010.

110. Marlin, B. M., R. S. Zemel, S. Roweis and M. Slaney, "Collaborative Filtering

and the Missing at Random Assumption", *Conference on Uncertainty in Artificial Intelligence*, pp. 267–275, Vancouver, Canada, 2007.

111. Hernández-Lobato, J. M., N. Houlsby and Z. Ghahramani, "Probabilistic Matrix Factorization with Non-Random Missing Data", *International Conference on Machine Learning*, p. 1512–1520, Beijing, China, 2014.

112. Hu, Y., Y. Koren and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets", *IEEE International Conference on Data Mining*, pp. 263–272, Pisa, Italy, 2008.

113. Craswell, N., O. Zoeter, M. Taylor and B. Ramsey, "An Experimental Comparison of Click Position-Bias Models", *International Conference on Web Search and Data Mining*, pp. 87–94, Palo Alto, California, United States, 2008.

114. Wang, T. and D. Wang, "Why Amazon's Ratings Might Mislead You: The Story of Herding Effects", *Big Data*, Vol. 2, No. 4, pp. 196–204, 2014.

115. Liu, Y., X. Cao and Y. Yu, "Are You Influenced by Others When Rating? Improve Rating Prediction by Conformity Modeling", *ACM Conference on Recommender Systems*, pp. 269–272, Boston, Massachusetts, United States, 2016.

116. Krishnan, S., J. Patel, M. J. Franklin and K. Goldberg, "A Methodology for Learning, Analyzing, and Mitigating Social Influence Bias in Recommender Systems", *ACM Conference on Recommender Systems*, pp. 137–144, Foster City, California, United States, 2014.

117. Ma, H., I. King and M. R. Lyu, "Learning to Recommend With Social Trust Ensemble", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 203–210, Boston, Massachusetts, United States, 2009.

118. Tang, J., H. Gao and H. Liu, "mTrust: Discerning Multi-Faceted Trust in a Connected World", *ACM International Conference on Web Search and Data Mining*,

pp. 93–102, Seattle, Washington, United States, 2012.

119. Chaney, A. J., D. M. Blei and T. Eliassi-Rad, "A Probabilistic Model for Using Social Networks in Personalized Item Recommendation", *ACM Conference on Recommender Systems*, pp. 43–50, Vienna, Austria, 2015.

120. Wang, X., S. C. Hoi, M. Ester, J. Bu and C. Chen, "Learning Personalized Preference of Strong and Weak Ties for Social Recommendation", *World Wide Web Conference*, pp. 1601–1610, Perth, Australia, 2017.

121. Dwork, C., M. Hardt, T. Pitassi, O. Reingold and R. Zemel, "Fairness Through Awareness", *Innovations in Theoretical Computer Science*, pp. 214–226, Cambridge, Massachusetts, United States, 2012.

122. Calders, T., F. Kamiran and M. Pechenizkiy, "Building Classifiers With Independency Constraints", *IEEE International Conference on Data Mining*, pp. 13–18, Miami, Florida, United States, 2009.

123. Hardt, M., E. Price and N. Srebro, "Equality of Opportunity in Supervised Learning", *Conference on Neural Information Processing Systems*, pp. 3315–3323, Barcelona, Spain, 2016.

124. Yang, K. and J. Stoyanovich, "Measuring Fairness in Ranked Outputs", *International Conference on Scientific and Statistical Database Management*, pp. 1–6, Chicago, Illinois, United States, 2017.

125. Abdollahpouri, H., R. Burke and B. Mobasher, "Managing Popularity Bias in Recommender Systems With Personalized Re-Ranking", *arXiv Preprint arXiv:1901.07555*, 2019.

126. Abdollahpouri, H., R. Burke and B. Mobasher, "Controlling Popularity Bias in Learning-to-Rank Recommendation", *ACM Conference on Recommender Systems*, pp. 42–46, Como, Italy, 2017.

127. Rosenbaum, P. R. and D. B. Rubin, "The Central Role of the Propensity Score in Observational Studies for Causal Effects", *Biometrika*, Vol. 70, No. 1, pp. 41–55, 1983.

128. Vapnik, V., "Principles of Risk Minimization for Learning Theory", *Conference on Neural Information Processing Systems*, pp. 831–838, Denver, Colorado, United States, 1991.

129. Joachims, T., A. Swaminathan and T. Schnabel, "Unbiased Learning-to-Rank With Biased Feedback", *ACM International Conference on Web Search and Data Mining*, pp. 781–789, Cambridge, United Kingdom, 2017.

130. Bradley, R. A. and M. E. Terry, "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons", *Biometrika*, Vol. 39, No. 3, pp. 324–345, 1952.

131. Plackett, R. L., "The Analysis of Permutations", *Applied Statistics*, pp. 193–202, 1975.

132. Herlocker, J. L., J. A. Konstan, L. G. Terveen and J. T. Riedl, "Evaluating Collaborative Filtering Recommender Systems", *ACM Transactions on Information Systems*, Vol. 22, No. 1, pp. 5–53, 2004.

133. Pu, P., L. Chen and R. Hu, "Evaluating Recommender Systems From the User's Perspective: Survey of the State of the Art", *User Modeling and User-Adapted Interaction*, Vol. 22, No. 4, pp. 317–355, 2012.

