EXPLORAÇÃO E EXPLOTAÇÃO EFICIENTE PARA RECOMENDAÇÃO SEQUENCIAL DE MÚSICAS

PEDRO DALLA VECCHIA CHAVES

EXPLORAÇÃO E EXPLOTAÇÃO EFICIENTE PARA RECOMENDAÇÃO SEQUENCIAL DE MÚSICAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Rodrygo L. T. Santos

Belo Horizonte

Outubro de 2021

PEDRO DALLA VECCHIA CHAVES

EFFICIENT EXPLORATION AND EXPLOITATION FOR SEQUENTIAL MUSIC RECOMMENDATION

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Rodrygo L. T. Santos

Belo Horizonte

October 2021

Chaves, Pedro Dalla Vecchia.

C512e Efficient exploration and exploitation for sequential music recommendation [manuscrito] / Pedro Dalla Vecchia Chaves. – 2021.
xiv, 54 f. il.

Orientador: Rodrygo Luis Teodoro Santos.
Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.49-54
.
1. Computação – Teses. 2. Sistemas de recomendação – Teses. 3. Recomendação online de músicas – Teses. 4. Apredizado do computador – Teses. 5. Aprendizado de ranqueamento – Teses I. Santos, Rodrygo Luis Teodoro II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*73(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa CRB 6ª Região nº 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Efficient Exploration and Exploitation for Sequential Music Recommendation

PEDRO DALLA VECCHIA CHAVES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RODRYGO LUIS TEODORO SANTOS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. FABRÍCIO MURAI FERREIRA
Departamento de Ciência da Computação - UFMG

PROF. Flávio Vinícius Diniz de Figueiredo
Departamento de Ciência da Computação - UFMG

PROF. MARCELO GARCIA MANZATO
Instituto de Ciências Matemáticas e de Computação - USP

Belo Horizonte, 8 de Outubro de 2021.

To everyone who contributed, in any form, to this achievement.

Acknowledgments

First, I want to thank my parents Mônica and Alexandre, both teachers and researchers, for, among other thousand things, inspiring me and teaching me the importance and beauty of research and education. Also, I would like to thank my girlfriend Beatriz for all the love, support, and patience, especially in these pandemic times. For my close friends from Colégio Santo Antônio, I want to thank them for their incredible friendship and support.

I want to thank both Rodrygo and Bruno, my official and unofficial advisors, for their extraordinary commitment, patience, and support. Without you, none of this would have been possible. For my colleagues at UFMG, especially Edson, Ivan, and Luiz, I want to thank you for all the support and friendship during these six years.

Lastly, I want to thank all my current and previous work colleagues at Denox, Kunumi, Wildlife Studios, and Nubank for the support and knowledge sharing.

"Um passo à frente e você não está mais no mesmo lugar."
(Chico Science)

Resumo

Os serviços de streaming de música dependem fortemente de sistemas de recomendação para adquirir, envolver e reter usuários. Um componente notável desses serviços são as listas de reprodução, que podem ser geradas dinamicamente de maneira sequencial com base no feedback do usuário durante uma sessão de escuta. Recentemente, métodos baseados em aprendizagem online para ranqueamento se mostraram eficazes ao aproveitar esse feedback para aprender as preferências dos usuários no espaço de representação vetorial de músicas. No entanto, essas abordagens podem sofrer de convergência lenta como resultado de seu componente de exploração aleatório e ficar presas em mínimos locais devido ao seu componente de explotação agnóstico à sessão. Para superar essas limitações, propomos um novo método de aprendizagem online para ranqueamento que explora com eficiência o espaço de modelos de recomendação candidatos, restringindo-se ao complemento ortogonal do subespaço de direções de exploração anteriores de baixo desempenho. Além disso, para ajudar a superar os mínimos locais, propomos um componente de explotação ciente de sessão que aproveita de forma adaptativa o melhor modelo atual durante as atualizações do modelo. Nossa criteriosa avaliação usando sessões de escuta simuladas na plataforma Last.fm demonstra melhorias substanciais em relação às abordagens estado da arte no desempenho em estágio inicial e convergência geral de longo prazo.

Palavras-chave: Recomendação sequencial de músicas, exploração eficiente, explotação adaptativa, aprendizado para ranqueamento online, feedback implícito.

Abstract

Music streaming services heavily rely upon recommender systems to acquire, engage, and retain users. One notable component of these services are playlists, which can be dynamically generated in a sequential manner based on the user's feedback during a listening session. Online learning to rank approaches have recently been shown effective at leveraging such feedback to learn users' preferences in the space of song features. Nevertheless, these approaches can suffer from slow convergence as a result of their random exploration component and get stuck in local minima as a result of their session-agnostic exploitation component. To overcome these limitations, we propose a novel online learning to rank approach which efficiently explores the space of candidate recommendation models by restricting itself to the orthogonal complement of the subspace of previous underperforming exploration directions. Moreover, to help overcome local minima, we propose a session-aware exploitation component which adaptively leverages the current best model during model updates. Our thorough evaluation using simulated listening sessions from Last.fm demonstrates substantial improvements over state-of-the-art approaches regarding early-stage performance and overall long-term convergence.

Keywords: Sequential music recommendation, efficient exploration, adaptive exploitation, online learning to rank, implicit feedback.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Modern music streaming services have changed the way people consume music, offering massively diverse song catalogs while optimizing the user experience through personalization [Celma, 2010; Hagen, 2015; Kamehkhosh et al., 2020]. A particularly prominent feature of these services are playlists, which offer users a somewhat coherent selection of songs for sequential consumption [Hagen, 2015]. Essentially, playlist curation practices allow more user ownership and personalization in a context where streaming services offer a possible homogeneous content, however for a wide and diverse audience. Nevertheless, due to the volume of tracks available in the music streaming collection, the manual creation of playlists requires much effort and is often infeasible [Kamehkhosh et al., 2020], being one of the reasons why automatic playlist generation is still heavily explored by researchers [Hauver and French, 2001; Logan, 2002; Pampalk et al., 2005; Baccigalupo and Plaza, 2006; Hariri et al., 2012; Bonnin and Jannach, 2014; Pelle, 2017; Vall et al., 2018].

To address the automatic playlist generation problem, static approaches leverage the users' historical listening patterns through collaborative filtering [Hauver and French, 2001], content-based, [Logan, 2002], case-based [Baccigalupo and Plaza, 2006], and other types of classic recommendation methods. Fundamentally, by taking into account the past history of interactions of a given user, and possibly other similar users, static approaches focus on recommending a fixed list of new songs by leveraging similarities in the past consumed songs, albums, artists, etc. Static approaches typically start from a seed song provided by the user [Bonnin and Jannach, 2014; Pelle, 2017; Vall et al., 2018] to guide the recommendation process. In contrast, dynamic approaches leverage the current context for live adjustments to the user's preferences [Pampalk et al., 2005; Bosteels and Kerre, 2009; Hariri et al., 2012, 2015]. In particular, dynamic approaches focus on the current listening session to evaluate similarity

information of recent and past consumed songs, adaptively recommending new ones based on the user's immediate explicit (e.g. ratings) or implicit (e.g. skips) feedback, in an interactive fashion. While approaches that learn from historical data work well to capture long-term listening habits, recent studies demonstrate significant improvements on session-based recommendations using short-term preferences [Quadrana et al., 2018; Wang et al., 2021].

Of particular interest to this dissertation, dynamic approaches based on contextual bandits interactively learn from the user's feedback by trading off the exploitation of already known user preferences for the exploration of new possibilities [Li et al., 2010]. Among these, online learning to rank (OLTR) approaches are able to scale to large item catalogs by learning a linear model of users' preferences in a continuous space of item features, as opposed to these users' preference for each individual item in the catalog [Yue and Joachims, 2009; Schuth et al., 2016]. OLTR approaches were originally conceived to address classical information retrieval tasks such as document search, requiring click feedback over ranked item lists to discern between promising and poor directions in the feature space. Because sequential music recommendation endows only one item (a song) with feedback (play or skip) at each time step, counterfactual approaches have been proposed to extrapolate this feedback to estimate the potential reward of multiple explored directions [Pereira et al., 2019]. By evaluating the position of a recently skipped song on rankings produced by candidate update directions, counterfactual approaches efficiently learn better recommender models without actually exposing users to multiple, possibly poor, candidate recommendations. Similarly, this dissertation focuses on the sequential music recommendation task where a user provides a seed song to start a listening session and, at each iteration, the recommender model must show one song out of a list of candidate ones, receiving the user's implicit feedback (play or skip) to guide the adaptive learning process.

Despite their relative effectiveness, existing OLTR approaches explore candidate directions uniformly at random, which often leads to poor directions being repeatedly explored over time. That is, at each interaction with the user, current OLTR approaches will produce candidate update directions by randomly sampling in the unit sphere around the current model, represented as a d-dimensional weight vector, without considering the performance of the previously generated directions. Ultimately, random exploration may slow down the learning process while exposing users to bad recommendations along the way [Wang et al., 2018]. Moreover, these approaches uniformly update the current model at each learning step regardless of the effectiveness of the model prior to the update. Particularly, the current model is exploited at a fixed rate in the update phase, without evaluating its capability of generating (un)promising

candidate directions. This fixed exploitation strategy may lead to further slow and suboptimal convergence [Polyak, 1964].

1.1 Dissertation Statement

The statement of this dissertation is that online learning to rank for sequential music recommendation can be improved via enhanced exploration and exploitation of candidate directions. In particular, by restricting the exploration of candidate directions to the orthogonal (null) space of previous underperforming ones we can avoid repeating known poorly performing directions. Combined with the aforementioned exploration strategy, we can also improve exploitation by adaptively leveraging the influence of the current best model in the update phase as well as the estimated performance of candidate exploratory directions, resuting in superior early-stage performance and overall convergence.

1.2 Dissertation Contributions

To ensure early-stage performance while still converging to an effective recommendation model, we propose a novel Online Learning to Rank approach for sequential music recommendation. In particular, to correct the learning course in the event of a negative feedback (a skip), instead of performing random exploration, we explore in the orthogonal complement of the subspace of exploration directions that previously underperformed. As a result, we avoid repeatedly exploring directions known to lead to bad recommendations. To improve exploitation efficiency, we adaptively leverage the current model at each update step depending on the estimated reward of directions spawned from it. Thorough experiments using simulated listening sessions from Last.fm demonstrate the benefits of our proposed approach at learning faster and converging to more effective recommendation models compared to the current state-of-the-art. To sum up, our main contributions are two-fold:

- We propose an online learning approach for sequential music recommendation which avoids exploring unpromising directions while adaptively exploiting promising ones.
- We thoroughly evaluate our proposed approach in comparison to state-of-the-art contextual bandit approaches regarding their learning efficiency and effectiveness.

1.3 Dissertation Outline

Here we present a brief summary of the contents detailed in each chapter of this dissertation:

- **Chapter 2** outlines previous research on automatic playlist generation, online learning to rank (OLTR), and efficient exploration and exploitation. It begins by presenting and contrasting static and dynamic automatic playlist generation approaches. Then, it introduces the OLTR framework, detailing the main proposed methods using a canonical algorithm notation. Lastly, previous literature on efficient exploration and exploitation applied to OLTR algorithms is reviewed.
- **Chapter 3** describes our approach, named Null Space Counterfactual Dueling Bandits (NSCDB), for efficient OLTR focused on the sequential music recommendation task. It outlines and explains the main adaptations and modifications done to improve the exploration and exploitation components of previous approaches, resulting in a novel efficient OLTR algorithm.
- **Chapter 4** describes the experimental setup employed to evaluate the performance of the proposed NSCDB algorithm against the current state-of-the-art.
- **Chapter 5** describes the results of the empirical evaluation of our approach with respect to the current state-of-the-art, presenting an in-depth investigation on the impact of the proposed exploration and exploitation components in the overall performance attained by NSCDB.
- **Chapter 6** presents our conclusions and main findings as well as possible future research directions on alternative approaches to improve exploration and exploitation on OLTR for the sequential music recommendation task, eventually applied on different recommendation scenarios.

Chapter 2

Related Work

In this section, we review several related studies covering the three main topics of this work: automatic playlist generation, online learning to rank (OLTR), and efficient exploration and exploitation.

2.1 Automatic Playlist Generation

The task of generating a playlist of songs can be performed manually or in an automated fashion [Dias et al., 2017]. While manual playlists built by a streaming service community can engage thousands of users [Pichl et al., 2017], their construction can sometimes be laborious and biased towards the creators' tastes [Bonnin and Jannach, 2014]. Moreover, playlist creators have to manually select from an enormous catalog of songs, which can be suboptimal [Oulasvirta et al., 2009]. Automatic playlist generation aims to address the aforementioned problems by automating the selection of songs through different methods and algorithms [Bonnin and Jannach, 2014]. Automatic playlist generation approaches can be stratified into two main branches: static and dynamic. On the one hand, static methods usually leverage historical user data to generate a fixed offline personalized playlist [Hauver and French, 2001; Logan, 2002; Baccigalupo and Plaza, 2006; Gartner et al., 2007; Chen et al., 2012; Bonnin and Jannach, 2014; Pelle, 2017]. On the other hand, besides using historical user data, dynamic approaches also leverage user's explicit (ratings, likes, etc.) or implicit feedback (plays or skips) to adaptively recommend new songs to an existing playlist during an online listening session [Xing et al., 2014; Wang et al., 2014; Hariri et al., 2015; Liebman et al., 2015; Pampalk et al., 2005; Bosteels and Kerre, 2009; King and Imbrasaitė, 2015; Hu et al., 2017].

2.1.1 Static Playlist Generation

The first proposed static playlist generation methods were based on classical recommendation techniques, such as collaborative [Hauver and French, 2001] and content-based [Logan, 2002; Gartner et al., 2007] filtering. A different branch of methods was proposed to generate recommendations by relying on a seed song as a starting point. For instance, Baccigalupo and Plaza [2006] proposed a case-based approach to learn from a set of playlists, generating new ones from the starting song. Chen et al. [2012] proposed addressing the same problem using Markov models, where transition probabilities were estimated based on the Euclidean distance between songs. This method was extended by Moore et al. [2012] adding songs' tag information.

Although successful, these methods tackle playlist generation in a static fashion, leveraging historical user preferences to recommend a fixed list of songs. In contrast, the method proposed in this dissertation focuses on targetting the sequential music recommendation task, in which songs are dynamically recommended based upon the user's context and feedback throughout the session [Wang et al., 2021].

2.1.2 Dynamic Playlist Generation

Different from static approaches, dynamic playlist generation algorithms target the recommendation task in an interactive way. By requiring the user's feedback over one or more recommendations at each interaction, dynamic strategies continuously adapt the recommender model towards the user's preferences in a listening session. Dynamic playlist generation methods commonly rely on a reinforcement learning framework to adaptively recommend new songs based on user feedback.

