Online Learning to Rank for Sequential Music Recommendation

Bruno L. Pereira
CS Dept., UFMG
Belo Horizonte, MG, Brazil
brunolaporais@dcc.ufmg.br

Alberto Ueda
CS Dept., UFMG
Belo Horizonte, MG, Brazil
ueda@dcc.ufmg.br

Gustavo Penha
TU Delft
Delft, The Netherlands
g.penha-1@tudelft.nl

Rodrygo L. T. Santos
CS Dept., UFMG
Belo Horizonte, MG, Brazil
rodrygo@dcc.ufmg.br

Nivio Ziviani
CS Dept., UFMG & Kunumi
Belo Horizonte, MG, Brazil
nivio@dcc.ufmg.br

ABSTRACT

The prominent success of music streaming services has brought increasingly complex challenges for music recommendation. In particular, in a streaming setting, songs are consumed sequentially within a listening session, which should cater not only for the user's historical preferences, but also for eventual preference drifts, triggered by a sudden change in the user's context. In this paper, we propose a novel online learning to rank approach for music recommendation aimed to continuously learn from the user's listening feedback. In contrast to existing online learning approaches for music recommendation, we leverage implicit feedback as the only signal of the user's preference. Moreover, to adapt rapidly to preference drifts over millions of songs, we represent each song in a lower dimensional feature space and explore multiple directions in this space as duels of candidate recommendation models. Our thorough evaluation using listening sessions from Last.fm demonstrates the effectiveness of our approach at learning faster and better compared to state-of-the-art online learning approaches.

CCS CONCEPTS

• **Theory of computation → Online learning theory**; • **Computing methodologies → Online learning settings**; **Learning from implicit feedback**.

KEYWORDS

Music recommendation, online learning to rank, implicit feedback

ACM Reference Format:
Bruno L. Pereira, Alberto Ueda, Gustavo Penha, Rodrygo L. T. Santos, and Nivio Ziviani. 2019. Online Learning to Rank for Sequential Music Recommendation. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347019

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347019

1 INTRODUCTION

Music consumption has changed tremendously in recent years with the increasing adoption of on-demand streaming services [8, 35], such as Spotify and Apple Music. By providing instant, ubiquitous access to millions of songs ranging from mainstream to independent artist productions, these services brought unique challenges for music recommendation. One key challenge that is of particular interest to this paper stems from the sequential nature of music consumption via playlists [35]. While manually curated playlists are popular, their content is agnostic to the preferences of individual users. Moreover, they cannot scale to cover the catalogs of modern streaming services, which severely limits discovery [33]. As a result, automatic playlist generation has seen substantial research from the music recommendation community [2, 5, 6, 12, 29].

Automatic playlist generation is typically triggered by a seed song provided by the user [5, 30]. This song is then expanded into a personalized playlist by some recommendation model encompassing one or more of a variety of features extracted from the user's historical listening activity [1, 9], current context [29], audio [4, 12, 27] or metadata [26] content of each song, as well as its popularity [16]. One key limitation of these approaches is that the recommendation model is learned offline, regardless of the feedback provided by the user *during* the current listening session. On the other hand, while online learning approaches for music recommendation have been investigated [15, 25, 38], they leverage explicit feedback signals (e.g., ratings) to guide the learning process, which requires an arguably intrusive interaction mechanism [18].

In this paper, we propose an online learning approach for sequential music recommendation aimed to continuously learn from the user's listening feedback. Given the large catalogs available in modern music streaming services, incrementally learning personalized preferences for individual songs may become a bottleneck, limiting the system's ability to explore beyond the user's historical preferences. To overcome this limitation, inspired by online learning to rank approaches for search [34, 40], we represent songs in a lower dimensional feature space and explore multiple directions in this space as duels of candidate recommendation models. To determine the winner among dueling models at each point in time, we leverage implicit feedback signals, such as song plays and skips, to improve the recommendation of the next song in a playlist. While such feedback can be unintrusively acquired as part of the user's interaction

with the system, it is typically a noisy reflection of the user's true preference [18, 19]. Moreover, only one song is recommended and hence only one such feedback is available at each point in time, which hinders our ability to evaluate multiple dueling models. To enhance model exploration, we propose a counterfactual estimation of the effectiveness of different models based on how they would have ranked that particular song at that time. To our knowledge, ours is the first attempt to perform online learning to rank in a personalized and feedback-scarce scenario. Thorough experiments using listening sessions from Last.fm attest the effectiveness of our approach at learning better recommendation models and with faster convergence compared to state-of-the-art online learning approaches from the literature.