134. Gopalan, A., S. Mannor and Y. Mansour, "Thompson Sampling for Complex Online Problems", *International Conference on Machine Learning*, pp. 100–108, Beijing, China, 2014.

135. Saha, A. and A. Gopalan, "Combinatorial Bandits With Relative Feedback", *Conference on Neural Information Processing Systems*, pp. 983–993, Vancouver,

Canada, 2019.

136. Yue, Y., J. Broder, R. Kleinberg and T. Joachims, "The K-armed Dueling Bandits Problem", *Journal of Computer and System Sciences*, Vol. 78, No. 5, pp. 1538–1556, 2012.

137. Saha, A. and A. Gopalan, "Battle of Bandits", *Conference on Uncertainty in Artificial Intelligence*, pp. 6–10, Monterey, California, United States, 2018.

138. Komiyama, J. and T. Qin, "Time-Decaying Bandits for Non-stationary Systems", *Web and Internet Economics*, pp. 460–466, Beijing, China, 2014.

139. Levine, N., K. Crammer and S. Mannor, "Rotting Bandits", *Conference on Neural Information Processing Systems*, pp. 3077–3086, Long Beach, California, United States, 2017.

140. Chakrabarti, D., R. Kumar, F. Radlinski and E. Upfal, "Mortal Multi-Armed Bandits", *Conference on Neural Information Processing Systems*, pp. 273–280, Vancouver, Canada, 2009.

141. Gentile, C., S. Li and G. Zappella, "Online Clustering of Bandits", *International Conference on Machine Learning*, pp. 754–765, Beijing, China, 2014.

142. Li, S., A. Karatzoglou and C. Gentile, "Collaborative Filtering Bandits", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 539–548, Pisa, Italy, 2016.

143. Zhao, X., W. Zhang and J. Wang, "Interactive Collaborative Filtering", *ACM International Conference on Information & Knowledge Management*, pp. 1411–1420, San Francisco, California, United States, 2013.

144. Kawale, J., H. H. Bui, B. Kveton, L. Tran-Thanh and S. Chawla, "Efficient Thompson Sampling for Online Matrix-Factorization Recommendation", *Con-*

ference on Neural Information Processing Systems, pp. 1297–1305, Montréal, Canada, 2015.

145. Kveton, B., C. Szepesvári, Z. Wen and A. Ashkan, "Cascading Bandits: Learning to Rank in the Cascade Model", *International Conference on Machine Learning*, pp. 767–776, Lille, France, 2015.

146. Kveton, B., Z. Wen, A. Ashkan and C. Szepesvári, "Combinatorial Cascading Bandits", *Conference on Neural Information Processing Systems*, pp. 1450–1458, Montréal, Canada, 2015.

147. Zong, S., H. Ni, K. Sung, N. R. Ke, Z. Wen and B. Kveton, "Cascading Bandits for Large-Scale Recommendation Problems", *Conference on Uncertainty in Artificial Intelligence*, New York City, New York, United States, 2016.

148. Komiyama, J., J. Honda and A. Takeda, "Position-Based Multiple-Play Bandit Problem With Unknown Position Bias", *Conference on Neural Information Processing Systems*, pp. 4998–5008, Long Beach, California, United States, 2017.

149. Ermis, B., P. Ernst, Y. Stein and G. Zappella, "Learning to Rank in the Position Based Model With Bandit Feedback", *ACM International Conference on Information & Knowledge Management*, pp. 2405–2412, Virtual Event, 2020.

150. Zoghi, M., T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari and Z. Wen, "Online Learning to Rank in Stochastic Click Models", *International Conference on Machine Learning*, pp. 4199–4208, Sydney, Australia, 2017.

151. Lattimore, T., B. Kveton, S. Li and C. Szepesvári, "TopRank: A Practical Algorithm for Online Stochastic Ranking", *Conference on Neural Information Processing Systems*, pp. 3949–3958, Montréal, Canada, 2018.

152. Liu, Y. and L. Li, "A Map of Bandits for E-Commerce", *Workshop on Multi-Armed Bandits and Reinforcement Learning: Advancing Decision Making in E-*

Commerce and Beyond in Conjunction with ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Gold Coast, Australia, 2021.

153. Wu, H. and X. Liu, "Double Thompson Sampling for Dueling Bandits", *Conference on Neural Information Processing Systems*, pp. 649–657, Barcelona, Spain, 2016.

154. Sui, Y., V. Zhuang, J. W. Burdick and Y. Yue, "Multi-dueling Bandits with Dependent Arms", *Conference on Uncertainty in Artificial Intelligence*, Sydney, Australia, 2017.

155. Sui, Y., M. Zoghi, K. Hofmann and Y. Yue, "Advancements in Dueling Bandits", *International Joint Conference on Artificial Intelligence*, pp. 5502–5510, Stockholm, Sweden, 2018.

156. Davidson, R. R. and D. L. Solomon, "A Bayesian Approach to Paired Comparison Experimentation", *Biometrika*, Vol. 60, No. 3, pp. 477–487, 1973.

157. Leonard, T., "An Alternative Bayesian Approach to the Bradley-Terry Model for Paired Comparisons", *Biometrics*, pp. 121–132, 1977.