Reinforcement learning (RL) [Sutton and Barto, 2018] is a subarea of machine learning (ML) [Murphy, 2012] that studies computational methods to learn from interactions with an environment. As illustrated in Figure 2.1, a RL problem can be understood as an agent continuously interacting with some environment by performing actions and receiving reward signals over those actions, altering the state of the environment every time a new action is performed. The goal of this agent is to maximize its cumulative reward over time.

RL can be considered as a third learning paradigm in ML [Sutton and Barto, 2018], being different from the two other existing paradigms: supervised and unsupervised learning. Different from supervised learning where we are trying to learn a mapping function from inputs to outputs given a labeled set of input-output pairs [Murphy, 2012], RL does not require labeled examples to perform the learning process, learning directly from the immediate rewards. RL is also different from unsupervised

learning as its goal is to maximize reward instead of uncovering possible hidden patterns in collections of unlabeled data.

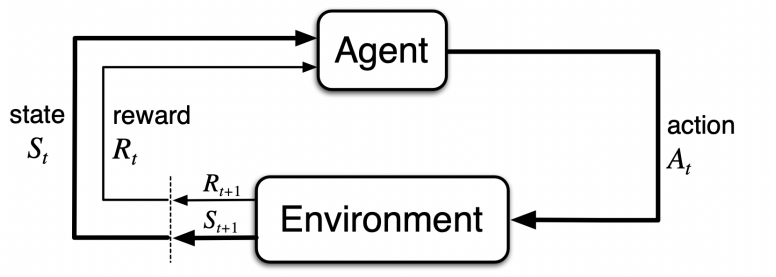


Figure 2.1. Reinforcement learning problem illustration: an agent continuously learning from the environment. Illustrative diagram from Sutton and Barto [2018].

RL-based algorithms can be employed to continuously adapt an online recommender model, being a flexible alternative to improve the song recommendation task by leveraging user's feedback. Wang et al. [2014] proposed a RL approach focused on improving the greedy nature of the classic collaborative filtering approaches (i.e. only recommending items with high associated ratings), by balancing exploitation with exploration, also modeling songs' novelty with respect to the user listening history. They use a Bayesian graphical model to recommend new songs, estimating the posterior distribution of the expected rating of a song given multiple parameters such as the user's preferences for different collaborative filtering latent factors, the relative strength of the user's memory (associated with the novelty of the songs), and others. After each recommendation, they collect the user's explicit rating over the recommended song and then update the posterior distribution of the parameters.

Multi-armed bandits (MAB) is a problem studied in probability theory that can be understood as a simplified instance of a RL problem where actions do not affect future environment states. One instance of the MAB problem is the k-armed bandit [Sutton and Barto, 2018] problem where a player (agent) must choose which lever to pull (action) from a set of k gambling machines. The goal of this player is to maximize the reward received from the k gambling machines over time.

Hariri et al. [2015] presented a MAB strategy to model the interactive music recommendation task that includes a component capable of adapting the recommendation based on changes in users' interests. They employ an adapted version of Thompson sampling [Chapelle and Li, 2011] to recommend new songs while future recommendations are continuously adapted at each interaction by measuring differences in the posterior distributions of ratings and song features on different time windows. If a computed difference is above a certain threshold, new recommendations are generated

only considering the most recent time window. Otherwise, the full listening history is considered.

Although successful, the aforementioned methods require the user's explicit feedback (e.g. ratings), imposing an intrusive interaction mechanism that can harm user experience [Jawaheer et al., 2010]. Alternatively, implicit feedback (e.g. plays or skips) can be used to guide the offline playlist generation, however it is still underexplored in the online sequential recommendation domain [Vall, 2015]. Pampalk et al. [2005] proposed the first methods that leveraged implicit feedback to add songs to an existing playlist by evaluating the similarity of audio features. Fuzzy set theory was later used to formalize the dynamic song recommendation task through different heuristics [Bosteels and Kerre, 2009]. Although successful for this particular scenario, these methods offer no flexibility in terms of personalized user adaptation, which means that they apply the same predefined rules on every model update. The contributions by King and Imbrasaitė [2015] and Hu et al. [2017] are notable exceptions that used RL algorithms based on implicit feedback to address personalized recommendation, however with a focus on song clusters rather than individual songs.

2.2 Online Learning to Rank (OLTR)

Contextual multi-armed bandit (CMAB) or contextual bandit problem is a particular variant of the k-armed bandit problem where an agent is capable of observing a feature vector (context) associated with each one of the k gambling machines to decide which lever she/he should pull [Bouneffouf et al., 2020]. Online learning to rank (OLTR) can be formalized as an instance of a contextual bandit problem [Grotov and de Rijke, 2016] where a recommender/model (agent) continuously interacts with the user (environment) by recommending items (actions) based on the user and a query (context), receiving feedback (reward) over those recommended items [Hofmann et al., 2013a]. The goal of this recommender is to maximize the cumulative reward (clicks, plays, etc.) from the user over time. Figure 2.2 illustrates the OLTR problem for the document ranking task.

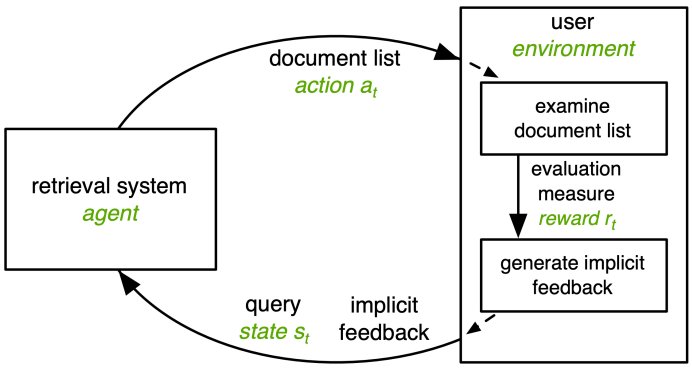


Figure 2.2. Online Learning to Rank (OLTR) problem for the document ranking task. Illustrative diagram from Hofmann et al. [2013a]

Similar to dynamic playlist generation approaches that employ RL to address the recommendation task, the OLTR framework can also be used to adaptively recommend new items based on the user's context and implicit feedbacks throughout the listening session. Here we describe and explain the canonical algorithm used in OLTR approaches, also outlining different variants of its structure targetting document ranking and sequential music recommendation tasks.

2.2.1 Canonical OLTR Algorithm

Several different OLTR approaches were proposed to tackle the document ranking task [Yue and Joachims, 2009; Schuth et al., 2016; Zhao and King, 2016; Wang et al., 2018]. Despite having their own particularities, they all share a common canonical algorithmic structure, outlined by Algorithm 1.

OLTR(u, δ, α)
1 $\mathbf{w}_0 \leftarrow$ *initializeModel*()
2 **for** $t \leftarrow 1$ **to** ∞ **do**
3 $\quad \mathbf{S} \leftarrow$ *retrieveCandidateItems*(u)
4 $\quad \mathbf{L} \leftarrow$ *exploreCandidateDirections*$(\mathbf{w}_{t-1}, \mathbf{S}, \delta)$
5 $\quad \mathbf{R} \leftarrow$ *estimateCandidateDirectionsReward*$(\mathbf{L})$
6 $\quad \mathbf{w}_t \leftarrow$ *exploitCandidateDirections*$(\mathbf{R}, \alpha)$
7 **end for**

Algorithm 1. Canonical OLTR algorithm.

The canonical algorithm receives as input a user u, an exploration parameter δ and an exploitation parameter α. Online learning interactions are performed in discrete time steps t, assuming a user session of inifinite time horizon. It is divided

into 5 main steps: (i) recommender model initialization, represented by the *initializeModel* function, (ii) retrieval of candidate items to be recommended, represented by the *retrieveCandidateItems* function, (iii) exploration of candidate model update directions, represented by the *exploreCandidateDirections*, (iv) estimation of candidate model update directions rewards, represented by the *estimateCandidateDirectionsReward* function, and (v) exploitation of candidate model update directions, represented by the *exploitCandidateDirections* function. Different OLTR approaches implement their own version of the previous outlined functions, possibly adding more parameters.

The *initializeModel* function is responsible for initializing the weights of a d-dimensional linear model employed as the recommender model. This model will be used to score items, represented in the same d-dimensional space as feature vectors, and will be updated during the online recommendation session based on the user's feedback (clicks, plays, etc.). Although multiple linear model initialization strategies have been proposed [Yam and Chow, 2000], almost all OLTR algorithms outlined in this section employ a uniform random initialization approach.

At every iteration, an OLTR algorithm must retrieve candidate items, such as documents or songs, to be recommended. As stated previously, candidate items are represented as d-dimensional feature vectors. The *retrieveCandidateItems* function is responsible for retrieving a set $\mathbf{S}$ of possible candidate items based on the user's previous interactions. Several retrieval functions have been proposed to retrieve documents for a given user query [Ghorab et al., 2013] and to retrieve candidate songs to be recommended during a listening session [Cheng et al., 2016], but their explanation and analysis is beyond the scope of this work.

Once candidate items are retrieved, OLTR algorithms explore possible candidate model update directions in the hope of improving the current model. The *exploreCandidateDirections* function is responsible for generating one or more candidate exploration directions that are then used to build candidate recommender models. A candidate recommender model is constructed by performing a vector operation between the current model $\mathbf{w}_{t-1}$ and a candidate direction $\mathbf{v}$, also taking into account the exploration parameter δ. Those candidate recommender models are employed to generate different possible rankings by scoring the retrieved items $\mathbf{S}$ via inner product (ordered from highest to lowest score). Candidate models and directions plus the respective produced rankings are stored in a set $\mathbf{L}$.

Candidate rankings must be directly or indirectly evaluated by the user. The *estimateCandidateDirectionsReward* function implements the logic behind this evaluation, returning a set $\mathbf{R}$ that holds, for each candidate ranking, its respective reward, and the candidate direction used to produce it. For the direct evaluation, OLTR

algorithms that target the document ranking task normally employ an interleaving [Radlinski et al., 2008; Zhao and King, 2016] strategy to combine multiple candidate rankings into a single unified ranking, which is then presented to the user to gather click information, used as a reward measure. In contrast, indirect ranking evaluation is needed for the sequential music recommendation task. Since only one song can be recommended at each iteration, interleaving ranking evaluation becomes infeasible. As a solution, counterfactual methods have been proposed to estimate candidate rankings' rewards as an alternative to interleaving for the sequential music recommendation task [Pereira et al., 2019], being further detailed in Section 2.2.4.

Once all candidate rankings are evaluated, an OLTR algorithm will hold reward values for candidate directions that generated each one of the candidate rankings. The *exploitCandidateDirections* function is responsible for updating the current model at a given iteration t by performing a vector operation between the current model $\mathbf{w}_{t-1}$ and a candidate direction $\mathbf{v}$, also taking into account the respective reward and the exploitation parameter α. All aforementioned steps from (ii) to (v) are continuously repeated during the user's session, trying to improve the recommender model at each interaction with the user.

All OLTR based algorithms share a common theoretical justification that guarantees their convergence towards an approximation of the best solution (i.e. that maximizes the reward over time). Flaxman et al. [2005] work prove that the online convex optimization performed in the bandit setting approximates the gradient descent (GD) optimization method [Spall, 2005] with a known regret bound. GD is a classic optimization algorithm that iteratively searches the gradient of a function, where $J(\Theta)$ defines an objective function with parameters $\Theta \in \mathbb{R}$ and the gradient calculation $\Delta_\Theta J(\Theta)$. The algorithm calculates the direction of greatest slope that minimizes the result $J(\Theta_t)$ of the objective function at time t and updates the parameter values for the next iteration Θ_{t+1}. Hence, the algorithm explores new combinations for the parameters until it reaches a convergence point where the difference between the results is less than a stopping criterion ϵ (i.e. $J(\Theta_{t+1}) - J(\Theta_t) < \epsilon$). Due to this stopping condition, the algorithm is not guaranteed to find the optimal solution [Spall, 2005].

The key idea behind Flaxman et al. [2005] work is that the gradient of the unknown objective function being optimized during the OLTR algorithm execution can be approximated by evaluating the function at random points in time. That is, it is possible to adjust the recommender model weights with candidate directions using feedback information and this is guaranteed to converge to a suboptimal solution.

In the rest of this section, we review several different OLTR approaches proposed to address the document ranking and sequential music recommendation tasks, explain-

ing their algorithms using the outlined canonical OLTR algorithm framework. Each OLTR approach implements its own version of the 5 functions previously explained.

2.2.2 Dueling Bandits Gradient Descent (DBGD)

Dueling Bandits Gradient Descent (DBGD) [Yue and Joachims, 2009] was the first OLTR algorithm proposed, targetting the document ranking task. DBGD's goal is to dynamically learn a document ranking model based on the user's feedback (clicks) over the documents retrieved for a given query. Algorithm 2 presents DBGD logic using the canonical OLTR algorithm adapted functions, outlined by Algorithms 3 to 6.

DBGD(u, δ, α)
1 $\mathbf{w}_0 \leftarrow$ *initializeModelRandom*()
2 $m \leftarrow 1$ # explore only one candidate direction
3 **for** $t \leftarrow 1$ **to** ∞ **do**
4 $q \leftarrow$ Receive user query
5 $\mathbf{S} \leftarrow$ *retrieveCandidateItemsDBGD*(u, q)
6 $\mathbf{L} \leftarrow$ *exploreCandidateDirectionsRandom*$(\mathbf{w}_{t-1}, \mathbf{S}, m, \delta)$
7 $\mathbf{R} \leftarrow$ *estimateCandidateDirectionsRewardInterleave*$(\mathbf{L})$
8 $\mathbf{w}_t \leftarrow$ *exploitCandidateDirectionsSingleRandomWinner*$(\mathbf{R}, \mathbf{w}_{t-1}, \alpha)$
9 **end for**

Algorithm 2. Dueling Bandit Gradient Descent (DBGD) algorithm [Yue and Joachims, 2009].

The model initialization phase is performed by the *initializeModelRandom* function (Algorithm 3) that randomly initializes the weights of the initial d-dimensional linear model $\mathbf{w}_0$. The number m of candidate directions to explore is set to be 1 (line 2 of Algorithm 4), meaning that DBGD will only explore a single candidate direction. Candidate documents to be recommended, represented by d-dimensional feature vectors, are retrieved by the *retrieveCandidateItemsDBGD* function based on the user's history and current query q at a given iteration t. As stated previously, the details of this retrieval function are beyond the scope of this work.

initializeModelRandom()
1 $\mathbf{w} \leftarrow$ *random*()
2 **return** $\mathbf{w}$

Algorithm 3. *initializeModelRandom* function logic.