In summary, our main contributions are two-fold:

- We propose a novel online learning to rank approach for music recommendation, capable of leveraging the user's implicit feedback via play or skip actions;
- We thoroughly assess the proposed approach in terms of its learning effectiveness and convergence.

In the remainder of this paper, Section 2 discusses related work on music recommendation and online learning approaches. Section 3 describes our proposed online learning to rank approach for music recommendation. Sections 4 and 5 describe the experimental setup and the results of our thorough investigation. Lastly, Section 6 provides our concluding remarks and directions for future work.

2 RELATED WORK

In this section, we describe related approaches for recommending personalized sequences of songs as a playlist as well as approaches for learning users' preferences in an online fashion.

2.1 Automatic Playlist Generation

Manual playlist generation requires substantial effort from human experts to create sequences of songs aimed to please as many users and tastes as possible. Moreover, the generated playlists may be biased toward the general popularity of individual songs or the experts' personal tastes. Instead, we focus on the automatic generation of personalized playlists [5, 21]. Static approaches leverage past user interactions. For instance, Ragno et al. [32] proposed a Markov random field approach by connecting songs that co-occur in manually generated playlists into an undirected graph. Using a seed song, provided by the user, as the starting point of a random walk on this graph, a playlist is recommended. Similarly, Flexer et al. [12] considered a scenario where both a seed song and a target song are provided and proposed to create a playlist that transitions smoothly between these songs by leveraging their audio features.

In contrast to static approaches, we target a sequential playlist generation scenario, where songs are dynamically recommended by taking into account user interactions during their listening session [31]. This scenario acknowledges that users' preferences may be affected by their listening context, such as their current mood or current activity (e.g., running or studying). As such, it incurs additional challenges such as the dynamic adaptation of user preference models [11, 14, 39] and the identification of when a particular contextual change leads to preference shifts [15]. A number of heuristic approaches have been proposed to tackle this problem. For instance, Pampalk et al. [29] used implicit feedback signals to steer the next recommended song in a playlist toward (in case of a play) or away from (in case of a skip) the current song by exploiting audio similarities. Such heuristics were later formalized and extended by Bosteels and Kerre [6] using fuzzy set theory.

Another branch of methods based on reinforcement learning has also tackled the sequential playlist generation scenario. These methods face the so-called exploration-exploitation dilemma, where an agent (e.g., a music recommender) must choose at each time step whether to exploit current knowledge (e.g., recommend the user's favorite song) or explore new knowledge (e.g., elicit the user's feedback on other songs). For instance, Wang et al. [38] proposed a Bayesian learning approach to balance exploration and exploitation for interactive and personalized music recommendation by leveraging users' explicit feedback signals. Relatedly, Liebman et al. [25] used explicit feedback to learn users' preferences over both individual songs as well as song transitions based on a tree-search heuristic. In common, these approaches relied on explicit user feedback, which requires an intrusive interaction mechanism for rating elicitation [18]. Although the use of implicit feedback, which is far more abundant, is well explored in offline music recommendation, it has been rarely mentioned in the online scenario [36]. One notable exception is the work of King and Imbrasaitė [20], who proposed a reinforcement learning approach to walk the space of song clusters using audio-based representations. Instead, here we focus on individual songs as the unit of recommendation.

2.2 Contextual Bandits

In reinforcement learning, there is a subclass of problems called multi-armed bandits (MAB). MAB is a probability theory problem modeled as follows: one player in front of k slot machines should decide how many times and in which order to play the machines. Each slot machine offers a random reward, also named as payoff, from an unknown probability distribution. Therefore, the player's goal is to maximize the accumulated payoff from a sequence of moves by estimating the reward distribution of each machine [23]. Several real-world problems could be modeled in a similar fashion, including the sequential music recommendation problem. More specifically, our work fits a particular version of the MAB problem, known as the contextual bandit problem. In this version, the player can also observe contextual information to decide which machine to bet on and further estimate its reward distribution. In a recommendation scenario, context is typically defined as any side information (like features) about users or items [22].