158. Carlson, B. C., "Appell Functions and Multiple Averages", *SIAM Journal on Mathematical Analysis*, Vol. 2, No. 3, pp. 420–430, 1971.

159. Schnabel, T., A. Swaminathan, A. Singh, N. Chandak and T. Joachims, "Recommendations as Treatments: Debiasing Learning and Evaluation", *International Conference on Machine Learning*, pp. 1670–1679, New York City, New York, United States, 2016.

160. Wang, Y., D. Liang, L. Charlin and D. M. Blei, "Causal Inference for Recommender Systems", *ACM Conference on Recommender Systems*, pp. 426–431, Virtual Event, 2020.

161. Dickey, J. M., "Multiple Hypergeometric Functions: Probabilistic Interpretations and Statistical Uses", *Journal of the American Statistical Association*, Vol. 78, No. 383, pp. 628–637, 1983.

162. Dickey, J. M., J.-M. Jiang and J. B. Kadane, "Bayesian Methods for Censored Categorical Data", *Journal of the American Statistical Association*, Vol. 82, No. 399, pp. 773–781, 1987.

163. Hankin, R. K. S., "A Generalization of the Dirichlet Distribution", *Journal of Statistical Software*, Vol. 33, No. 11, pp. 1–18, 2010.

164. Jiang, T. J., J. B. Kadane and J. M. Dickey, "Computation of Carlson's Multiple Hypergeometric Function R for Bayesian Applications", *Journal of Computational and Graphical Statistics*, Vol. 1, No. 3, pp. 231–251, 1992.

165. Duane, S., A. D. Kennedy, B. J. Pendleton and D. Roweth, "Hybrid Monte Carlo", *Physics Letters B*, Vol. 195, No. 2, pp. 216–222, 1987.

166. Hunter, D. R., "MM Algorithms for Generalized Bradley-Terry Models", *The Annals of Statistics*, Vol. 32, No. 1, pp. 384–406, 2004.

167. Sopher, B. and G. Gigliotti, "Intransitive Cycles: Rational Choice or Random Error? An Answer Based on Estimation of Error Rates With Experimental Data", *Theory and Decision*, Vol. 35, No. 3, pp. 311–336, 1993.

168. Regenwetter, M., J. Dana and C. P. Davis-Stober, "Transitivity of Preferences", *Psychological Review*, Vol. 118, No. 1, pp. 42–56, 2011.

169. Doucet, A., A. M. Johansen *et al.*, "A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later", *Handbook of Nonlinear Filtering*, Vol. 12, No. 3, pp. 656–704, 2009.

170. Neal, R. M., "Slice Sampling", *The Annals of Statistics*, Vol. 31, No. 3, pp. 705–

767, 2003.

171. Gilks, W. R., N. G. Best and K. K. C. Tan, "Adaptive Rejection Metropolis Sampling Within Gibbs Sampling", *Journal of the Royal Statistical Society*, Vol. 44, No. 4, pp. 455–472, 1995.

172. Gündoğdu, I., *Sequential Monte Carlo Approach to Inference in Bayesian Choice Models*, M.Sc. Thesis, Bogazici University, 2019.

173. Nie, X., X. Tian, J. Taylor and J. Zou, "Why Adaptively Collected Data Have Negative Bias and How to Correct for It", *International Conference on Artificial Intelligence and Statistics*, pp. 1261–1269, Lanzarote, Canary Islands, 2018.

174. Cantador, I., P. Brusilovsky and T. Kuflik, "2nd Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011)", *ACM Conference on Recommender Systems*, Chicago, Illinois, United States, 2011.

175. Komiyama, J., J. Honda, H. Kashima and H. Nakagawa, "Regret Lower Bound and Optimal Algorithm in Dueling Bandit Problem", *Conference on Learning Theory*, pp. 1141–1154, Paris, France, 2015.

176. Zoghi, M., S. Whiteson, R. Munos and M. Rijke, "Relative Upper Confidence Bound for the K-Armed Dueling Bandit Problem", *International Conference on Machine Learning*, pp. 10–18, Beijing, China, 2014.

177. Yue, Y. and T. Joachims, "Beat the Mean Bandit", *International Conference on Machine Learning*, pp. 241–248, Bellevue, Washington, United States, 2011.

178. Urvoy, T., F. Clerot, R. Féraud and S. Naamane, "Generic Exploration and K-Armed Voting Bandits", *International Conference on Machine Learning*, pp. 91–99, Atlanta, Georgia, United States, 2013.

179. Ailon, N., Z. Karnin and T. Joachims, "Reducing Dueling Bandits to Cardinal

Bandits", *International Conference on Machine Learning*, pp. 856–864, Beijing, China, 2014.

180. Zoghi, M., S. A. Whiteson, M. De Rijke and R. Munos, "Relative Confidence Sampling for Efficient on-Line Ranker Evaluation", *ACM International Conference on Web Search and Data Mining*, pp. 73–82, New York, New York, United States, 2014.

181. Zoghi, M., Z. S. Karnin, S. Whiteson and M. De Rijke, "Copeland Dueling Bandits", *Conference on Neural Information Processing Systems*, pp. 307–315, Montréal, Canada, 2015.