Candidate directions exploration phase is done by the *exploreCandidateDirectionsRandom* function (Algorithm 4). It creates $m + 1$ candidate directions $\mathbf{v}_i$, being the

first one a zeroed feature vector (line 4 of Algorithm 4) and, the other m, random uniform feature vectors (line 6 of Algorithm 4). Each candidate direction is used to build a candidate recommender $\mathbf{w}_i$ (line 8 of Algorithm 4) by adding $\delta\mathbf{v}_i$ to $\mathbf{w}_{t-1}$. Please note that $\mathbf{w}_0$ will be equal to $\mathbf{w}_{t-1}$ since $\mathbf{v}_0$ is a zeroed feature vector. Then, each $\mathbf{w}_i$ is used to score each document in $\mathbf{S}$ via inner product $\langle \mathbf{w}_i, \mathbf{S} \rangle$, producing a candidate ranking $\mathbf{l}_i$ (line 9 of Algorithm 4). Finally, all respective $\mathbf{v}_i$, $\mathbf{w}_i$, and $\mathbf{l}_i$ are stored in a set $\mathbf{L}$ (line 10 of Algorithm 4), returned by the function. Since DBGD only explores a single candidate direction ($m = 1$), $\mathbf{L}$ will contain only two elements: one holding a candidate ranking $\mathbf{l}_0$ generated by $\mathbf{w}_0$ (equal to $\mathbf{w}_{t-1}$) and another candidate ranking $\mathbf{l}_1$ generated by $\mathbf{w}_1$, built using the single exploration direction $\mathbf{v}_1$.

exploreCandidateDirectionsRandom$(\mathbf{w}_{t-1}, \mathbf{S}, m, \delta)$

```
1  L ← {}
2  for i ← 0...m do
3    if i == 0 then
4      v_i ← {0, ..., 0}  # first candidate direction is an empty vector so that l_0 is generated by
       the current model w_{t-1}
5    else
6      v_i ← random()                          # other candidate directions are random
7    end if
8    w_i ← w_{t-1} + δv_i                      # candidate direction exploration
9    l_i ← ⟨w_i, S⟩                            # candidate ranking generation
10   L ← L ∪ {v_i, l_i, w_i}
11 end for
12 return L
```

Algorithm 4. *exploreCandidateDirectionsRandom* function pseudocode.

The two rankings in $\mathbf{L}$ will be evaluated using function *estimateCandidateDirectionsRewardInterleave* (Algorithm 5). An interleaving technique [Radlinski et al., 2008] is used to unify both rankings into a single ranking $\mathbf{C}$ (line 1 of Algorithm 5). Next, the unified ranking $\mathbf{C}$ is presented to the user and the number of clicks c_i (reward) for the documents of each ranking $\mathbf{l}_i$ on $\mathbf{C}$ are recorded in a set $\mathbf{R}$, alongside the respective model $\mathbf{w}_i$ that produced $\mathbf{l}_i$ and the direction $\mathbf{v}_i$ that was used to build $\mathbf{w}_i$ (lines 2 and 3 of Algorithm 5).

***estimateCandidateDirectionsRewardInterleave*(L)**
1 **C** ← Interleave or Multileave all candidate rankings $\mathbf{l}_i$ in **L**
2 Show **C** to the user and compute the received number of clicks c_i for each candidate direction $\mathbf{v}_i$ that produced the corresponding ranking l_i on **L**
3 $\mathbf{R} \leftarrow \{\{\mathbf{v}_i, \mathbf{w}_i, \mathbf{c}_i\}\} \; \forall \; \{\mathbf{v}_i, \mathbf{w}_i\} \in \mathbf{L}$
4 **return R**

Algorithm 5. *estimateCandidateDirectionsRewardInterleave* function pseudocode.

After the number of clicks for each ranking has been recorded, DBGD exploits the candidate directions that indirectly produced those rankings using the *exploitCandidateDirectionsSingleRandomWinner* function (Algorithm 6). It first selects all $\mathbf{w}_i$ and the respective $\mathbf{v}_i$ from **R** that had the highest associated number of clicks c_i, storing the results in a set **W** (line 1 of Algorithm 6). If $\mathbf{w}_{t-1}$ is present in **W**, no candidate model $\mathbf{w}_i$ was able to generate a ranking that received a number of clicks higher than the ranking generated by the current model $\mathbf{w}_{t-1}$. This indicates that the current model should be kept, thus not being updated (line 3 of Algorithm 6). Conversely, if $\mathbf{w}_{t-1}$ is not present in **W**, it is an indicative DBGD was able to find a candidate model built using an exploration direction $\mathbf{v}_i$ that produced a ranking that got more clicks than the one generated by the current model. If DBGD had explored more than a single candidate direction, the set **W** could have contained multiple elements, requiring a tie-breaking step (in this case a random one, on line 5 of Algorithm 6) to select a single winning candidate direction **v**. As DBGD explores only a single candidate direction, the tie-breaking will always select the only candidate direction **v** on **W**. Finally, if a candidate direction **v** is selected, it will be used to update the current model by adding $\alpha\mathbf{v}$ to $\mathbf{w}_{t-1}$ (line 6 of Algorithm 6), producing an updated model $\mathbf{w}_t$ for the next interaction with the user.

***exploitCandidateDirectionsSingleRandomWinner*(R, $\mathbf{w}_{t-1}$, α)**
1 **W** ← Select all $\mathbf{w}_i$ and the respective $\mathbf{v}_i$ in **R** that had the highest associated number of clicks c_i
2 **if** $\mathbf{w}_{t-1} \in \mathbf{W}$ **then**
3 $\quad \mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$ # keep current model
4 **else**
5 $\quad \mathbf{v} \leftarrow$ Select random $\mathbf{v}_i \in \mathbf{W}$ # random tie-breaking
6 $\quad \mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + \alpha\mathbf{v}$ # update current model
7 **end if**
8 **return** $\mathbf{w}_t$

Algorithm 6. *exploitCandidateDirectionsSingleRandomWinner* function pseudocode.

As outlined in Section 2.2.1 for the canonical OLTR algorithm, the previously mentioned functions (*retrieveCandidateItemsDBGD*, *exploreCandidateDirectionsRan-*

dom, *estimateCandidateDirectionsRewardInterleave*, and *exploitCandidateDirectionsSingleRandomWinner*) are repeated on every interaction with the user in the hope of improving the recommender model, thus generating recommendations tailored to the user's interests.

2.2.3 Multileave Gradient Descent (MGD)

Multileave Gradient Descent (MGD) [Schuth et al., 2016] was later proposed as an extension of DBGD to improve the document recommendation task where multiple candidate ranking models are jointly evaluated at each iteration via multileaving [Schuth et al., 2014] instead of only two on DBGD via interleaving, increasing the probability of finding better candidate rankers.

Algorithm 7 presents MGD pseudocode, which is exactly the same as DBGD's algorithm, but (i) with the number of candidate directions m not being fixed to 1 and (ii) a specific *retrieveCandidateItemsMGD* document retrieval function. Since multiple candidate directions are evaluated, the set $\mathbf{W}$ on *exploitCandidateDirectionsSingleRandomWinner* function can contain more than two winning directions. In this case, random tie-breaking is necessary to select a single winning direction. Schuth et al. [2016] also presented another strategy to exploit multiple winning candidate directions by updating the current model towards the average vector of the multiple directions.

MGD $(u, \delta, \alpha,$ **,** $m)$
1 $\mathbf{w}_0 \leftarrow$ *initializeModelRandom*()
2 **for** $t \leftarrow 1$ **to** ∞ **do**
3 $q \leftarrow$ Receive user query
4 $\mathbf{S} \leftarrow$ *retrieveCandidateItemsMGD*(u, q)
5 $\mathbf{L} \leftarrow$ *exploreCandidateDirectionsRandom*$(\mathbf{w}_{t-1}, \mathbf{S}, m, \delta)$
6 $\mathbf{R} \leftarrow$ *estimateCandidateDirectionsRewardInterleave*$(\mathbf{L})$
7 $\mathbf{w}_t \leftarrow$ *exploitCandidateDirectionsSingleRandomWinner*$(\mathbf{R}, \mathbf{w}_{t-1}, \alpha)$
8 **end for**

Algorithm 7. Multileave Gradient Descent (MGD) algorithm [Schuth et al., 2016].

By evaluating multiple candidate models at each iteration MGD is able to achieve better performance results compared to DBGD, also requiring a significantly reduced number of interactions with the user, improving learning speed and thus reducing the user's exposure to unsuitable recommendations, but trading off computational cost to generate and evaluate multiple candidate directions.

2.2.4 Counterfactual Dueling Bandits (CDB)

Since DBGD and MGD target the document ranking task, both methods were conceived to address a ranking scenario where the user gives feedback over a list of items. However, sequential music recommendation suffers from scarcity of feedback, since only one song can be presented at a time, which renders a direct application of DBGD and MGD infeasible. Counterfactual Dueling Bandits (CDB) [Pereira et al., 2019] introduced a novel OLTR method by estimating the quality of candidate rankers in a counterfactual fashion, based upon a single implicit feedback signal from the user at each interaction. This allows CDB to perform multiple duels between models as in MGD, but without exposing the user to the explored candidates. To the best of our knowledge, CDB was the first proposed OLTR algorithm to address the sequential music recommendation task, being one of the core inspirations for the new OLTR algorithm proposed in this work.

Algorithm 8 presents the pseudocode of CDB using adapted functions from the canonical OLTR algorithm. CDB introduces a new parameter $\mathbf{s}_0$ representing the initial playlist seed song to start the listening session. At each iteration t, CDB recommends a new song $\mathbf{s}_t$ from a pool of retrieved songs $\mathbf{S}$ (also based in the user history and current session, being represented as d-dimensional feature vectors) by selecting the one with the highest associated score from the current model $\mathbf{w}_{t-1}$ (line 7 of Algorithm 8). The user u then provides her or his implicit feedback p_t over the recommended song. A positive feedback ($p_t = 1$) indicates that the user listened to more than half of the duration of the song (play). Conversely, a negative feedback ($p_t = 0$) indicates the opposite (skip). CDB triggers the model update phase only after a negative feedback, otherwise it keeps the current model unaltered. As in the previously outlined OLTR algorithms, the model update phase consists of an initial exploration of candidate directions step, followed by a reward estimation and a final exploitation step of the generated candidate directions. The functions that implement the previous steps are very similar to the ones used in DBGD and MGD, however with a few key modifications to address the sequential music recommendation task.

CDB$(u, \mathbf{s}_0, m, \delta, \alpha)$
$\mathbf{w}_0 \leftarrow$ *initializeModelRandom*()
for $t \leftarrow 1$ **to** ∞ **do**
  $\mathbf{S} \leftarrow$ *retrieveCandidateItemsCDB*$(u, \mathbf{s}_0, ..., \mathbf{s}_{t-1})$
  $\mathbf{s}_t \leftarrow \arg\max_{\mathbf{s} \in \mathbf{S}} \langle \mathbf{w}_{t-1}, \mathbf{s} \rangle$ # recommend next song
  $p_t \leftarrow$ *payoff*$(\mathbf{s}_t)$ # receive user feebdack
  if $p_t > 0$ **then**
    $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$ # carry current model over to next time step
    $r \leftarrow 1$ # reset current model reward to maximum
  else
    $r \leftarrow 1/(1/r + 1)$ # penalize estimated model reward
    $\mathbf{L} \leftarrow$ *exploreCandidateDirectionsRandom*$(\mathbf{w}_{t-1}, \mathbf{S}, m, \delta)$
    $\mathbf{R} \leftarrow$ *estimateCandidateDirectionsRewardCounterfactual*$(\mathbf{L}, \mathbf{s}_t)$
    $\mathbf{w}_t \leftarrow$ *exploitCandidateDirectionsCDB*$(\mathbf{R}, r, \alpha)$
  end if
end for

Algorithm 8. The CDB algorithm [Pereira et al., 2019]. Adapted to be in the OLTR canonical algorithm format.

The candidate directions exploration step is the same one performed in DBGD and MGD, via *exploreCandidateDirectionsRandom*. However, the reward estimation of the generated candidate directions cannot be performed, for example, via multileaving as in MGD, as only one song can be presented to the user, as opposed to multiple documents in a single unified ranking for the document ranking task. To address this issue, Pereira et al. [2019] proposed a counterfactual approach to estimate the reward of candidate directions for the sequential music recommendation task, implemented by the *estimateCandidateDirectionsRewardCounterfactual* function (Algorithm 9). Since the model update phase happens after a negative feedback, the estimated reward r_c of a candidate direction $\mathbf{v}_i$ is computed as the reciprocal rank of a skipped song $\mathbf{s}_t$ in the ranking $\mathbf{l}_i$ produced by a model $\mathbf{w}_i$ built using $\mathbf{v}_i$. That is, the reward of a candidate direction is inversely proportional to the position of a skipped song in a ranking produced by a candidate model built using this candidate direction if the user was allowed to see this ranking. CDB presents an inversed reward estimation rationale. Higher r_c values indicate that the skipped song would have been placed at the top of the candidate ranking. Conversely, lower r_c values indicate that the skipped song would have been placed at the bottom of the candidate ranking.

estimateCandidateDirectionsRewardCounterfactual($\mathbf{L}$, $\mathbf{s}_t$)
1 $\mathbf{R} \leftarrow \{\}$
2 **for all** $\{\mathbf{v}_i, \mathbf{l}_i, \mathbf{w}_i\} \in \mathbf{L}$ **do**
3 $\quad r_c \leftarrow 1/rank(\mathbf{s}_t, \mathbf{l}_i)$
4 $\quad \mathbf{R} \leftarrow \mathbf{R} \cup \{\mathbf{v}_i, r_c\}$
5 **end for**
6 **return** $\mathbf{R}$

Algorithm 9. *estimateCandidateDirectionsRewardCounterfactual* function pseudocode.

CDB's candidate directions exploitation phase is also different from the one performed on DBGD and MGD, being implemented by the *exploitCandidateDirectionsCDB* function (Algorithm 10). Instead of updating the model towards candidate directions that received a reward greater than the current model (line 6 of Algorithm 6), CDB applies a more conservative approach by updating the current model towards the opposite of the underperforming candidate directions (vector difference on line 4 of Algorithm 10) that received a reward r_c higher than the current model reward r. It is worth noting that, differently from candidate models rewards r_c, the current model reward r serves as a threshold value to allow the model update. By applying a decay penalization policy over r when a negative feedback is received (line 10 of Algorithm 8), CDB increases the probability of updating the current underperforming model (i.e. a model that is recommending songs being skipped by the user). Although not explicitly mentioned, r can have an initial random value for $\mathbf{w}_0$, as it is reset to 1 every time a user gives a positive feedback.

exploitCandidateDirectionsCDB($\mathbf{R}$, r, α)
1 $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$
2 **for all** $\{\mathbf{v}_i, r_c\} \in \mathbf{R}$ **do**
3 $\quad$ **if** $r_c > r$ **then**
4 $\quad\quad \mathbf{w}_t \leftarrow \mathbf{w}_t - \alpha r_c \mathbf{v}_i$
5 $\quad$ **end if**
6 **end for**
7 **return** $\mathbf{w}_t$

Algorithm 10. *estimateCandidateDirectionsRewardCounterfactual* function pseudocode.