Li et al. [23] modeled news recommendation as a contextual bandit problem. Their proposed approach, denoted LinUCB, leveraged user clicks to optimize the selection of top news stories. In our experiments, we adapt LinUCB for sequential music recommendation based on implicit play and skip signals and use it as a baseline. However, one potential limitation of this approach is the fact that it models items (in our setting, songs) as arms whose reward must be estimated. Given the large song catalogs available for recommendation in modern music streaming platforms, exploration may be hindered. As an alternative, online learning to rank approaches have recently been introduced that represent items in a lower-dimensional feature space and model "dueling" ranking

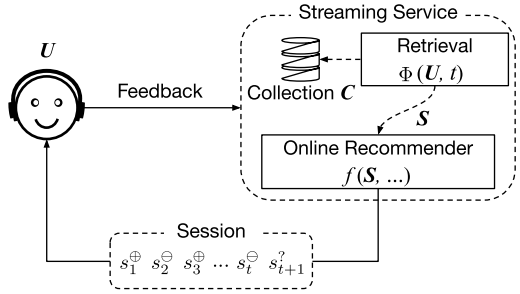

Figure 1: The sequential music recommendation scenario addressed by CDB.

models defined on this space as arms to be explored [28]. To estimate model rewards, rankings produced by different models are interleaved and presented to the user for feedback, typically in the form of clicks. For instance, Yue and Joachims [40] proposed Dueling Bandit Gradient Descent (DBGD), an online learning to rank approach capable of comparing two dueling models at a time. DBGD was later extended into Multileave Gradient Descent (MGD) by Schuth et al. [34], enabling the comparison of multiple dueling models at a time and hence more efficient exploration. Nevertheless, the mode of interaction underlying these approaches is unsuited for sequential music recommendation, where only one song (as opposed to an entire ranking) is presented to the user for feedback at each time step. To overcome this limitation, we introduce a novel online learning to rank approach for sequential music recommendation, capable of estimating the rewards of multiple dueling models based on a single feedback per user interaction.

3 COUNTERFACTUAL DUELING BANDITS

Inspired by state-of-the-art contextual bandit approaches for online learning to rank [34, 40], we also explore multiple dueling directions to learn better recommendation models. However, our proposed Counterfactual Dueling Bandits (CDB) approach employs a slightly different reward model that is better suited to scarce-feedback problems such as sequential music recommendation. This reward model only requires a single implicit feedback signal from the user at each interaction: either a positive one (play) or a negative one (skip). This modification allows us to perform multiple duels between models without the necessity of interleaved rankings of songs.

The music recommendation scenario we focus on in this work is illustrated in Figure 1. A user consumes a music streaming service through a listening session, where the songs are sequentially recommended by the service. Each song receives from the user either a positive or a negative implicit feedback. For instance, in this work we model the feedback as follows: either the song is entirely listened by the user (a positive feedback) or it is skipped before its ending (a negative feedback). In particular, the first song s_1 is provided by the user to the recommender system and it is assumed that it would have a positive feedback from the user. This task formulation is implemented in popular music streaming services such as Spotify's "Start a Radio" functionality [30]. Specifically, we are interested in the online recommendation task of this scenario, i.e., in learning a function f that recommends a single song at a time given a set of

CDB $(u, \mathbf{s_1}, m, \delta, \gamma)$

1 $\mathbf{S} \leftarrow \Phi(u, 1)$ ▷ initial set of candidate songs
2 $\mathbf{w} \leftarrow init()$
3 $r \leftarrow 1 \,/\, \langle \mathbf{w}, \mathbf{S} \rangle^{-1}(\mathbf{s_1})$ ▷ set the first model reward
4 **for** $t \leftarrow 1$ **to** ∞ **do**
5 **for** $i \leftarrow 1$ **to** m **do** ▷ perform m duels
6 $\mathbf{v} \leftarrow rand\_unit()$
7 $\mathbf{w_c} \leftarrow \mathbf{w} + \delta\mathbf{v}$ ▷ generate a candidate model
8 $r_c \leftarrow 1 \,/\, \langle \mathbf{w_c}, \mathbf{S} \rangle^{-1}(\mathbf{s_t})$ ▷ calculate model reward
9 **if** $r_c > r$ **then** ▷ perform duel
10 $\mathbf{w} \leftarrow \mathbf{w} + \gamma r_c \mathbf{v}$ ▷ update current model
11 **end if**
12 **end for**
13 $\mathbf{S} \leftarrow \Phi(u, t{+}1)$
14 $\mathbf{s_{t+1}} \leftarrow \arg\max_{\mathbf{s} \in \mathbf{S}} \langle \mathbf{w}, \mathbf{s} \rangle$ ▷ recommend song $\mathbf{s_{t+1}}$
15 $p \leftarrow payoff(u, \mathbf{s_{t+1}})$ ▷ receive feedback on $\mathbf{s_{t+1}}$
16 $r \leftarrow (p > 0)\ ?\ 1\ :\ 1/(1/r + 1)$ ▷ update current reward
17 $\gamma \leftarrow (p > 0)\ ?\ |\gamma|\ :\ -|\gamma|$ ▷ set exploitation direction
18 **end for**