182. Komiyama, J., J. Honda and H. Nakagawa, "Copeland Dueling Bandit Problem: Regret Lower Bound, Optimal Algorithm, and Computationally Efficient Algorithm", *International Conference on Machine Learning*, pp. 1235–1244, New York City, New York, United States, 2016.

183. Jagerman, R., H. Oosterhuis and M. de Rijke, "To Model or to Intervene: A Comparison of Counterfactual and Online Learning to Rank From User Interactions", *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 15–24, Paris, France, 2019.

184. Oosterhuis, H. and M. de Rijke, "Unifying Online and Counterfactual Learning to Rank: A Novel Counterfactual Estimator That Effectively Utilizes Online Interventions", *ACM International Conference on Web Search and Data Mining*, pp. 463–471, Virtual Event, 2021.

185. Caron, F. and A. Doucet, "Efficient Bayesian Inference for Generalized Bradley–Terry Models", *Journal of Computational and Graphical Statistics*, Vol. 21, No. 1, pp. 174–196, 2012.

186. Yellott Jr, J. I., "The Relationship Between Luce's Choice Axiom, Thurstone's

Theory of Comparative Judgment, and the Double Exponential Distribution", *Journal of Mathematical Psychology*, Vol. 15, No. 2, pp. 109–144, 1977.

187. Gumbel, E. J., *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, Tech. rep., 1954.

188. Balog, M., N. Tripuraneni, Z. Ghahramani and A. Weller, "Lost Relatives of the Gumbel Trick", *International Conference on Machine Learning*, pp. 371–379, Sydney, Australia, 2017.

189. Carpenter, B., A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li and A. Riddell, "Stan: A Probabilistic Programming Language", *Journal of Statistical Software*, Vol. 76, No. 1, 2017.

190. Guiver, J. and E. Snelson, "Bayesian Inference for Plackett-Luce Ranking Models", *International Conference on Machine learning*, pp. 377–384, Montréal, Canada, 2009.

191. Falahatgar, M., A. Orlitsky, V. Pichapati and A. T. Suresh, "Maximum Selection and Ranking under Noisy Comparisons", *International Conference on Machine Learning*, pp. 1088–1096, Sydney, Australia, 2017.

192. Szörényi, B., R. Busa-Fekete, A. Paul and E. Hüllermeier, "Online Rank Elicitation for Plackett-Luce: A Dueling Bandits Approach", *Conference on Neural Information Processing Systems*, pp. 604–612, Montréal, Canada, 2015.

193. Jamieson, K. G. and R. Nowak, "Active Ranking Using Pairwise Comparisons", *Conference on Neural Information Processing Systems*, pp. 2240–2248, Granada, Spain, 2011.

194. Busa-Fekete, R., E. Hüllermeier and B. Szörényi, "Preference-based Rank Elicitation using Statistical Models: The Case of Mallows", *International Conference on Machine Learning*, Vol. 32, Beijing, China, 2014.

195. Busa-Fekete, R., B. Szorenyi, W. Cheng, P. Weng and E. Hüllermeier, "Top-k Selection based on Adaptive Sampling of Noisy Preferences", *International Conference on Machine Learning*, pp. 1094–1102, Atlanta, Georgia, United States, 2013.

196. Mohajer, S., C. Suh and A. Elmahdy, "Active Learning for Top-K Rank Aggregation from Noisy Comparisons", *International Conference on Machine Learning*, pp. 2488–2497, Sydney, Australia, 2017.

APPENDIX A: APPENDICES TO SECTION 4

A.1. Posterior Predictive Inference in the Dirichlet-Luce Model

We can write $p(k_{1:T} \mid C_{1:T}, \alpha, \beta)$, probability of a sequence of choices conditioned on a sequence of presentations and hyperparameters, as ratios of $\mathcal{R}$ functions,

$$\begin{aligned} p(k_{1:T} \mid C_{1:T}, \alpha, \beta) &= \int_{\Delta} p(\theta \mid \alpha, \beta) p(k_{1:T} \mid C_{1:T}, \theta) d\theta \\ &= \int_{\Delta} p(\theta \mid \alpha, \beta) \prod_{k} \theta_k^{y_k} \prod_{C \in \mathcal{C}} (\theta^\top \mathbf{z}_{:,C})^{-\mu(C)} d\theta \\ &= \frac{\prod_k (\alpha_k)_{(y_k)}}{(\sum_k \alpha_k)_{(N)}} \frac{\mathcal{R}(\alpha + \mathbf{y}, Z, \beta + \mu)}{\mathcal{R}(\alpha, Z, \beta)}, \end{aligned} \tag{A.1}$$

where the notation $(x)_{(n)} = \frac{\Gamma(x+n)}{\Gamma(x)}$ denotes the rising factorial and $\mathbf{y}$ is the vector $(y_1, y_2, \cdots, y_K)$

This gives the predictive preference of option k (over all $[K]$) as

$$\begin{aligned} p(k|[K], \alpha, \beta) &= \int_{\Delta} p(\theta \mid \alpha, \beta) \theta_k d\theta \\ &= \frac{\alpha_k}{\sum_j \alpha_j} \frac{\mathcal{R}(\alpha + \mathbf{k}, Z, \beta + [0, 0, \cdots, 1])}{\mathcal{R}(\alpha, Z, \beta)}, \end{aligned}$$

where $\mathbf{k}$ is the indicator vector of size K where all but the k-th element are 0.