CDB's algorithm is used as the basis for the OLTR approach proposed in this dissertation, which also targets the sequential music recommendation task. However, just as in DBGD and MGD, CDB presents possible inefficiencies in the exploration and exploitation phases because of the random exploration component and the fixed exploitation strategy of the current best model, respectively. The next section details

these inefficiencies, reviewing proposed solutions and linking the core ideas with the approach proposed in this dissertation.

2.3 Efficient Exploration and Exploitation

OLTR algorithms such as DBGD, MGD, and CDB rely on a random exploration component to generate candidate recommender models during online interactions. However, by randomly exploring the space of candidate models, these methods fail to avoid already known poor directions [Wang et al., 2018], which can harm learning speed and convergence, exposing users to unsuitable recommendations. At the same time, these approaches incorporate explored directions uniformly, regardless of the effectiveness of the model learned thus far, which in turn may lead to inefficient exploitation.

Different approaches have been proposed to address the issues of random exploration and uniform exploitation on OLTR algorithms. The rest of this chapter outlines those approaches, that were also a source of inspiration for our proposed OLTR algorithm.

2.3.1 Improving Exploration

Different approaches have been suggested to address the learning speed and convergence problems of OLTR algorithms, therefore promoting efficient exploration. Hofmann et al. [2013b] reused historical user interactions to (i) improve interleaved tests when comparing different rankers and (ii) perform a preselection of candidate rankers. By evaluating candidate rankers on previous data they aim to filter out rankers known to perform poorly. Oosterhuis and de Rijke [2017] followed a similar idea, preselecting a set of reference documents to improve learning speed by evaluating the performance of a candidate ranker on this set.

Although successful, the previously mentioned algorithms still apply a random exploration strategy when generating candidate update directions. Moreover, randomly exploring the space of possible candidate directions leads to candidate models that are independent of past interactions with the user, offering no means to avoid previously known underperforming directions.

2.3.2 Null Space Gradient Descent (NSGD)

To address the random exploration issues present on previously outlined OLTR algorithms, Wang et al. [2018] suggested a new exploration approach for the document rank-

ing task, named Null Space Gradient Descent (NSGD). In NSGD, candidate rankers are generated based on the null (orthogonal) space of recently explored gradients (directions) that led to poorly performing models. By restricting the exploration to this null space, they avoid repeating known directions of inferior performance, increasing learning speed and overall convergence of the learned models when compared to the random exploration strategy performed by DBGD and MGD. Algorithm 11 presents NSGD pseudocode in the OLTR canonical algorithmic format.

NSGD(u, δ, α, m, n, k, h)
1 $\mathbf{w}_0 \leftarrow$ *initializeModelRandom*()
2 $\mathbf{Q} \leftarrow$ Initialize empty queue of size h
3 **for** $t \leftarrow 1$ **to** ∞ **do**
4 $q \leftarrow$ Receive user query
5 $\mathbf{S} \leftarrow$ *retrieveCandidateItemsNSGD*(u, q)
6 $\mathbf{L} \leftarrow$ *exploreCandidateDirectionsNSGD*($\mathbf{w}_{t-1}, \mathbf{S}, m, \delta, \mathbf{Q}, k, n$)
7 $\mathbf{R} \leftarrow$ *estimateCandidateDirectionsRewardInterleave*($\mathbf{L}$)
8 $\mathbf{w}_t \leftarrow$ *exploitCandidateDirectionsNSGD*($\mathbf{R}, \alpha, \mathbf{Q}$)
9 **end for**

Algorithm 11. Null Space Gradient Descent (NSGD) algorithm [Wang et al., 2018]. Adapted to be in the OLTR canonical algorithm format.

NSGD works in a very similar way when compared to DBGD and MGD, but with key modifications on both exploration and exploitation phases that promote superior performance. NSGD introduces a queue $\mathbf{Q}$ (initialized on line 2 of Algorithm 11) holding h previous underperforming update directions, being used to guide the efficient exploration. *exploreCandidateDirectionsNSGD* function (Algorithm 12) implements the exploration phase of NSGD. The top-k worst directions present in $\mathbf{Q}$ are selected as input to the null space computation, producing a set $\mathbf{N}$ with update directions that are orthogonal to all k worst directions (line 2 of Algorithm 12), serving as a safer reduced subspace of possible update directions. NSGD randomly samples n update directions from $\mathbf{N}$, which are further filtered to ensure that they are discriminative when compared to the current best model. Precisely, NSGD keeps in a set $\mathbf{V}$ the m directions with the highest inner product with respect to the average document feature vector computed from $\mathbf{S}$ (line 4 of Algorithm 12), increasing the probability of generating candidate models that would produce rankings different from the one produced by the current model [Wang et al., 2018]. The rest of *exploreCandidateDirectionsNSGD* function works in the exact same way as *exploreCandidateDirectionsRandom* function (Algorithm 4), however selecting candidate update directions from the $\mathbf{V}$ set.

exploreCandidateDirectionsNSGD$(\mathbf{w}_{t-1}, \mathbf{S}, m, \delta, \mathbf{Q}, k, n)$

1 $\mathbf{L} \leftarrow \{\}$
2 $\mathbf{N} \leftarrow null(worst(\mathbf{Q}, k))$ # get null space of k worst directions
3 $\mathbf{B} \leftarrow sample(\mathbf{N}, n)$ # sample n null space directions
4 $\mathbf{V} \leftarrow filter(\mathbf{B}, m, \mathbf{S})$ # keep m directions with highest coverage of $\mathbf{S}$
5 **for** $i \leftarrow 0...m$ **do**
6 **if** $i == 0$ **then**
7 $\mathbf{v}_i \leftarrow \{0, ..., 0\}$ # first candidate direction is an empty vector so that $\mathbf{l}_0$ is generated by the current model w_{t-1}
8 **else**
9 $\mathbf{v}_i \leftarrow$ Get $\mathbf{v}_i$ from the i-th position of $\mathbf{V}$
10 **end if**
11 $\mathbf{w}_i \leftarrow \mathbf{w}_{t-1} + \delta\mathbf{v}_i$ # candidate direction exploration
12 $\mathbf{l}_i \leftarrow \langle \mathbf{w}_i, \mathbf{S} \rangle$ # candidate ranking generation
13 $\mathbf{L} \leftarrow \mathbf{L} \cup \{\mathbf{v}_i, \mathbf{l}_i, \mathbf{w}_i\}$
14 **end for**
15 **return** $\mathbf{L}$

Algorithm 12. *exploreCandidateDirectionsNSGD* function pseudocode.

The candidate directions reward estimation phase is exactly the same as the one performed on MGD, via multileaving (Algorithm 5). Accordingly, the exploitation phase is also very similar to the one performed on DBGD and MGD, but with two major differences, outlined by the *exploitCandidateDirectionsNSGD* (Algorithm 13). The first one is regarding how NSGD performs tie-breaking when multiple winning candidate directions receive the same number of clicks. Instead of randomly picking one out of the winner set $\mathbf{W}$, NSGD evaluates candidate models built with the candidate directions on past difficult queries in an offline fashion, selecting the one that generated a ranking with the highest number of clicks (line 5 of Algorithm 13). This advanced tie-breaking approach is experimentally proven to improve performance [Wang et al., 2018]. The second difference regards the queue update step performed on lines 9 to 14 of Algorithm 13. To ensure that the null space computation is being performed over an updated list of recent underperforming directions, NSGD updates the queue on a First-In-First-Out (FIFO) fashion based in the comparison of the received number of clicks between the candidate direction and the current model. If the candidate direction $\mathbf{v}_i$ produced a candidate model that generated a ranking that received a number of clicks lower than the ranking generated by the current model, it is an indicator that $\mathbf{v}_i$ is underperforming in comparison with the current model and should be avoided, being pushed into the queue $\mathbf{Q}$.

exploitCandidateDirectionsNSGD$(\mathbf{R}, \alpha, \mathbf{Q})$

$\mathbf{W} \leftarrow$ Select all $\mathbf{w}_i$ and the respective $\mathbf{v}_i$ in $\mathbf{R}$ that had the highest associated number of clicks c_i
if $\mathbf{w}_{t-1} \in \mathbf{W}$ **then**
  $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$ # keep current model
else
  $\mathbf{v} \leftarrow$ Select the winner $\mathbf{v}_i$ used to build $\mathbf{w}_i$ that produced a ranking $\mathbf{l}_i$ which got the highest number of clicks on past preselected difficult queries
  $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + \alpha\mathbf{v}$ # update current model
end if
$\mathbf{v}_0, \mathbf{w}_0, c_0 \leftarrow$ Get the element in zero-th position of $\mathbf{R}$
for $i \leftarrow 1...m$ **do**
  $\mathbf{v}_i, \mathbf{w}_i, c_i \leftarrow$ Get the element in i-th position of $\mathbf{R}$
  if $c_i < c_0$ **then** # check for underperforming direction
    $push(Q, \mathbf{v}_i)$ # update queue
  end if
end for
return $\mathbf{w}_t$

Algorithm 13. *exploitCandidateDirectionsNSGD* function pseudocode.

The NSGD algorithm inspired the proposed algorithm in this disseration, where the null space of previously underperforming directions is used to guide the efficient exploration alongside an improved exploitation component, explained in Section 3.

2.3.3 Improving Exploitation

Previous outlined approaches [Hofmann et al., 2013b; Oosterhuis and de Rijke, 2017; Wang et al., 2018] focused on improving the exploration phase of OLTR algorithms while keeping the exploitation of candidate directions unmodified. Similar to DBGD, MGD, and CDB, they rely on a single exploitation control factor α to update the current model towards or away from candidate update directions.

To the best of our knowledge, Hofmann et al. [2011] were the only to propose to improve exploitation on OLTR algorithms. In particular, they proposed a method to balance exploration and exploitation by adding a complementary exploitation parameter k that controls the probability of choosing documents from a candidate ranking when interleaving, correcting the bias of introducing documents from an exploitatory ranking with a discount factor. The proposed method was empirically shown to perform effectively when compared to solely using the exploitation control factor α.

Although not vastly studied in the context of OLTR algorithms, exploitation strategies were extensively reviewed in the optimization area [Polyak, 1987]. One of the

simplest approaches to control exploitation in stochastic optimization algorithms, such as the previously mentioned Gradient Descent (Section 2.2.1), is to add a momentum [Polyak, 1964] factor in the model update phase, controlling the influence of the current model during the update. By adding a momentum factor we force a smoother path towards a local or global minimum when applying gradient descent, as shown in Figure 2.3, possibly reducing variance.

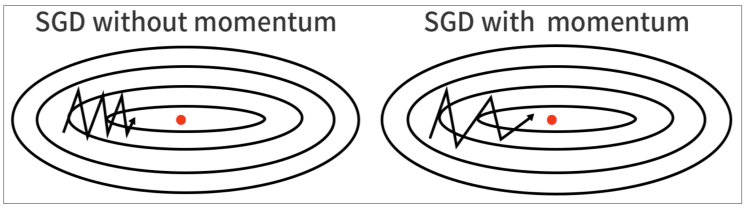


Figure 2.3. Comparison between stochastic gradient descent with and without momentum. Image taken from Du [2019] work.

As OLTR algorithms approximate the Gradient Descent algorithm [Flaxman et al., 2005], the momentum strategy can be applied during the exploitation phase of OLTR, particularly during the model update step. This idea serves as an inspiration to the adaptive exploitation component developed for the OLTR approach proposed in this dissertation.

2.4 Summary

In this chapter we reviewed several different studies covering the three main topics of this dissertation: automatic playlist generation, online learning to rank (OLTR), and efficient exploration and exploitation. We began by reviewing static and dynamic approaches for automatic playlist generation, outlining reinforcement learning (RL) as a framework for implementing dynamic strategies that adaptively change the current recommender model based on the user's feedback. Then, we introduced OLTR as a particular instance of a RL problem (named Contextual Multi-Armed Bandits (CMAB)), detailing different classical algorithms that use the OLTR framework to tackle the document ranking and sequential music recommendation tasks. Finally, we outlined the possible issues with the exploration and exploitation phases of those algorithms, introducing different proposed approaches to address them.

The proposed OLTR approach in this dissertation can be intuitively seen as an extension of CDB [Pereira et al., 2019] to incorporate the null space exploration mechanism of NSGD [Wang et al., 2018], with (un)promising directions determined via counterfactual reward estimates as opposed to (factual) multileaving in NSGD and

MGD. Moreover, to also improve exploitation, we propose a novel adaptive mechanism that considers both the promise of candidate models relative to the current model [Zhao and King, 2016], as well as the momentum [Polyak, 1964] carried by the current model itself. As demonstrated in our extensive experiments in Chapter 5, these mechanisms significantly contribute to improving both learning speed and convergence compared to the current state-of-the-art CDB approach for the sequential music recommendation task.

Chapter 3

Null Space Counterfactual Dueling Bandits

Online learning to rank approaches have been successfully deployed in contextual bandit settings where large item catalogs must be explored, such as in web search [Grotov and de Rijke, 2016]. Rather than learning user preferences over individual items, these approaches explore candidate ranking models defined in the space of item features. Candidate models are then compared to the current model in one [Yue and Joachims, 2009] or multiple [Schuth et al., 2016] duels by leveraging the user's feedback (typically clicks) on the rankings produced by these models. In contrast, for sequential music recommendation, at each time step, only one song is recommended and hence gets feedback (either a play or skip), which precludes a direct estimation of the reward attained by multiple dueling models. As an alternative, Pereira et al. [2019] proposed the CDB algorithm, which performs a counterfactual reward estimation of multiple exploratory dueling models by extrapolating from the single observed user feedback at each point in time.

Despite learning effective sequential music recommendation models, CDB suffers from one inefficiency which may impact its performance and, as a consequence, the user experience during the learning process. Just like previous OLTR approaches, CDB explores the space of candidate models uniformly at random [Flaxman et al., 2005], which may slow down convergence and increase variance throughout the learning process by repeatedly sampling previous underperforming directions [Wang et al., 2018]. To address this issue, inspired by NSGD [Wang et al., 2018], we propose Null Space Counterfactual Dueling Bandits (NSCDB), an efficient OLTR approach for sequential music recommendation. NSCDB aims to address the slow convergence behavior observed in the original CDB algorithm. In particular, instead of randomly exploring

the unit sphere around the current model, NSCDB improves exploration by restricting its search to the orthogonal subspace (also known as the null space) of previous underperforming models. As a result, it avoids updating the online recommender model towards directions of recognized inferior performance.

CDB also suffers from two other limitations, equally shared by NSGD, when exploiting candidate directions to update the current best model. On the one hand, both CDB and NSGD restrict updates to steer the current model either away from or towards candidate directions, respectively. On the other hand, both methods leverage the current best model at a fixed exploitation rate during updates, neglecting its ability to spawn future models (good or bad) in the subsequent learning iterations. To address these limitations, NSCDB implements bidirectional updates with momentum, allowing the current model to be updated adaptively depending on its estimated reward relative to that of candidate directions spawned from it.