Algorithm 1: The CDB algorithm.

candidate songs S. This set could be provided, for instance, by some standard retrieval function Φ based on the historical preferences of the target user. Several approaches to the retrieval function Φ have been proposed [10] and are beyond the scope of this work.

The pseudo-code of CDB is presented in Algorithm 1. As input, CDB receives a user u, an initial song s_1 provided by the user as a seed for subsequent recommendations, the number of duels m to be performed after each user interaction, the exploration factor δ, and the exploitation factor γ. Firstly, we rely on a retrieval function Φ (line 1) to provide a set of candidate songs for recommendation, e.g., based on the target user's historical feedback. Each candidate song $\mathbf{s}$ is modeled as a d-dimensional feature vector, as described later in Section 4.2. At every step of CDB, we have a current model $\mathbf{w}$ in charge of continuously recommending songs to the user. Following DBGD and MGD, we consider d-dimensional linear models. The initial model $\mathbf{w}$ can be instantiated (line 2) using different heuristics. In our experiments in Section 5.3, we evaluate two of them: a random initialization and an initialization based on a constant value set to all model weights uniformly.

Next, we must set the first reward to model $\mathbf{w}$ (line 3) based on the song s_1 provided by the user. However, in reality, song s_1 may be different from the song s^* that model $\mathbf{w}$ would have chosen to display to the user (e.g., the model's highest scored song). Therefore, the model reward cannot be computed directly. Instead, we propose a simple counterfactual estimation of model reward, aimed to approximate the user's feedback on song s^* should it have been displayed instead of s_1. In particular, the function $\langle \mathbf{w}, \mathbf{S} \rangle^{-1}(\mathbf{s_t})$ returns the position (or rank) of song $\mathbf{s_t}$ in the ranking of candidate songs S, attained with the inner product between $\mathbf{w}$ and S. Precisely, r is the reciprocal rank of song s_1 in the ranking produced by model $\mathbf{w}$ with the candidate songs of S. Throughout the algorithm, the current reward r will be compared with the rewards attained by other candidate models and the results of these comparisons will determine whether or not the current model $\mathbf{w}$ should be updated.

Next, the algorithm starts a continuous loop of interactions with user u (lines 4 to 18), composed by three key parts: model dueling (lines 5 to 12), song recommendation (lines 13 to 14), and feedback gathering (lines 15 to 17). The model dueling consists in m comparisons between each candidate model $\mathbf{w_c}$ and the current model $\mathbf{w}$. Each model $\mathbf{w_c}$ is the result of the vector addition of the original model $\mathbf{w}$ with a random unit vector $\mathbf{v}$. The goal of using this sum is to explore different directions (or perturbations) from $\mathbf{w}$. To control the *exploration level* on generating candidate models, we use the factor $\delta \in [0,1]$. The higher the value of δ, the greater the distance between each candidate model $\mathbf{w_c}$ and the current model $\mathbf{w}$.

Each $\mathbf{w_c}$ is associated to a reward r_c. This reward is calculated in the same way as the first reward (line 3) previously discussed. If r_c is greater than the current model reward r, we have found a possibly better direction to move $\mathbf{w}$ in and hence $\mathbf{w}$ should be updated. The update of the current model $\mathbf{w}$ (line 10) depends on factor γ, reward r_c and $\mathbf{v}$, the random direction that led us to a reward greater than we currently have. The impact of this update is proportional to the reward value r_c. The factor $\gamma \in [0,1]$ controls the relative importance between the update vector $r_c\mathbf{v}$ and the original model $\mathbf{w}$. The higher the value of γ, the more weight the update vector will have in the new model $\mathbf{w}$. Conversely, the less weight the original model $\mathbf{w}$ will have in its updated version. Thus, the factor γ controls the *exploitation level* of the recommendation: the lower the value of γ, the more important the model $\mathbf{w}$ will be in the next recommendation, i.e., the next recommended song will tend to be more similar to the ones that $\mathbf{w}$ would recommend by itself—for instance, a song for which the user had already given one or more positive feedbacks in previous interactions.