A.2. Bayesian Learning of Preferences from Restricted Choices

Although the assumptions implied in [29] come with the trade-off of failing to capture multi-modal preferences, they allow fast inference from a small fraction of possible presentations. Motivated by the psychology literature, for the multi-modal case a mixture extension is conjectured in the main report.

Dirichlet-Luce is not the only Bayesian choice model we can write that assumes the restricted multinomial likelihood, but it is a natural one considering an interactive system. Let us review an alternative model, due to [185], and then highlight our main motivation.

The choice model in [29] corresponds to a random utility model with Gumbel noise [186]. That is to say, the multinomial choice can be modeled with the so-called *Gumbel-max* procedure [187]. Assuming positive mean utilities u_κ, $\forall \kappa \in [K]$, the following procedure,

$$\begin{aligned} z_\kappa &\sim \mathcal{GU}(\log(u_\kappa), 1), \\ k &= \arg\max_{\kappa \in C} z_\kappa, \end{aligned}$$

where $\mathcal{GU}(l, 1)$ denotes the Gumbel distribution with location l and scale 1, gives the restricted multinomial [29]:

$$p(k \mid C) = \frac{u_k}{\sum_{\kappa \in C} u_\kappa}.$$

A Bayesian approach would treat an unnormalized choice probabilitiy $u_k \sim \mathcal{U}_k$ as a random variable. A notable method is due to [185]. There, the multinomial choice restricted to a presentation is cast as an *exponential race* with the following generative model:

$$\begin{aligned} u_\kappa &\sim \mathcal{G}(a, b), \\ z_{t,\kappa} &\sim \mathcal{E}(u_\kappa), \\ k_t &= \arg\min_{i, c_{t,i}=1} z_{t,i}. \end{aligned}$$

Here, $\mathcal{G}(a, b)$ denotes the gamma distribution with shape a and the inverse scale b (*i.e.*, $\mathbb{E}U_\kappa = a/b$). $\mathcal{E}(u)$ denotes the exponential distribution with rate u. We refer to [188] for the relation of the exponential race and the Gumbel-max trick. In the Caron-Doucet model [185], restricted to pairwise choices, we define $x_{t,i,j} = \min(z_{t,i}, z_{t,j})$. Then of course, $x_{t,i,j} \sim \mathcal{E}(u_i + u_j)$, and $\sum_t x_{t,i,j} = X_{i,j} \sim \mathcal{G}(\mu(\{i, j\}), u_i + u_j)$. The complete

conditionals, obtained as,

$$\begin{aligned} x_{i,j} \mid \mathbf{u}, \mathbf{C}, \mathbf{k} &\sim \mathcal{G}(\mu(\{i,j\}), u_i + u_j), \\ u_i \mid X, \mathbf{C}, \mathbf{k} &\sim \mathcal{G}\left(a + y_i, b + \sum_{i<j, \mu(\{i,j\})>0} X_{i,j} + \sum_{i>j, \mu(\{i,j\})>0} X_{j,i}\right) \end{aligned}$$

can be utilized to implement a Gibbs sampler. The statistics y and μ are identical to the Dirichlet-Luce case. Both models operate with the same statistics, converge to similar (when normalized versions of posterior u_κ's are used in [185]) preferences, and require sampling for inference.

In the main text, in the case where the observations comply with the choice axiom [29], the following scenario was described as an example where $K-1$ distinct pairwise preferences would be sufficient to recover the preferences. There, we first pick an arbitrary pivot option, and then observe preferences from pairs of options formed by the pivot and one of the other $K-1$ options. We highlight that despite seemingly combinatorial dimensionality of the statistics, both the Dirichlet-Luce model and Caron-Doucet model [185] converge to a reasonable estimate of overall preferences, *i.e.*, $p(k \mid [K])$ (Figure A.1). There, the Gibbs sampler in [185], and a Hamiltonian Monte Carlo [165] sampler implemented in Stan probabilistic programming language [189] for the Dirichlet-Luce model are used to obtain posterior samples.

If the task of the hypothesized interactive system was only to infer preferences given a batch of choice observations, other choice models, as, *e.g.*, [185] described for pairwise preferences case and [190] for L-wise preferences, would serve our purposes. In fact, they converge to similar preferences conditioned on a batch of choice observations. But the system's task is two-fold. In addition to the *inference task*, *the discovery task*—as K is so large in practice, *i.e.*, to assume responsibility for *finding all good items* without overlooking any alternative [132], is on the learning system.

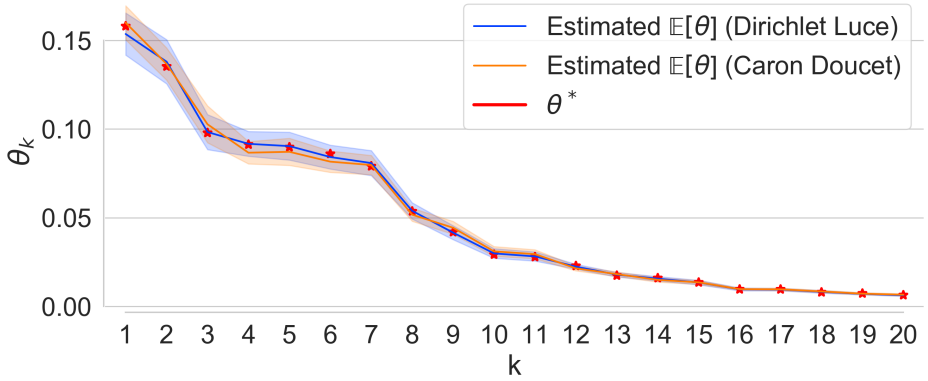


Figure A.1. Estimated (from 500 samples) posterior mean of θ (over 20 runs) conditioned on *many* (2000) choices from pairwise presentations constructed with a randomly selected pivot option and the others. θ^* is ordered, and $\mathbb{E}[\theta]$ estimates are conformably permuted for visualization. Shaded regions denote the standard deviation.