Algorithm 14 presents the pseudocode of NSCDB. We chose not use the previously outlined OLTR canonical algorithm format with the purpose of generating a more self-contained algorithm. However, a similar notation is used to help identify the aforemetioned model initialization, retrieval of candidate items, exploration, and exploitation phases. Similarly to CDB [Pereira et al., 2019], NSCDB receives as input the current user u, a seed song $\mathbf{s}_0$, a feature extractor Φ, and a series of hyperparameters controlling the exploration and exploitation behavior of the algorithm. The algorithm starts by initializing a d-dimensional model vector $\mathbf{w}_0$ uniformly at random in line 1. Likewise, in line 2, a fixed-size queue $\mathcal{Q}$ is also initialized to keep track of the m most recent underperforming exploratory directions. Starting at line 3, online learning is performed iteratively in discrete time steps assuming an infinite horizon. At each time step t, candidate songs are retrieved from the available catalog and represented as a feature matrix $\mathbf{S}$ (line 4). Each candidate song $\mathbf{s} \in \mathbf{S}$ is represented as a d-dimensional feature vector according to the provided feature extractor Φ.[1] From the pool of candidates, the current model $\mathbf{w}_{t-1}$ then recommends the next song s_t to the user (line 5), who will in turn provide his or her feedback p_t (line 6). As in CDB, a positive feedback (payoff $p_t = 1$) will trigger the maintenance of the current model for the next iteration (line 8) with a reset of its estimated reward r to the maximum possible value (line 9). Otherwise, a negative feedback (payoff $p_t = 0$, line 10) penalizes the estimated reward of the current model (line 11) and triggers its update. As discussed in the remainder of this section, this encompasses the exploration of candidate directions (lines 12-14)

[1] In our experiments in Chapter 5, we adopt the same candidate retrieval and feature extraction strategies used by Pereira et al. [2019], which leverage information about the target user u and the songs $\mathbf{s}_0, \ldots, \mathbf{s}_{t-1}$ recommended in the session thus far.

NSCDB $(u, \mathbf{s}_0, \Phi, d, m, k, n, l, \delta, \alpha)$

1 $\mathbf{w}_0 \leftarrow rand(d)$ # initialize d-dimensional model
2 $\mathcal{Q} \leftarrow queue(m)$ # initialize fixed-size queue of recent bad directions
3 **for** $t \leftarrow 1$ **to** ∞ **do** # perform learning step
4 $\mathbf{S} \leftarrow \Phi(u, \mathbf{s}_0, \ldots, \mathbf{s}_{t-1})$ # retrieve candidate song feature vectors
5 $\mathbf{s}_t \leftarrow \arg\max_{\mathbf{s} \in \mathbf{S}} \langle \mathbf{w}_{t-1}, \mathbf{s} \rangle$ # recommend next song
6 $p_t \leftarrow payoff(\mathbf{s}_t)$ # receive user feedback
7 **if** $p_t > 0$ **then** # trigger maintenance of current model
8 $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1}$ # carry current model over to next time step
9 $r \leftarrow 1$ # reset current model reward to maximum
10 **else** # trigger model update
11 $r \leftarrow 1/(1/r + 1)$ # decrease current model reward estimate
12 $\mathcal{N} \leftarrow null(worst(\mathcal{Q}, k))$ # get null space of k worst directions
13 $\mathcal{B} \leftarrow sample(\mathcal{N}, n)$ # sample n null space vectors
14 $\mathcal{V} \leftarrow filter(\mathcal{B}, \mathbf{S}, l)$ # keep l vectors with highest coverage of $\mathbf{S}$
15 **for all** $\mathbf{v} \in \mathcal{V}$ **do** # perform duels
16 $\mathbf{w}_c \leftarrow \mathbf{w}_{t-1} + \delta \mathbf{v}$ # generate candidate model
17 $r_c \leftarrow 1 \ / \ \langle \mathbf{w}_c, \mathbf{S} \rangle^{-1}(\mathbf{s}_t)$ # estimate counterfactual reward
18 $e \leftarrow (r_c < r) \ ? \ +1 \ : \ -1$ # determine update orientation
19 **if** $e < 0$ **then** # check for bad exploration direction
20 $push(\mathcal{Q}, \mathbf{v})$ # update queue
21 **end if**
22 $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + e\alpha \mathbf{w}_c$ # update current model
23 **end for**
24 **end if**
25 **end for**

Algorithm 14. The NSCDB algorithm.

and their exploitation (lines 15-23).

3.1 Efficient Exploration

While the target objective function underlying online learning to rank approaches is unknown, uniformly sampling candidate models in the entire feature space at each time step ensures an unbiased estimate of its gradient [Flaxman et al., 2005]. On the other hand, uniform random exploration may lead to high variance, as regions of the feature space known to underperform are repeatedly revisited. Ultimately, convergence is impacted, which also contributes to a degraded user experience during learning. Inspired by the NSGD algorithm [Wang et al., 2018], we propose to trade-off unbiasedness for reduced variance by restricting our exploration to the orthogonal complement of the subspace induced by previous underperforming exploratory directions.

To avoid repeatedly exploring known underperforming directions, NSCDB maintains a fixed-size queue $\mathcal{Q}$ (initialized in line 2 and updated in line 20 of Algorithm 14) holding the m most recently identified underperforming exploratory directions (more on the estimation of candidate model rewards in Section 3.2). These serve as a selection of directions that recently led to bad recommendations and hence should be avoided, with hyperparameter m controlling the memory span of this mechanism. Inspired by NSGD, to have further control over this blocklist, we pick the k worst directions among the m kept thus far and compute their null space $\mathcal{N}$ (line 12). Because every vector in $\mathcal{N}$ is orthogonal to all k selected directions, the null space serves as a reduced, safer subspace for random exploration compared to the original d-dimensional space. Figure 3.1 illustrates the null space exploration. Accordingly, we sample n random unit vectors from $\mathcal{N}$ (line 13) as candidate exploratory directions. Following Wang et al. [2018], to ensure these directions are discriminative with respect to the current best model $\mathbf{w}_{t-1}$, we perform a further filter step (line 14). In particular, we keep the l directions with the highest inner product with respect to the average song feature vector $\bar{\mathbf{s}}$. Intuitively, as gradients to be potentially added to the current model $\mathbf{w}_{t-1}$, these directions are the most likely to cause a change in ranking scores compared to $\mathbf{w}_{t-1}$. The filtered set $\mathcal{V}$ is then used as our final exploratory set.

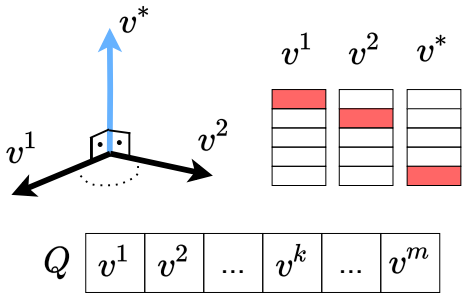


Figure 3.1. Null space exploration. Previous underpeforming candidate directions $\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^k$, stored in the queue $\mathcal{Q}$, are used to build the null space $\mathcal{N}$ containing multiple candidate directions $\mathbf{v}^*$ that are orthogonal to all worst k directions in $\mathcal{Q}$. As an example, $\mathbf{v}^1$ and $\mathbf{v}^2$ had previously been used to build candidate models that ranked a skipped song (highlighted in red) in the top positions of the ranking. As $\mathbf{v}^*$ is orthogonal to $\mathbf{v}^1$ and $\mathbf{v}^2$, it is expected to avoid directions associated with poor recommendations, possibly being used to build a new candidate model which would rank a skipped song in the bottom positions of the ranking.

3.2 Efficient Exploitation

Once exploratory directions have been generated, we must assess their potential as candidate recommendation models. To this end, we produce a candidate model $\mathbf{w}_c$ by

interpolating each candidate direction $\mathbf{v} \in \mathcal{V}$ with the current best model $\mathbf{w}_{t-1}$, with δ serving as an interpolation hyperparameter (line 16). In contrast to the current model $\mathbf{w}_{t-1}$, a candidate model $\mathbf{w}_c$ cannot be assessed directly, for it did not have a chance to expose its top recommended song $\mathbf{s}_t^c$ to the user. Rather than estimating its reward factually, we perform a counterfactual reward estimation of $\mathbf{w}_c$ based on how the user would have reacted to $\mathbf{s}_t^c$. Following Pereira et al. [2019], we assume this reward to be inversely proportional to the ranking distance between $\mathbf{s}_t^c$ and the song $\mathbf{s}_t$ actually recommended to the user (see line 5). In practice, this counterfactual reward r_c is equivalent to the reciprocal rank of $\mathbf{s}_t$ in the ranking induced by $\mathbf{w}_c$ (line 17).

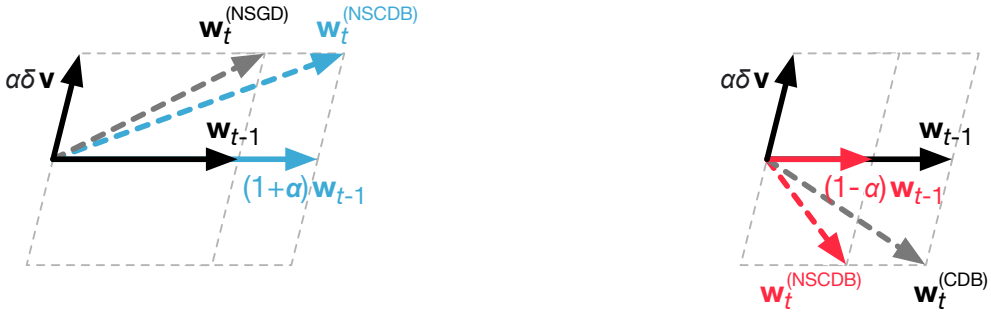
Because the current update was triggered by a negative feedback (a skip) on $\mathbf{s}_t$ (line 10), a higher reciprocal rank r_c actually means $\mathbf{w}_c$ would have underperformed compared to $\mathbf{w}_{t-1}$, whereas a lower r_c indicates the opposite. As previously discussed, both CDB [Pereira et al., 2019] and NSGD [Wang et al., 2018] would only act on one of these two possible cases. Precisely, because it avoids known bad directions via null space exploration, NSGD performs an optimistic exploitation, steering model updates towards sampled candidate directions. In contrast, CDB triggers a model update upon receiving a negative user feedback (a skip) and performs a pessimistic exploitation by driving updates away from directions that would otherwise promote the skipped song. In reality, a candidate direction orthogonal to previous underperforming ones is not guaranteed to perform well. Likewise, a bad direction might not be as bad as the current model, particularly after the current model received multiple skips in a row. Inspired by DP-DBGD [Zhao and King, 2016], where two opposite exploration directions are evaluated at each iteration to reduce exploratory uncertainty, we propose to unify both CDB and NSGD exploitation strategies into a bidirectional strategy. In particular, given the update orientation e determined by contrasting r and r_c (line 18), NSCDB can update the current model either towards a promising direction (i.e. $r_c < r$) or away from an unpromising one (i.e. $r_c \geq r$). In the latter case, the unpromising direction is also added to the queue $\mathcal{Q}$ (line 20) so it can be avoided in future explorations.

NSCDB also introduces a second exploitation enhancement focused on overcoming CDB and NSGD's lack of flexibility when leveraging the current best model $\mathbf{w}_{t-1}$ itself during an update. As previously discussed, both algorithms update the current model by interpolating it with either a promising (in the case of NSGD) or unpromising (in the case of CDB) direction $\mathbf{v}$, i.e. $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} \pm \alpha\mathbf{v}$, for a given exploitation hyperparameter α. In contrast, we propose to carry over the momentum [Polyak, 1964] of the current model $\mathbf{w}_{t-1}$ according to its ability to spawn a promising or unpromising candidate model $\mathbf{w}_c$. To better visualize this enhanced mechanism, we can substitute the definition of $\mathbf{w}_c$ (line 16) into NSCDB's model update equation (line 22). After

simple distributive expansions to factor out $\mathbf{w}_{t-1}$, we can rewrite line 22 as:

$$\mathbf{w}_t \leftarrow (1 + e\alpha)\mathbf{w}_{t-1} + e\alpha\delta\mathbf{v}, \tag{3.1}$$

where e controls the orientation (likewise, α controls the magnitude) of the contribution of not only the exploratory direction $\mathbf{v}$, but also of the current model $\mathbf{w}_{t-1}$. Figure 3.2 illustrates this mechanism geometrically for a toy example in a 2-dimensional feature space considering both a positive ($e = +1$) and a negative ($e = -1$) update.



(a) Positive update ($e = +1$). **(b)** Negative update ($e = -1$).

Figure 3.2. Geometric illustration of NSCDB model updates. Solid lines in black denote the current model $\mathbf{w}_{t-1}$ and exploratory direction $\mathbf{v}$ weighted by hyperparameters α, $\delta \in [0, 1]$. Solid lines in color illustrate the impact of the momentum term on $\mathbf{w}_{t-1}$ for NSCDB. Dashed lines denote resultant model vectors $\mathbf{w}_t$ according to NSCDB, CDB, and NSGD. (a) Positive update given a promising direction $\mathbf{v}$. (b) Negative update given an unpromising direction $\mathbf{v}$.

From Figure 3.2, we first observe that both CDB and NSGD perform model updates uniformly regardless of the relative promise of the exploratory direction $\mathbf{v}$ with respect to the current model $\mathbf{w}_{t-1}$. In contrast, for NSCDB, we observe its bidirectional update with momentum at play in two different circumstances. In Figure 3.2a, a promising ($e = +1$) exploratory direction $\mathbf{v}$ triggers an update towards $\mathbf{v}$ with an augmented contribution of the current model $\mathbf{w}_{t-1}$ (solid line in blue). In turn, in Figure 3.2b, an unpromising ($e = -1$) direction $\mathbf{v}$ triggers an update against $\mathbf{v}$ with a diminished contribution of $\mathbf{w}_{t-1}$ (solid line in red). As will be discussed in Section 5, these adaptive exploitation strategies coupled with the safer null space exploration strategy in NSCDB result in significant improvements in both learning speed and convergence.

3.3 Summary

In this chapter we outline the proposed OLTR algorithm in this disseration named Null Space Counterfactual Dueling Bandits (NSCDB). Inspired by CDB and NSGD, NSCDB addresses the exploration and exploitation inefficiencies of CDB by promoting null space exploration and adaptive bidirectional updates, respectively. Our thorough evaluation in Chapter 5 shows the superior performance of NSCDB in the sequential music recommendation task when compared to the state-of-the-art approach proposed by CDB.

Chapter 4

Experimental Setup

In this section, we describe the setup that supports our experiments in Section 5. We aim to answer the following research questions:

Q1. Can we improve exploration with null space sampling?

Q2. Can we improve exploitation with adaptive updates?

Q3. How sensitive is our approach to its hyperparameters?