After the model dueling phase, model $\mathbf{w}$ has possibly undergone successive updates and will be used to make a song recommendation. Again, we use the retrieval function Φ to obtain a new set of candidate songs $\mathbf{S}$ (line 13). The song $\mathbf{s_{t+1}} \in \mathbf{S}$ that CDB will recommend to the user is the one that has the maximum score according to the current model $\mathbf{w}$ (attained by the inner product of $\mathbf{w}$ and $\mathbf{S}$). Lastly, in the feedback gathering phase, the system receives the user feedback on the recommended song $\mathbf{s_{t+1}}$ and updates the current reward r. The payoff $p \in \{0,1\}$ indicates whether the feedback was positive (play, $p = 1$) or negative (skip, $p = 0$). When the song $\mathbf{s_{t+1}}$ receives a play, the reward r is assigned the maximum possible value ($r = 1$), which prevents the current model $\mathbf{w}$ from being updated in the dueling phase, since a candidate model will never achieve a reward r_c better than r (line 9). The rationale supporting this strategy is that a model that recommends a song that receives a positive feedback is an effective model and must be kept the same until the next interaction with the user. In our initial experiments, we also allowed the update of the model $\mathbf{w}$ in case of a positive feedback (assigning a different value for its reward r) but we found that the effectiveness of the recommendation was equivalent when we used instead the negative feedbacks only. Other strategies to leverage the user's positive feedback are open directions for research and can be easily integrated with CDB.

When the song $\mathbf{s_{t+1}}$ instead receives a skip from the user, CDB starts a search for models better than the current $\mathbf{w}$. Firstly, it decreases the current reward r (line 16), since model $\mathbf{w}$ recommended a song whose feedback was negative. The decreasing level of the reward r follows a decay policy based on the reciprocal rank measure. After each consecutive skip, the reward is decreased as if the first relevant item (in the computation of reciprocal rank) was placed one position lower. Hence, the rate of decay of rewards is associated to the occurrence of consecutive skips. We use this decay policy to highly penalize the model $\mathbf{w}$ for initial errors (i.e., skips), increasing the probability of $\mathbf{w}$ being updated in the next dueling phase. In this perspective, the current reward r can be interpreted as a threshold value for the update of the current model $\mathbf{w}$: the smaller the reward r, the greater the probability of $\mathbf{w}$ being updated. Moreover, we update the exploitation factor γ of CDB to either its absolute value—in case of a play—or the inverse of its absolute value (necessarily negative)—in case of a skip. This will determine whether model $\mathbf{w}$ will move closer to (vector addition) or away from (vector difference) the candidate models during its update (line 10). Figure 2 illustrates this update after a skip.

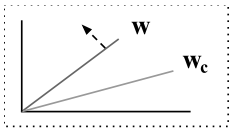

Figure 2: The current model w moves away from candidate model $\mathbf{w}_c$ after the latter highly ranks a skipped song.

4 EXPERIMENTAL SETUP

In this section, we describe the setup that supports our investigations in Section 5 to address the following research questions:

- *Q1.* How fast can CDB learn?
- *Q2.* How effective is CDB at convergence?
- *Q3.* How is CDB impacted by model initialization?
- *Q4.* How is CDB impacted by the number of duels?

4.1 Recommendation Dataset

Our experiments use the Last.fm 1K dataset [8], comprising listening events of Last.fm[1] users between July 2005 and May 2009. Each event contains the user ID, event timestamp, artist ID, song ID, and song title. We enrich the original dataset by joining it with two external sources of song information: Spotify,[2] for audio attributes, and MusicBrainz,[3] for content and social attributes. To respect the sequential nature of the task and avoid peeking into future events during training, we maintain the temporal order of the data before splitting it for evaluation. Moreover, to avoid seasonal effects, we consider multiple such splits, each covering six months' worth of interactions. For consistency, we filter out splits with less than 500 distinct users and less than 500 monthly interactions per user on average. After this filtering, the first, second, and last splits are discarded. The final dataset[4] after enrichment and filtering comprises 776 unique users, 656,483 unique songs, and 11,222,409 listening events, spread over five 6-month splits, as illustrated in Figure 3. We evaluate all models in a sliding window setup: for every two

[1] http://www.last.fm
[2] https://developer.spotify.com/web-api/get-several-audio-features
[3] http://www.musicbrainz.org/
[4] Available upon request.