We need a sequential decision making procedure, an online presentation mechanism. The fundamental motivation to devising Dirichlet-Luce is that it gives us a conjugate density which can be directly utilized in the interactive learning scenario. The resulting procedure is straightforward, this sequential decision making procedure can be implemented with a sequential sampler, and finally, although the posterior is updated at each interaction round, $k_{1:T} \mid C_{1:T}, \alpha, \beta$, the sequence of choices when choice probabilities are integrated out, is *exchangeable* (as can be seen from Equation A.1).

The preference learning illustration in the main text is not the only example that we can devise to demonstrate the efficiency of the inference procedure. We will presently give other scenarios, which again, utilize the underlying model assumptions. The first one is to ensure consistency, and the second one is by analogy to stochastic ranking algorithms:

In another scenario, we assume we observe preferences to presentations $C_1 = \{1, 2\}, C_2 = \{2, 3\}, \cdots, C_{K-1} = \{K-1, K\}$. Note that here, $\nu_{k,C}$ (the underlying contingency table in the main text) is ambiguous since the statistics that the model

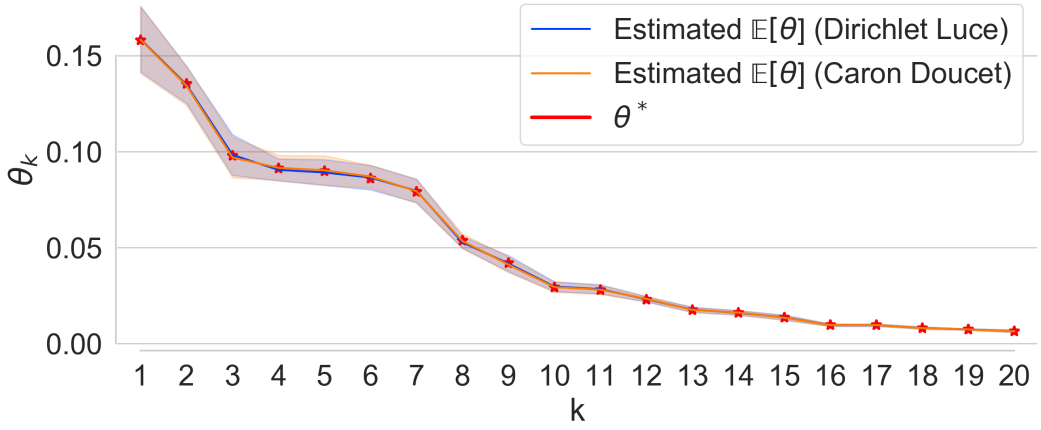
utilizes include only the margins of ν, and an option is presented together with multiple options. The dimensionality of μ is again $K-1$. In this scenario, too, both in the case of ours and in [185], the posterior predictive probabilities converge to latent preferences (Figure A.2(a))

Assuming a single preference vector underlying choice probabilities leads to the implicit assumption that preferences are (stochastically) transitive—options admit a total ordering in their probability of being chosen against all others. By analogy, a sorting algorithm—relying on transitivity—would find the ordering of K options with $O(K \log K)$ pairwise comparisons. Similarly, in the field of active learning, stochastic ranking from pairwise preferences has been widely explored [191–196]. Here too, the objective is to attain a probably approximately correct ranking of preferences within low sample complexity. Analogously to our case, the algorithm receives a stochastic comparison (random preference) feedback instead of a deterministic comparison result.