In the following, we describe the dataset, preprocessing steps, and the overall evaluation methodology adopted in our experiments, which closely follows the one proposed by Pereira et al. [2019]. To facilitate reproducibility, all the code used in our experiments is available upon request.

4.1 Dataset

Our experiments are based on the Last.fm 1K dataset [Celma, 2010]. The dataset comprises historical listening sessions of nearly 1,000 users, from 2005 up to 2009. For a particular user ID, the dataset holds information such as timestamp, artist, and music title for each song the user has listened to in the period. Songs interrupted before 50% of their duration are discarded; all others are assumed to represent a play event [Bosteels and Kerre, 2009]. To reduce the possibility of a seasonality effect, the dataset is divided chronologically into five 6-month partitions, each containing at least 500 users, segmented into validation and test users, with each user having at least 500 listening events on average per month. The evaluation follows a sliding window with every two consecutive partitions used in pairs, with the first partition used as background data for feature extraction and the second for hyperparameter

Partition	Validation users	Test users
2007-01	100	380
2007-02	100	422
2008-01	100	444
2008-02	100	478

Table 4.1. Number of users in each dataset partition, segmented into validation and test sets.

tuning with validation users and evaluation with test users. Figure 4.1 illustrates the dataset partitioning procedure for evaluation, further detailed in Section 4.4. Overall, the preprocessed dataset contains over 660,000 unique songs, 800 unique users, and 11,223,000 play events. The final number of users in each partition, segmented by validation and test sets, is outlined in Table 4.1. Validation users have at least 400 listening events while test users have 500 listening events.

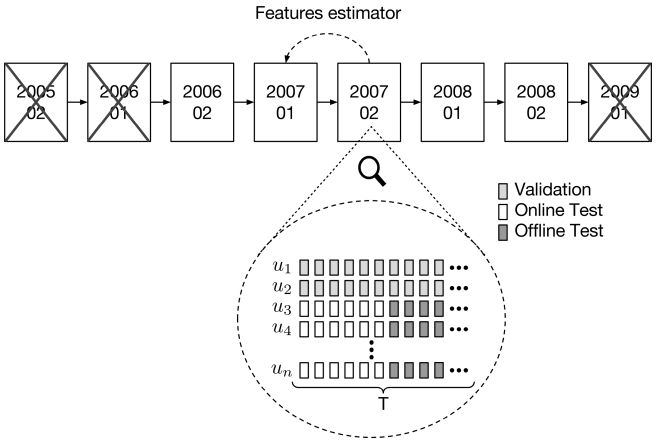


Figure 4.1. Dataset temporal partitioning logic. Each partition contains validation and test users. Validation users are used only for online evaluation (see Section 4.6) while test users are used for both online and offline evaluation. As an illustrative example, users u_1 and u_2 would be assigned to the validation set while users u_3 to u_n would be in the test set. 2005-02, 2006-01, and 2009-01 partitions are discarded due to not meeting the minimum number of users and listening events. Image taken from Pereira et al. [2019]

4.2 Feature Engineering

Although the Last.fm dataset compiles information about listening sessions of multiple users, storing song and artist unique identifiers, it does not contain data to generate social, content, and audio features. Hence, we had to augment the original dataset

by joining those unique artist and song identifiers with external sources. Content and social information was acquired using MusicBrainz API[1] and audio information was retrieved from Spotify API.[2]

For feature engineering, we use the same 22-dimensional feature representation adopted by Pereira et al. [2019], with features divided into 6 main groups:

1. **Collaborative**: collaborative features are based on the Collaborative Filtering framework [Su and Khoshgoftaar, 2009] where the information about a given user preference over items is computed by taking into account the preferences of other users. Collaborative features are generated following Volkovs and Yu [2015], explained as follows. We first build a binary matrix of implicit preferences $R_{U\times V}$, with U rows representing users and V columns representing items (songs or artists), where an element of R indicates whether or not user u listened or not to the song or artist v. From the binary matrix R we build a second matrix S containing estimated item preference information following a neighbor-based approach:

$$S(u,v) = \sum_{v' \in V(u)} R(:,v)^T R(:,v'), \tag{4.1}$$

where $S(u,v)$ indicates user u estimated preference for item v and $V(u)$ represents the list of items that the user interacted with. Higher $S(u,v)$ values indicates that v is similar to items that the user has previously expressed preference for. S is normalized following Volkovs and Yu [2015], using Equation 4.2:

$$S(:,v) = \frac{S(:,v)}{\sqrt{\sum_{u\in U} S(u,v)^2}} \tag{4.2}$$

The normalized matrix is used as input to a Truncated Singular Value Decomposition (T-SVD) [Xu, 1998] for matrix factorization through k latent factors:

$$S = U_k \Sigma_k V_k^T, \tag{4.3}$$

[1] `https://musicbrainz.org/doc/MusicBrainz_API`
[2] `https://developer.spotify.com/documentation/web-api/reference/`

where U_k represents each user and V_k^T represents each item on a k-dimensional latent space, with $k = 100$ latent dimensions.

2. **Social**: Social tags (i.e. single word or a small set of words) information associated with each song and artist [Eck et al., 2007] are used as input to a vector space model [Salton et al., 1975] aimed at representing textual content information in 2,638-dimensional word frequency vectors via frequency-inverse document frequency (TF-IDF).

3. **Content**: Similar to social tag features, content-based features are also built using a vector space model [Salton et al., 1975] using as input information such as song title and artist name, producing 176,564-dimensional word frequency vectors, also via TF-IDF.

4. **Audio**: Songs' audio characteristics such as acousticness, danceability, intensity, and tempo are represented on a 24-dimensional numeric feature vector.

5. **Derived**: Derived features are built using the information of the previously defined features, such as: (i) the SVD score (i.e. inner product elements of U_k and V_k^T) for a song, (ii) the SVD score for an artist, (iii) cosine similarity between a song and the song with highest associated SVD score for a given user, and (iv) cosine similarity between an artist and the artist with highest associated SVD score for a given user.

6. **Other**: Global song popularity (i.e. number of times a song was played) and song novelty score z are also used as features. The novelty score for a song is computed following Wang et al. [2014]:

$$z = 1 - e^{-t/h}, \tag{4.4}$$

where t represents the time elapsed since the user interacted with the song and h is a smoothing factor set to the default value suggested by the authors.

Collaborative, social, content, and audio features are computed for both songs and artists, being the artists represented as an average feature vector of all their respective songs. The independent collaborative, social, content, and audio feature representations of each song and artist are combined to form a unified contextual feature representation by computing pairwise cosine similarity between each isolated representation of the first song (artist) and the last played song (artist) in the listening session,

totalizing 16 similarity values. Aggregated with the 6 numerical values generated from derived and other feature groups, each song is represented as a final 22-dimensional contextual feature vector, detailed in Table 4.2.

Feature Group	Description	Number of Features
Collaborative	Cosine similarity between the SVD vectorized representation of the first played song/artist	2
Collaborative	Cosine similarity between the SVD vectorized representation of the last played song/artist	2
Social	Cosine similarity between the Social Tag vectorized representation of the first played song/artist	2
Social	Cosine similarity between the Social Tag vectorized representation of the last played song/artist	2
Content	Cosine similarity between the Textual Content vectorized representation of the first played song/artist	2
Content	Cosine similarity between the Textual Content vectorized representation of the last played song/artist	2
Audio	Cosine similarity between the Audio vectorized representation of the first played song/artist	2
Audio	Cosine similarity between the Audio vectorized representation of the last played song/artist	2
Derived	SVD score for the song/artist	2
Derived	Cosine similarity between a song and the song/artist with highest associated SVD score for a given user	2
Other	Global song popularity	1
Other	Song novelty score	1

Table 4.2. Detailed 22-dimensional contextual representation of a song.

4.3 Task Formulation

Our core task is to dynamically generate a playlist by sequentially recommending songs, one at a time, during a user's listening session, based on an initial seed song. This task formulation is implemented in popular music streaming services such as Spotify's "Start a Radio" functionality [Pelle, 2017]. Following Pereira et al. [2019], to ensure an unbiased evaluation, we simulate randomly logged feedback [Li et al., 2011; Gentile et al., 2017] by complementing the song actually played by the user at time t (taken as a positive example with payoff $p_t = 1$) with 99 other songs (taken as negative examples with payoff $p_t = 0$) randomly sampled from a set of 1,000 retrieved songs similar to the user's provided seed song s_0, using the cosine similarity with the SVD vectorized

representation previously mentioned in Section 4.2. As a result, the recommendation at time t can be cast as a ranking problem, namely, of selecting a single song from a list of 100 candidates, as illustrated by Figure 4.2.

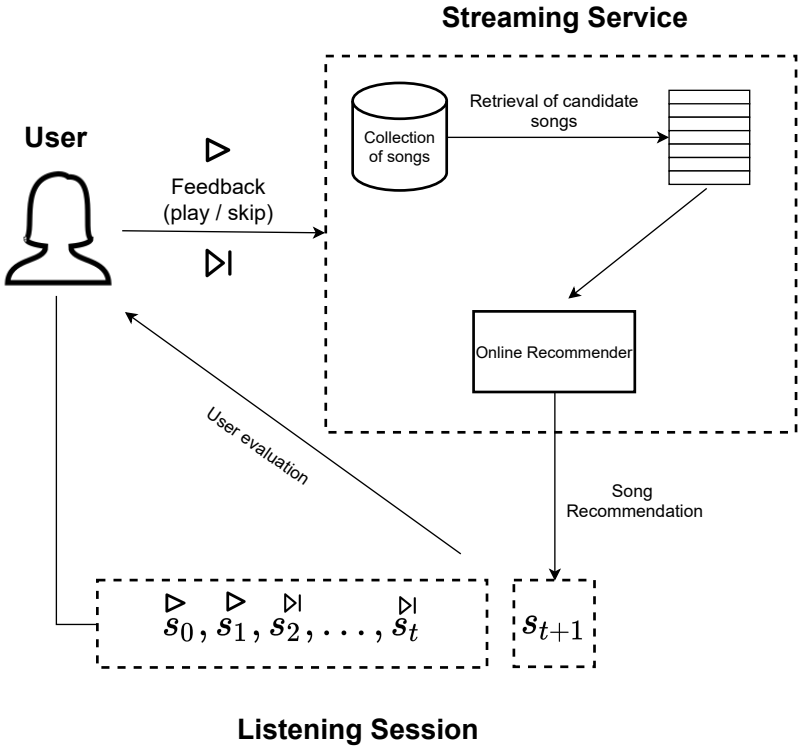


Figure 4.2. Task formulation illustration.

4.4 Evaluation Procedure

Following Pereira et al. [2019], we consider two complementary evaluation scenarios: online and offline, illustrated in Figure 4.3. The online evaluation captures the extent to which the user experience is impacted during learning. As our primary online evaluation metric, we consider the cumulative online payoff $P_{online}@t$ for a given user u up to time t, defined as:

$$P_{online}@t = \sum_{i=1}^{t} \lambda^i p_i, \tag{4.5}$$

where p_i is the payoff obtained at time i and $\lambda = 0.995$ is a discount factor used to discourage the execution of indefinitely many exploration steps before an exploitation action is taken [Hofmann et al., 2013a]. We report $P_{online}@t$ averaged across test users at each discrete time step $t \in \{1, \cdots, \kappa_1\}$, with $\kappa_1 = 400$ in our experiments.

In addition to the online test, we perform a complementary offline evaluation to assess the convergence of each recommendation model learned online. For a given user u, we use a fixed held-out test set with test cases chronologically placed after those used in the online evaluation for the same user. Precisely, given a model $\mathbf{w}_t$ learned online at time $t \in \{1, \cdots, \kappa_1\}$, we define its (instantaneous) offline payoff $P_{offline}@t$ as:

$$P_{offline}@t = \sum_{i=\kappa_1+1}^{\kappa_1+\kappa_2} p_i^{(t)}, \tag{4.6}$$

where $p_i^{(t)}$ is the payoff obtained by model $\mathbf{w}_t$ (learned online) at the i-th offline test case, for a total of $\kappa_2 = 100$ such test cases. Similarly to $P_{online}@t$, we report $P_{offline}@t$ averaged across test users at each discrete time step $t \in \{1, \cdots, \kappa_1\}$, with $\kappa_1 = 400$. To further ensure an unbiased evaluation, online and offline tests are repeated a total of six times, with results averaged across executions.

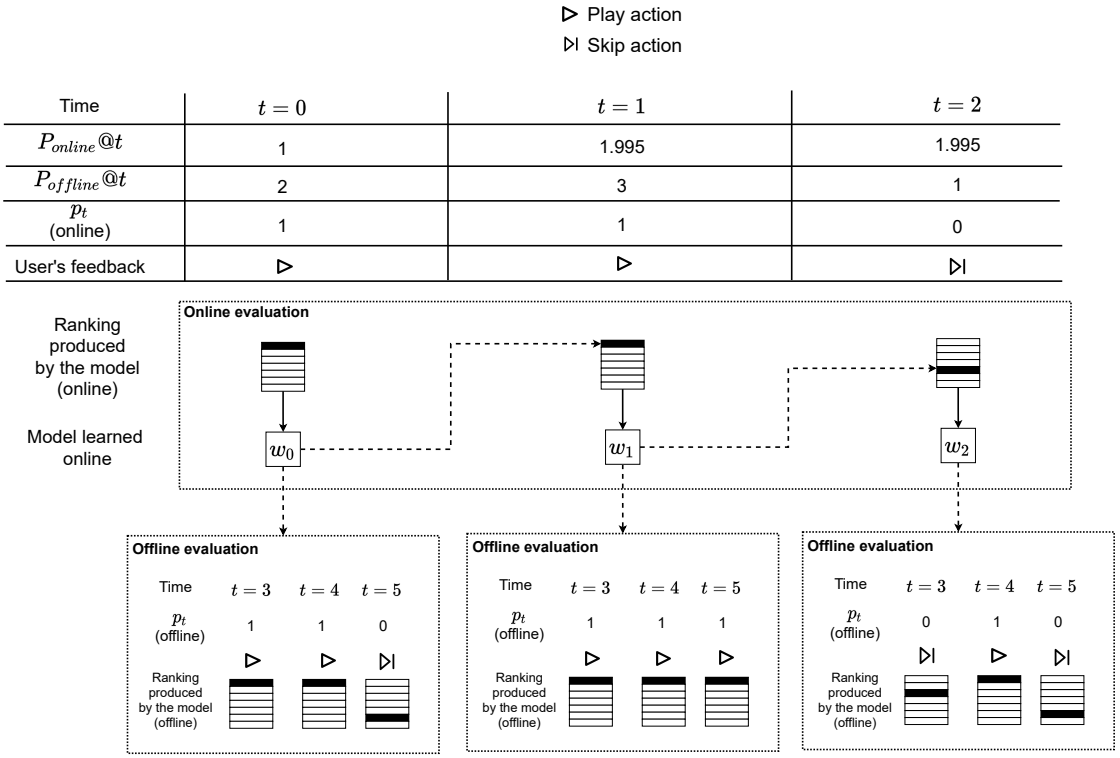


Figure 4.3. Example of online and offline evaluation procedures for a given user. At each time t a model $\mathbf{w}_t$ is learned online (solid arrow) after the user's feedback (play or skip), producing (dashed arrow) a single ranked list of songs for the current online test case and multiple rankings for each offline test case (out of time). A model is rewarded only if it places the user's actually played song (highlighted in black) at the top of the produced ranking. For the illustrative example we use $k_1 = k_2 = 3$. Diagram adapted from Pereira et al. [2019]

4.5 Experimental Baselines

As a state-of-the-art baseline in our investigations, we consider CDB [Pereira et al., 2019], which was originally proposed as an extension of online learning to rank algorithms such as DBGD [Yue and Joachims, 2009] and MGD Schuth et al. [2016] to tackle the scarce feedback scenario posed by sequential music recommendation. In particular, CDB was previously shown to outperform strong contextual bandit approaches [Li et al., 2011] in this scenario. In addition to the standard CDB implementation, we test multiple variants of it specially designed to help assess the impact of the exploration and exploitation components introduced in NSCDB.