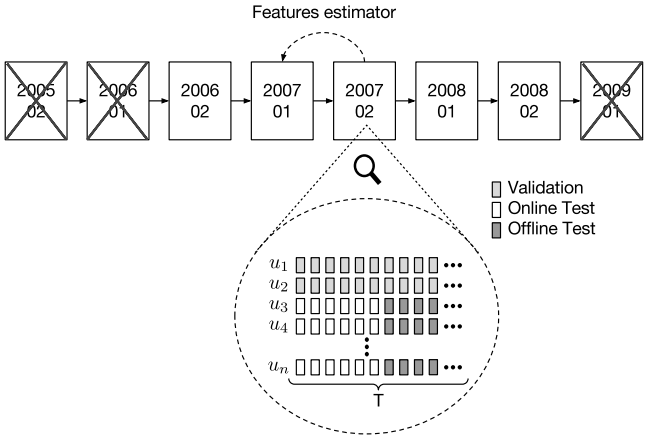

Figure 3: Temporal data partitioning into five 6-month splits. Consecutive splits are used in pairs, with the first split used as background for feature extraction and the second split divided into validation, online test, and offline test.

consecutive splits in Figure 3, the first is used as background data for feature extraction while the second is used for online learning and evaluation. These two processes are discussed next.

4.2 Feature Extraction

Our online learning approach represents each candidate song in a lower dimensional feature space for efficient exploration of dueling recommendation models. In our experiments, we consider both collaborative as well as content-based features. Collaborative features rely on the implicit feedback data available in the background split immediately preceding each online learning split. Following Volkovs and Yu [37], we represent users and songs in a 100-dimensional latent space obtained via low-rank matrix factorization of the background data. Artists are then represented as an average of the vector representation of their songs. Content-based features rely on song representations computed from external sources. These include a sparse 176,564-dimensional textual representation using data such as song title and artist name and a sparse 2,638-dimensional social representation using tags, both obtained from MusicBrainz, and a dense 24-dimensional audio representation obtained from the Spotify API, comprising numeric attributes such as acousticness, danceability, intensity, key, tempo, etc.

Given the collaborative, textual, social, and audio representations produced for each object (a user,[5] song, or artist) in the background split, during online learning, at each time t, we represent each user-candidate song pair $\langle u, s_t \rangle$ as a 22-dimensional feature vector. Of these, a total of 16 session-based features are produced using each of the aforementioned representations by encoding the similarity of song s_t (respectively artist a_t) with respect to: (i) the first song (artist) listened to in the current session; (ii) the last song (artist) listened to in the current session. In addition, to promote discovery, inspired by Xing et al. [39], we encode the time since s_t was last played in the session as a feature. Moreover, using collaborative representations, we further compute another 4 personalized preference features based on the similarity of song s_t (respectively artist a_t) with respect to: (i) the target user u; (ii) u's most listened song (artist). Lastly, as a non-personalized feature, we include the popularity of s_t in the background split.

[5]For users, only a collaborative representation is produced.

4.3 Evaluation Procedure

We tackle a dynamic playlist generation scenario where songs are recommended sequentially, one at a time. Following standard practice [7], we assume a negative feedback (i.e., a skip) if the song is interrupted before 50% of its execution or a positive feedback (i.e., a play) otherwise. In practice, such feedback is only available for the songs actually consumed by the users in our dataset. Since we have no control over the mechanisms that triggered this feedback in the first place, we cannot guarantee its unbiasedness [24]. Instead, to ensure our estimates are reliable, we follow Gentile et al. [13] and simulate randomly logged feedback. In particular, at each time t, we retain the song s_t actually consumed by user u with payoff p_t (1 = "play", 0 = "skip"). In addition, we include 99 extra songs drawn uniformly at random from the set of 1,000 most similar songs to the first song in the user's session, given these songs' previously computed collaborative representations as described in Section 4.2. Hence, we turn the original problem into a ranking problem: at each time t, the recommendation algorithm must select a single song from the list of 100 candidates. In contrast to sampling from the entire song catalog, this session-constrained negative sampling provides for an arguably more challenging ranking task, making it harder to distinguish the consumed song from the non-consumed ones. To ensure the robustness of our findings, sampling is performed six times, with results averaged across executions.