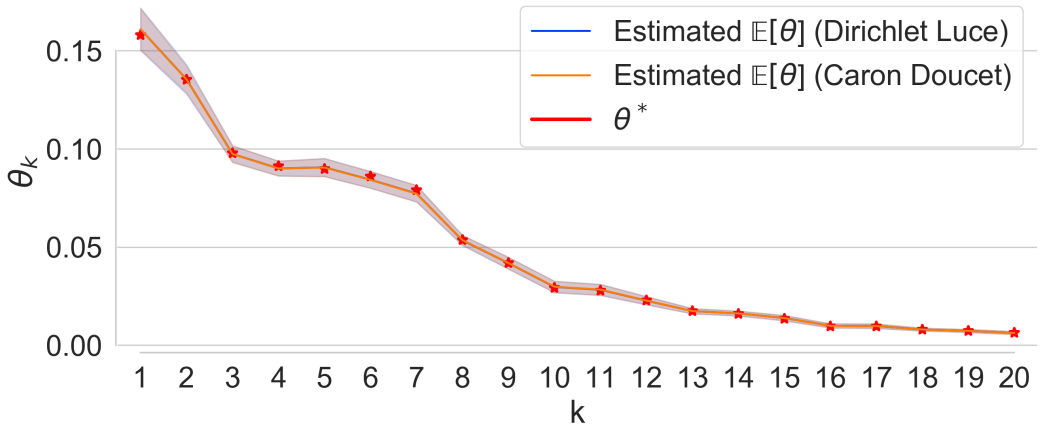
In this light, we explore whether the Dirichlet-Luce and Caron-Doucet [185] constructions are able to recover a good representation of preferences, given a number of samples on the same order as a stochastic ranking algorithm. We take the *Merge-Rank* algorithm [191], a stochastic variant of the *mergesort* algorithm for active subset selection—selecting pairs of options to ask for a preference feedback from the environment. We fix a preference vector θ^* and run the Merge-Rank algorithm, generating a set of pairwise comparisons $C_{1:T}$ (corresponding to our presentations) and stochastic feedback $k_{1:T}$ (corresponding to choices). In Figure A.2(b), we find that θ^* can be recovered accurately based on the same number of samples required by a stochastic ranker. As in mergesort, the number of unique presentations is $O(K \log K)$.

A.3. Sequential Sampling Procedure in Dirichlet-Luce Bandit Algorithm

The complete online learning to recommend algorithm along with the sequential sampling routine is listed in Figure A.3. Detailed treatment of the move kernel and computational aspects of full-conditional density evaluations can be found in [172].



(a) Simulated data obtained by repeatedly (2000 times each) presenting $\{1,2\},\{2,3\},\ldots,\{K-1,K\}$



(b) Merge-Rank data, with bias and confidence parameters $\epsilon = 0.05$ and $\delta = 0.1$ of presentations due to transitivity and with simulated choices

Figure A.2. Estimated (from 500 samples) posterior mean (over 20 runs) conditioned on a constructed ((a)) or actively selected data set *draws* ((b)). θ^* is ordered, and $\mathbb{E}[\theta]$ estimates are conformably permuted for visualization. Shaded regions denote the standard deviation

Input T : Number of interactions, N : Number of particles

Initialize α, and set $\beta \leftarrow \beta_0$

$\mu(C) \leftarrow 0$ for all $C \in \mathcal{C}$, $y_k \leftarrow 0$ for all $k \in [K]$

$\theta^{(i)} \sim \mathcal{D}(\mathbf{1}), \quad i = 1, 2, \ldots, N$, $w_0^{(i)} = 1, \quad i = 1, 2, \ldots, N$

for $t = 1$ **to** T **do**

Sample θ_t from the particle cloud $\{\theta^{(i)}\}_{i=1}^N$, with $P(\theta_t \leftarrow \theta^{(i)}) = w_t^{(i)} / \sum_{i'} w_t^{(i')}$

Form C_t with top L elements of θ_t and get preference feedback k_t to C_t

$\mu(C_t) \leftarrow \mu(C_t) + 1$ {Update sufficient statistics}

$y_{k_t} \leftarrow y_{k_t} + 1$

$w_t^{(i)} = w_{t-1}^{(i)} \dfrac{\theta_{k_t}^{(i)}}{\sum_{\kappa \in C_t} \theta_\kappa^{(i)}}$

$\text{ESS}_t = (\sum_i w_t^{(i)})^2 / \sum_i (w_t^{(i)})^2$

if $ESS_t < 0.5 * N$ **then**

for all $j \in \{1, 2, \ldots, N\}$ **do**

$\theta^{(j)} \leftarrow \theta^{(i)}$ with probability $w_t^{(i)} / \sum_{i'} w_t^{(i')}$ {Resample}

end for

for $i = 1$ **to** N **do**

for $j = 1$ **to** $K - 1$ **do**

$r = 1 - \sum_{j' \neq j} \theta_{j'}$

$\hat{\theta}_j^{(i)} \sim \mathcal{U}(0, r)$ {Propose a coordinate update for θ_j}

$\lambda = \min\{1, f_j(\hat{\theta}_j^{(i)}) / f_j(\theta_j^{(i)})\}$ {$f_j(.)$ is the full-conditional density of θ_j}

$u \sim \mathcal{U}(0, 1)$

if $u < \lambda$ **then**

$\theta_j^{(i)} = \hat{\theta}_j^{(i)}$ {Accept the proposed coordinate update}

end if

end for

$w_t^{(i)} = 1$ {Reset particle weights}

end for

end if

end for

Figure A.3. Complete specification for the Dirichlet-Luce bandit algorithm.