4.6 Hyperparameter Tuning

For each 6-month partition in our sliding window setup (starting from the second; see Section 4.1), we retain the sessions of the first 100 users as validation data to tune the hyperparameters of NSCDB and CDB as a baseline. For CDB, these include the exploration rate δ and exploitation rate γ, both in the range $\{0.1, 0.2, \cdots, 0.9, 1.0\}$. For NSCDB, we tune the exploration rate δ and exploitation rate α, both in the range $\{0.25, 0.5, 0.75, 1\}$, queue size $m \in \{5, 10, 20\}$, number of top worst directions $k \in \{1, 2, 3, 5\}$ to build the null space, number of null space directions $n \in \{5, 10, 20\}$ to sample, and number of top preselected null space directions $l \in \{1, 3, 5\}$. For both NSCDB and CDB, the best hyperparameter settings are chosen for each individual partition in order to maximize P_{online}@400 for the validation users in that partition.

4.7 Summary

In this chapter we detailed the experimental setup used to evaluate the performance of the proposed NSCDB algorithm which applies the same methodology by Pereira et al. [2019]. We began by outlining the characteristics of the dataset used in this evaluation and how it was segmented into different partitions. Then, we explained the feature engineering pipeline applied to build vectorized song representations based on contextual information provided in the listening session. We formalized the sequential music recommendation task, also outlining the online and offline evaluation procedures employed to assess the performance of NSCDB. Finally, we use CDB as a strong baseline for comparison, listing the hyperparameter tuning procedure performed for both CDB and NSCDB to select the best model during validation.

Chapter 5

Results and Discussion

In this section, we evaluate our proposed NSCDB algorithm by addressing the research questions posed in Section 4. For each question, we compare the original NSCDB algorithm performance against its variants, contrasting with CDB as a baseline result. Online and offline results for questions *Q1* and *Q2* are executed 6 times with different random seeds and reported as the average across users in all test partitions described in Section 4. The hyperparameter sensitivity analysis for *Q3* is performed using validation results. All results apply a 95% confidence interval estimation, computed using bootstrap sampling [Efron, 1992] with 1000 samples.

5.1 Null Space Exploration

In this initial experiment, we aim to address research question *Q1*, by assessing the impact of our proposed null space exploration strategy. Figure 5.1 presents online and offline performance results for NSCDB (which explores in the null space of previous underperforming exploratory directions) in contrast to four variants of CDB (which explore uniformly at random in the entire feature space): the standard CDB (with unidirectional gradient updates), CDB-M (with momentum updates), CDB-B (with bidirectional updates), and CDB-MB (with bidirectional updates with momentum).

From Figure 5.1, we first note that NSCDB consistently and significantly outperforms all CDB variants, regardless of their deployed exploitation strategy (unidirectional, bidirectional, with or without momentum updates). Indeed, not only does NSCDB offer an improved user experience during learning as shown in Figure 5.1a, but it also improves learning speed and convergence as shown in Figure 5.1b. Indeed, NSCDB converges substantially faster and to a higher performance level compared to all CDB variants. Recalling *Q1*, this remarkable result attests to the positive impact

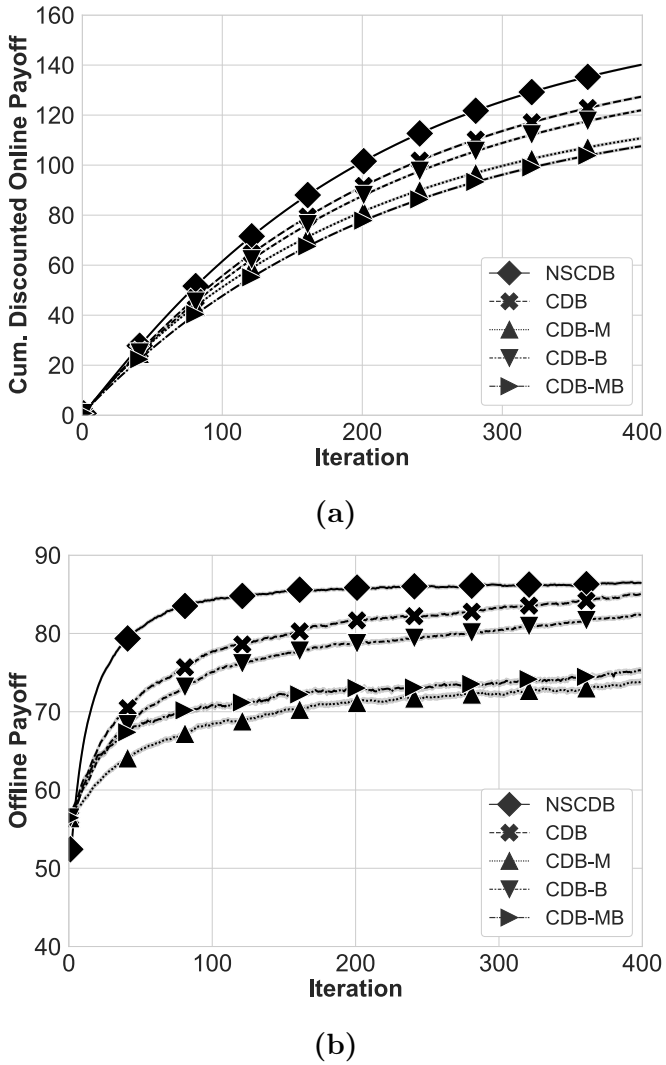


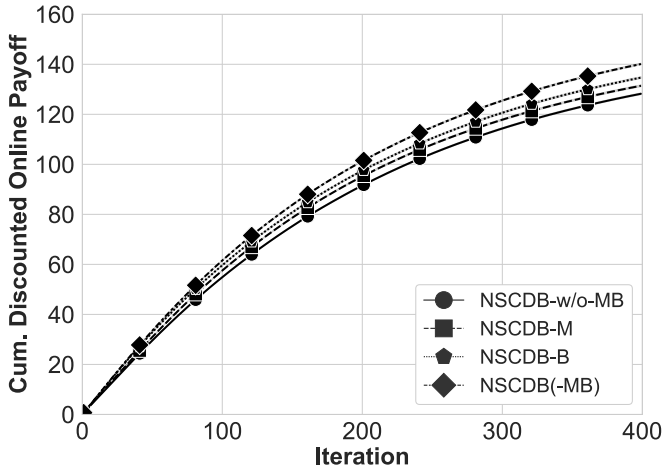
Figure 5.1. (a) Average cumulative online payoff and (b) average offline payoff for different exploration strategies.

of the null space exploration strategy in NSCDB compared to the random exploration strategy in CDB. Interestingly, the standard CDB is still the best-performing among the considered CDB variants, which suggests the more sophisticated exploitation strategies of momentum and bidirectional updates are too risky when deployed in conjunction with pure random exploration.

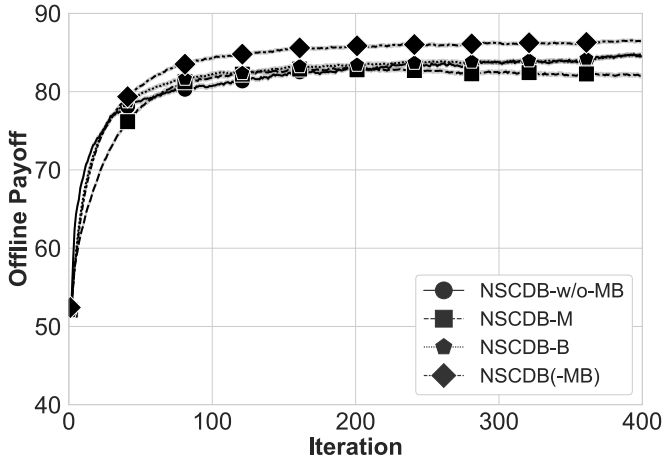
5.2 Adaptive Exploitation

The results in Section 5.1 demonstrated the positive impact of the null space exploration strategy in NSCDB. To complement our investigation, we now fix the null space exploration strategy and turn to *Q2* to investigate the impact of the adaptive exploitation

strategies in NSCDB. Figure 5.2 shows the results of this investigation, once again in terms of both online and offline performance, but now for multiple variants of NSCDB: the standard NSCDB (here denoted NSCDB(-MB) to explicitly highlight both its bidirectional and momentum update strategies), NSCDB-M (momentum only), NSCDB-B (bidirectional updates only), and NSCDB-w/o-MB (neither bidirectional updates nor momentum).



(a)



(b)

Figure 5.2. (a) Average cumulative online payoff and (b) average offline payoff for different exploitation strategies.

From Figure 5.2, we first observe that both NSCDB-M and NSCDB-B contribute to improving upon the baseline variant NSCDB-w/o-MB, which uses the simple unidirectional exploitation strategy without momentum present in CDB. This result is particularly pronounced in terms of online performance as shown in Figure 5.2a, with both NSCDB-M and NSCDB-B attaining a higher cumulative online payoff compared

to NSCDB-w/o-MB throughout the session. The offline results in Figure 5.2b further attest to the effectiveness of NSCDB-B, particularly in early iterations. In turn, by steering updates exclusively away from both unpromising directions as well as the current best model, NSCDB-M converges more slowly and may even penalize the overall performance towards later iterations. Nonetheless, consistent and significant improvements can be observed in terms of both online and offline performance when both strategies are combined in the full NSCDB(-MB) variant. Recalling *Q2*, these results demonstrate the complementary nature of our proposed exploitation strategies and their combined positive impact in terms of both online user experience as well as offline learning convergence. Moreover, contrasting the results of NSCDB(-MB) with those previously attained by CDB-MB in Figure 5.1 further corroborates the contribution of the null space exploration strategy at identifying safer directions for exploitation.

5.3 Hyperparameters Sensitivity

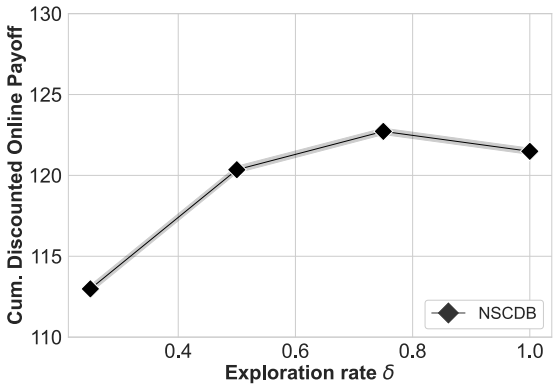
In this last experiment, we address *Q3* by investigating the impact of NSCDB's various hyperparameters on the overall performance of the algorithm. For this analysis, we target P_{online}@400, which was chosen to guide hyperparameter tuning in the first place. Figures 5.3a-f show the results of this investigation, which we discuss next.

Exploration rate δ As shown in Figure 5.3a, NSCDB shows an upward trend in performance as we increase δ with a peak around $\delta = 0.8$. This suggests that our safer null space exploration coupled with our proposed adaptive exploitation strategies encourages a more aggressive exploration of the identified candidate directions.

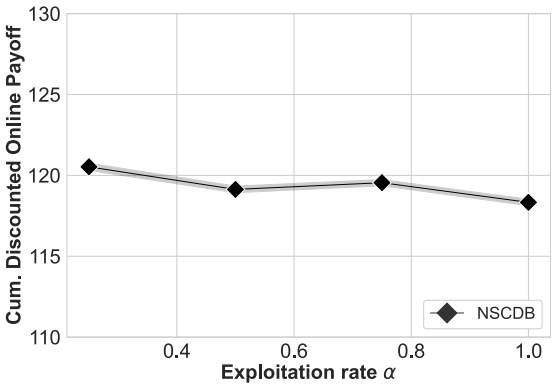
Exploitation rate α In contrast to δ, we observe a slight downward trend as we increase the exploitation rate α in Figure 5.3b. While seemingly minor, this trend might suggest a potential overshooting of the true gradient, with each subsequent learned model stepping too far away from the previous one.

Queue size m Results are relatively stable when we increase m in Figure 5.3c. Since we control the number k of worst directions from which to compute the null space, increasing m only affects space and execution time as we need to maintain and evaluate more directions.

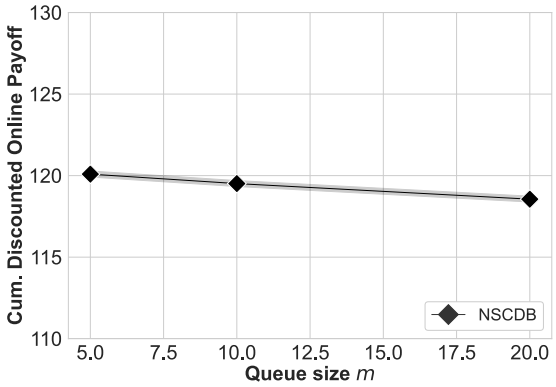
Number k of worst directions As shown in Figure 5.3d, increasing k leads to poor performance. As observed by Wang et al. [2018], the null space has rank at most $d-k$.



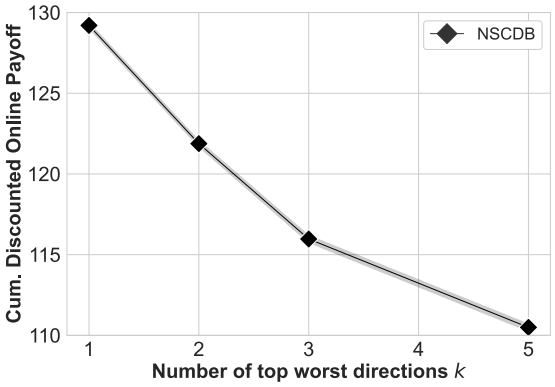
(a) Exploration rate δ.



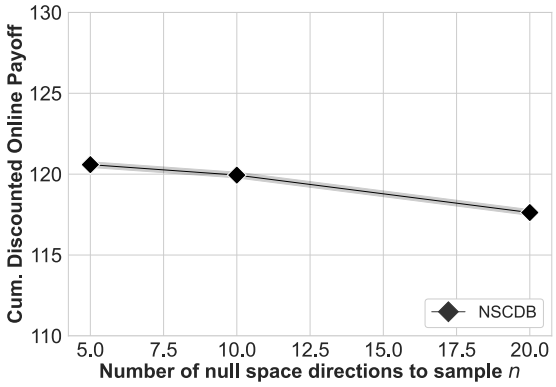
(b) Exploitation rate α.



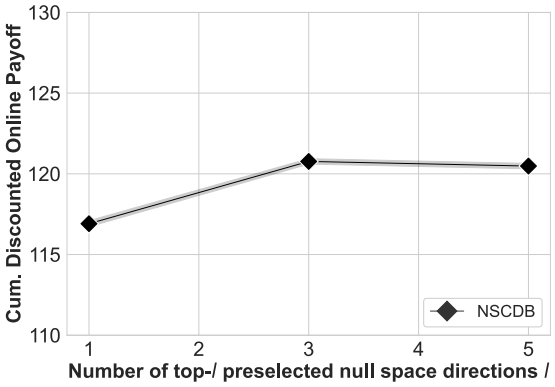
(c) Queue size m.



(d) Number k of worst directions.



(e) Number n of sampled NS directions.



(f) Number l of exploratory directions.

Figure 5.3. Hyperparameters sensitivity analysis for NSCDB. Cumulative discounted online payoff results are fixed at the last listening session iteration.

Intuitively, a large k induces a restricted null space, and hence limits the exploratory capacity of NSCDB, forcing it to converge to a suboptimal model.

Number n of sampled null space directions We observe a slight downward trend in Figure 5.3e as we increase n. While a larger n means more possibilities to explore, it also

increases the risk of sampling noisy null space directions that can harm performance.

Number of exploratory directions l As shown in Figure 5.3f, increasing l may be beneficial, primarily since these directions undergo a filtering step to retain those with high discriminative power and hence a high potential to contribute to the current best model.

5.4 Summary

In this chapter we presented the experimental results of the proposed Null Space Counterfactual Dueling Bandits (NSCDB) algorithm, contrasting its performance with the current state-of-the-art Counterfactual Dueling Bandits (CDB) algorithm and variants. Experiments were designed in the hope of answering our three research questions outlined in Chapter 4. We demonstrated through online and offline evaluation procedures that NSCDB outperforms CDB on both early-stage performance and final convergence, producing recommender models that fastly learn users' preferences through implicit feedback.

In particular, we showed that NSCDB's null space exploration is indeed a better strategy when compared to pure random exploration performed by CDB, answering our first research question. Next, we assessed the impact of our proposed exploitation strategy, evaluating its internal components (momentum and bidirectional updates) isolatedly and combined. We showed that both exploitation components complement the performance gains attained using null space exploration, answering our second research question. Finally, we investigated the impact of each one of NSCDB hyperparameters, reaching similar conclusions presented by Wang et al. [2018], however with a focus on the sequential music recommendation task.

Chapter 6

Conclusions and Future Work

Music streaming services are present in the lives of millions of users across the globe, offering an enormous and diverse catalog of songs through a personalized experience. Playlists can be identified as one of the main personalization features available in these services, being manually or automatically generated for users. Although manual curation of playlists can allow more ownership and personalization, this process requires much effort and time due to the huge amount of songs available. Automatic playlist generation was proposed as a solution to avoid the laborious process of manual approaches, while still guaranteeing a personalized experience for the user, being actively explored by researchers.

Many different approaches were designed to tackle the automatic playlist generation problem, mainly through static and dynamic strategies. On the one hand, although successful, static strategies produce a fixed list of songs based on past listening habits that cannot adaptively change to the user's current context. On the other hand, dynamic strategies leverage the current listening session and the short-term history to provide an adaptive recommendation experience through user feedback.

Recently, Online Learning to Rank (OLTR) dynamic strategies were proposed to address the automatic playlist generation problem, in particular the sequential music recommendation task. Although successful, such approaches employ a random exploration component that can potentially harm user experience by repeating similar previously known unsuitable recommendations. Moreover, these approaches also rely on a fixed exploitation component that does not consider the current recommender performance in the online update phase, further affecting overall performance.

As a potential solution to the previously outlined exploration and exploitation gaps, we proposed Null Space Counterfactual Dueling Bandits (NSCDB), an efficient OLTR algorithm for sequential music recommendation. NSCDB seeks to improve ex-

ploration efficiency by sampling candidate directions from the null space of previous underperforming ones. To also improve exploitation efficiency, NSCDB adaptively leverages the current model according to the estimated reward of candidate directions spawned from it. Thorough experiments using simulated listening sessions from Last.fm show that NSCDB achieves better learning speed and convergence results when contrasted to state-of-the-art algorithms such as Counterfactual Dueling Bandits (CDB).

6.1 Summary of Findings

Here we outline the main findings of this dissertation:

- Null space exploration [Wang et al., 2018] strategies previously used to improve OLTR approaches targetting the document ranking scenario can be adapted for music recommendation, improving current state-of-the-art OLTR methods [Pereira et al., 2019] focused on addressing the sequential music recommendation task. By sampling new candidate model update directions from the null space of recent underperforming ones, we avoid exploring subspaces of known inferior performance.

- Simple exploitation strategies such as momentum [Polyak, 1964], used in classical optimization algorithms, can be efficiently integrated into OLTR algorithms targetting the sequential music recommendation task to improve performance, provided that null space exploration is performed. Once a safer subspace of exploration is provided via null space sampling, we can adaptively leverage the influence of the current model in the update phase by having access to the estimated performance of candidate directions spawned from it.

- The observed performance impact of null space exploration hyperparameters such as the number k of worst directions, used to build the null space, follows a similar pattern on both document ranking and sequential music recommendation tasks. This suggests a generalized task-agnostic behavior that can be explored by hyperparameter tuning heuristics to reduce search execution time.

6.2 Future Directions

Based on our findings, we list the following possible future research directions:

- Variance reduction methods for gradient exploration [Wang et al., 2019] could be an alternative to the null space strategy by narrowing the exploration down to the projection space of recommended items.
- Strategies that control the OLTR algorithms' exploration phase by directly modeling distributions over rankings [Oosterhuis and de Rijke, 2018], instead of resorting to extensive sampling, could also be good candidate exploration approaches.
- We would like to investigate other exploitation rate schedules that are sensitive to individual user characteristics, such as their propensity to switch contexts at different points during a session [Meng and Shi, 2020].
- We also plan to assess the adaptability of NSCDB to other scarce feedback scenarios, such as mobile push notifications [Mehrotra et al., 2019].
- Last.fm simulated listening sessions data served as a robust initial experimental input to assess online and offline performance of NSCDB, focused on sessions with a large time horizon (i.e. 400 iterations). In contrast, we would like to evaluate NSCDB performance on shorter listening sessions data [Chen et al., 2018] to verify if the efficient convergence is kept in such scenario.

Bibliography

Baccigalupo, C. and Plaza, E. (2006). Case-based sequential ordering of songs for playlist recommendation. In *ECCBR*, pages 286--300.

Bonnin, G. and Jannach, D. (2014). Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):1--35.

Bosteels, K. and Kerre, E. E. (2009). A fuzzy framework for defining dynamic playlist generation heuristics. *Fuzzy Sets and Systems*, 160(23).

Bouneffouf, D., Rish, I., and Aggarwal, C. (2020). Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1--8. IEEE.

Celma, O. (2010). *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. 1st edition.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24:2249--2257.

Chen, C.-W., Lamere, P., Schedl, M., and Zamani, H. (2018). Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 527--528.

Chen, S., Moore, J. L., Turnbull, D., and Joachims, T. (2012). Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 714--722.

Cheng, Z., Jialie, S., and Hoi, S. C. (2016). On effective personalized music retrieval by exploring online user behaviors. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 125--134.

Dias, R., Gonçalves, D., and Fonseca, M. J. (2017). From manual to assisted playlist creation: a survey. *Multimedia Tools and Applications*, 76(12):14375--14403.

Du, J. (2019). The frontier of sgd and its variants in machine learning. In *Journal of Physics: Conference Series*, volume 1229, page 012046. IOP Publishing.

Eck, D., Lamere, P., Bertin-Mahieux, T., and Green, S. (2007). Automatic generation of social tags for music recommendation. *Advances in neural information processing systems*, 20:385--392.

Efron, B. (1992). Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569--593. Springer.

Flaxman, A. D., Kalai, A. T., and McMahan, H. B. (2005). Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 385--394.

Gartner, D., Kraft, F., and Schaaf, T. (2007). An adaptive distance measure for similarity based playlist generation. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 1, pages I--229. IEEE.

Gentile, C., Li, S., Kar, P., Karatzoglou, A., Zappella, G., and Etrue, E. (2017). On context-dependent clustering of bandits. In *Proc. of ICML*, pages 1253--1262.

Ghorab, M. R., Zhou, D., O'connor, A., and Wade, V. (2013). Personalised information retrieval: survey and classification. *User Modeling and User-Adapted Interaction*, 23(4):381--443.

Grotov, A. and de Rijke, M. (2016). Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1215--1218.

Hagen, A. N. (2015). The playlist experience: Personal playlists in music streaming services. *Popular Music and Society*, 38(5):625--645.

Hariri, N., Mobasher, B., and Burke, R. (2012). Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 131--138.

Hariri, N., Mobasher, B., and Burke, R. (2015). Adapting to user preference changes in interactive recommendation. In *Proc. of ICAI*, pages 4268--4274.

Hauver, D. B. and French, J. C. (2001). Flycasting: Using collaborative filtering to generate a playlist for online radio. In *Proc. of IEEE Web Delivering of Music*, pages 123--130.

Hofmann, K. et al. (2013a). Fast and reliable online learning to rank for information retrieval. In *SIGIR Forum*, volume 47, page 140.

Hofmann, K., Schuth, A., Whiteson, S., and De Rijke, M. (2013b). Reusing historical interaction data for faster online learning to rank for ir. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 183--192.

Hofmann, K., Whiteson, S., and De Rijke, M. (2011). Balancing exploration and exploitation in learning to rank online. In *European Conference on Information Retrieval*, pages 251--263. Springer.

Hu, B., Shi, C., and Liu, J. (2017). Playlist recommendation based on reinforcement learning. In *International Conference on Intelligence Science*, pages 172--182. Springer.

Jawaheer, G., Szomszor, M., and Kostkova, P. (2010). Comparison of implicit and explicit feedback from an online music recommendation service. In *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, pages 47--51.

Kamehkhosh, I., Bonnin, G., and Jannach, D. (2020). Effects of recommendations on the playlist creation behavior of users. *User Modeling and User-Adapted Interaction*, 30(2):285--322.

King, J. and Imbrasaitė, V. (2015). Generating music playlists with hierarchical clustering and q-learning. In *Advances in Information Retrieval*, pages 315--326.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proc. of WWW*, pages 661--670.

Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297--306.

Liebman, E., Saar-Tsechansky, M., and Stone, P. (2015). Dj-mc: A reinforcement-learning agent for music playlist recommendation.

Logan, B. (2002). Content-based playlist generation: Exploratory experiments. In *ISMIR*, volume 2, pages 295--296.

Mehrotra, A., Hendley, R., and Musolesi, M. (2019). Notifymehere: Intelligent notification delivery in multi-device environments. In *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval*, pages 103--111.

Meng, L. and Shi, C. (2020). A context-aware interest drift network for session-based news recommendations. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 1967--1971. IEEE.

Moore, J. L., Chen, S., Joachims, T., and Turnbull, D. (2012). Learning to embed songs and tags for playlist prediction. In *ISMIR*, volume 12, pages 349--354.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Oosterhuis, H. and de Rijke, M. (2017). Balancing speed and quality in online learning to rank for information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 277--286.

Oosterhuis, H. and de Rijke, M. (2018). Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1293--1302.

Oulasvirta, A., Hukkinen, J. P., and Schwartz, B. (2009). When more is less: the paradox of choice in search engine use. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 516--523.

Pampalk, E., Pohle, T., and Widmer, G. (2005). Dynamic playlist generation based on skipping behavior. In *Proc. of ISMIR*, pages 634--637.

Pelle, S. (2017). More of the same–on spotify radio. *Culture Unbound. Journal of Current Cultural Research*, 9(2):184--211.

Pereira, B. L., Ueda, A., Penha, G., Santos, R. L., and Ziviani, N. (2019). Online learning to rank for sequential music recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 237--245.

Pichl, M., Zangerle, E., and Specht, G. (2017). Understanding user-curated playlists on spotify: A machine learning approach. *International Journal of Multimedia Data Engineering and Management (IJMDEM)*, 8(4):44--59.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1--17.

Polyak, B. T. (1987). Introduction to optimization. optimization software. *Inc., Publications Division, New York*, 1.

Quadrana, M., Cremonesi, P., and Jannach, D. (2018). Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51(4):1--36.

Radlinski, F., Kurup, M., and Joachims, T. (2008). How does clickthrough data reflect retrieval quality? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 43--52.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613--620.

Schuth, A., Oosterhuis, H., Whiteson, S., and de Rijke, M. (2016). Multileave gradient descent for fast online learning to rank. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 457--466.

Schuth, A., Sietsma, F., Whiteson, S., Lefortier, D., and de Rijke, M. (2014). Multileaved comparisons for fast online evaluation. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 71--80.

Spall, J. C. (2005). *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons.

Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Vall, A. (2015). Listener-inspired automated music playlist generation. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 387--390.

Vall, A., Dorfer, M., Schedl, M., and Widmer, G. (2018). A hybrid approach to music playlist continuation based on playlist-song membership. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1374--1382.

Volkovs, M. and Yu, G. W. (2015). Effective latent models for binary feedback in recommender systems. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 313--322.

Wang, H., Kim, S., McCord-Snook, E., Wu, Q., and Wang, H. (2019). Variance reduction in gradient exploration for online learning to rank. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 835--844.

Wang, H., Langley, R., Kim, S., McCord-Snook, E., and Wang, H. (2018). Efficient exploration of gradient space for online learning to rank. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 145--154.

Wang, S., Cao, L., Wang, Y., Sheng, Q. Z., Orgun, M. A., and Lian, D. (2021). A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(7):1--38.

Wang, X., Wang, Y., Hsu, D., and Wang, Y. (2014). Exploration in interactive personalized music recommendation: a reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):1--22.

Xing, Z., Wang, X., and Wang, Y. (2014). Enhancing collaborative filtering music recommendation by balancing exploration and exploitation. In *Proc. of ISMIR*, pages 445--450.

Xu, P. (1998). Truncated svd methods for discrete linear ill-posed problems. *Geophysical Journal International*, 135(2):505--514.

Yam, J. Y. and Chow, T. W. (2000). A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219--232.

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

Zhao, T. and King, I. (2016). Constructing reliable gradient exploration for online learning to rank. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1643--1652.