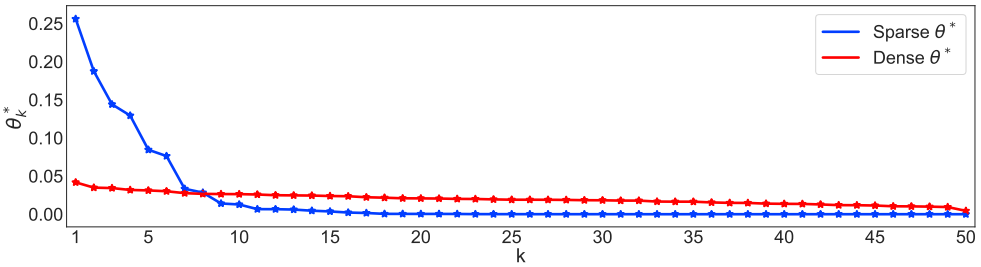
A.4. Simulation Details

All dueling bandit simulations were run based on a sparse and then a dense θ^*. Figure A.4(a) shows these 50-dimensional vectors. One hyperparameter for the *MM* algorithm is set based on a held-out simulation assuming the same θ^*s. The Last.FM simulation is performed using a θ^* that is inferred by fitting LDA to artist listening data. The top-50 artists in terms of probability of being listened according to this θ^* are shown in Figure A.5

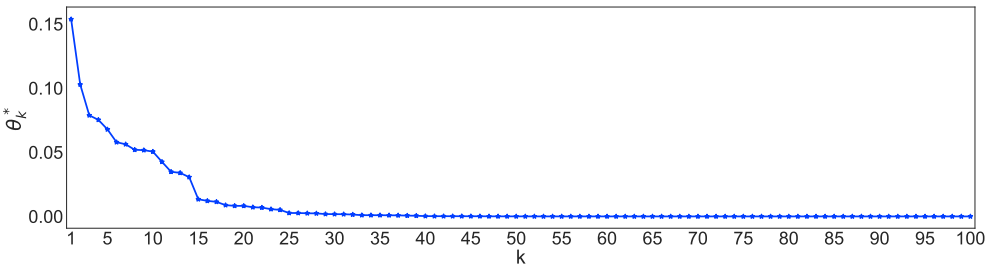
For each experiment in the online learning to rank experiments, TopRank hyperparameter is selected based on a held-out simulation assuming the same θ^*. Finally with presentation size $L = 5$, we used the θ^* in Figure A.4(c)

A.5. Reuse of Graphic Elements in the Thesis

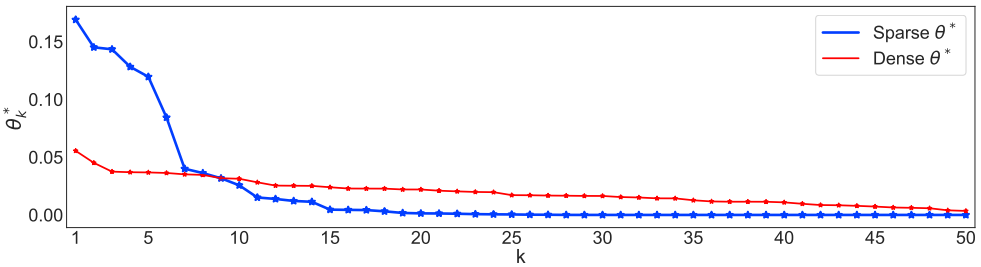
The graphic elements included in this thesis have been created by the author and are contained in the author's previously published research studies. They are reused in this thesis in accordance with the publishing agreements with the corresponding publishers.



(a) θ^* in dueling bandits experiments



(b) θ^* (100-dimensional) in top-2 performance experiments



(c) θ^* in online learning to rank experiments

Figure A.4. Simulated θ^*'s, sorted in descending order for visualization

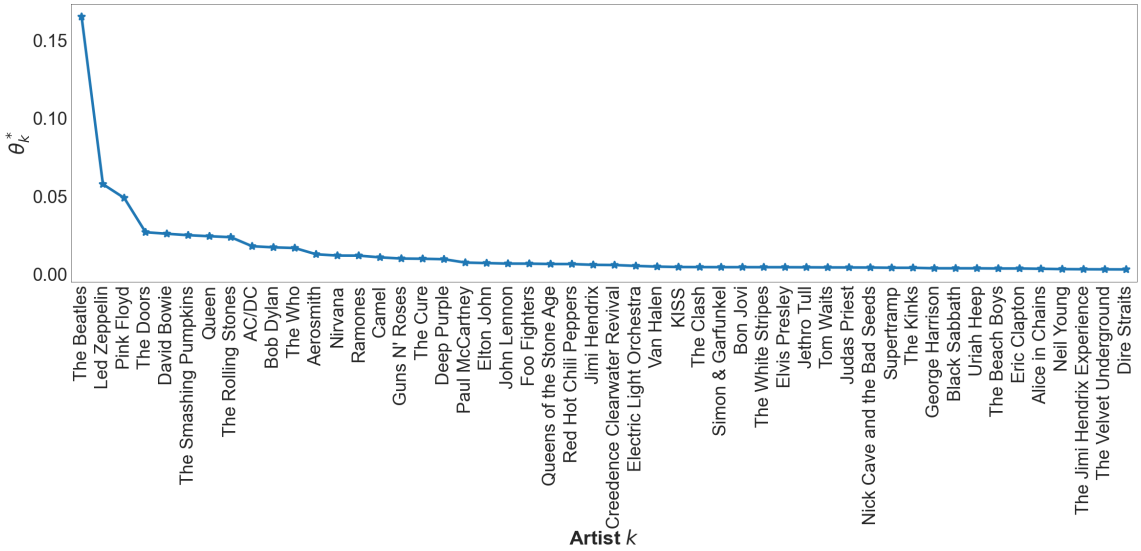


Figure A.5. Top-50 elements of the θ^* used as preference probabilities in Last.FM simulations