Following this setup, we further divide the data in each split for different purposes, as illustrated in Figure 3. In particular, 10% of the users in a split are used as validation data for tuning the hyperparameters of CDB and its baselines (see Section 4.4) via grid-search [3]. The remaining 90% of the users in the split are used for testing under two complementary schemes, further illustrated in Figure 4: online test and offline test. The *online test* aims to assess the effectiveness of an online music recommender based on the payoff it accumulates while learning, capturing the impact its exploration-exploitation trade-off has on the user's listening experience. To this end, we define the cumulative online payoff for a user u up to time t as:

$$OnlinePayoff@t = \sum_{i=1}^{t} \lambda^i p_i, \tag{1}$$

where p_i is the (instantaneous) payoff at time i and $\lambda = 0.995$ is a discount factor aimed to give credit to systems that are able to produce correct recommendations earlier. Such a discount factor is commonly employed in infinite horizon online evaluation scenarios where $t \to \infty$ to penalize trivial attempts of exhaustive exploration before any exploitation is performed [17]. In our experiments in Section 5, we report *OnlinePayoff@t* averaged across test users in each data split at discrete target times $t \in \{1, \cdots, \kappa_1\}$, $\kappa_1 = 400$.

In addition to the online test, we set aside interaction data for each test user to perform an *offline test*. This complementary test aims to assess the convergence of the recommendation models

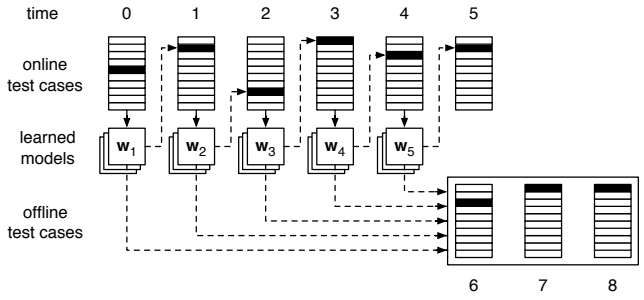

Figure 4: Online and offline evaluation schemes for one user. At each time i, a model $\mathbf{w}_i$ is learned online after a series of duels leveraging (solid arrow) the user's most recent feedback and produces (dashed arrow) a ranked list of songs. For evaluation purposes, a model is rewarded only for placing the user's actually played song (highlighted in black) at the top of the ranking. Models learned online are evaluated both online on $\kappa_1 = 5$ test cases (cases 1–5) during the user's listening session and offline on $\kappa_2 = 3$ held-out test cases (6–8).

learned online, however using a fixed, held-out set. More specifically, the offline test set for user u comprises $\kappa_2 = 100$ test cases chronologically placed after the first $\kappa_1 = 400$ test cases included in the user's online test. Formally, given a model learned online at time t, we compute its offline payoff as:

$$OfflinePayoff@t = \sum_{i=\kappa_1+1}^{\kappa_1+\kappa_2} p_i^{(t)}, \tag{2}$$

where $p_i^{(t)}$ is the (instantaneous) payoff obtained by a model learned (online) up to time t for the (offline) test case at time i. Similarly to the *OnlinePayoff*@t, we report *OfflinePayoff*@t averaged across test users in each split at discrete times $t \in \{1, \cdots, \kappa_1\}$, $\kappa_1 = 400$.

4.4 Experimental Baselines

As a strong online learning baseline, we adapt LinUCB [23], a state-of-the-art k-armed contextual bandit approach, originally conceived to learn users' preferences for a fixed set of news stories given their click feedback. In our adaptation, LinUCB leverages implicit feedback (play or skip) to learn preferences for a fixed set of songs at each time t. Similar to CDB, LinUCB leverages feature-based representations as context for selecting an arm at each time t. However, while LinUCB treats candidate songs as arms, our approach treats candidate recommendation models as arms, selecting the best model at time t via multiple duels. Moreover, while LinUCB was originally proposed for non-personalized recommendation, our approach learns recommendation models tailored to the preferences of individual users. Therefore, to assess the effect of personalization, we produce two variants of LinUCB: (i) LinUCB-U, initialized and updated independently for each user and split, i.e., a personalized recommender; and (ii) LinUCB-G, initialized only once per split and continuously updated on all users, storing what has already been learned with previous users, in a non-personalized fashion. To avoid network effects, users are randomized during test.

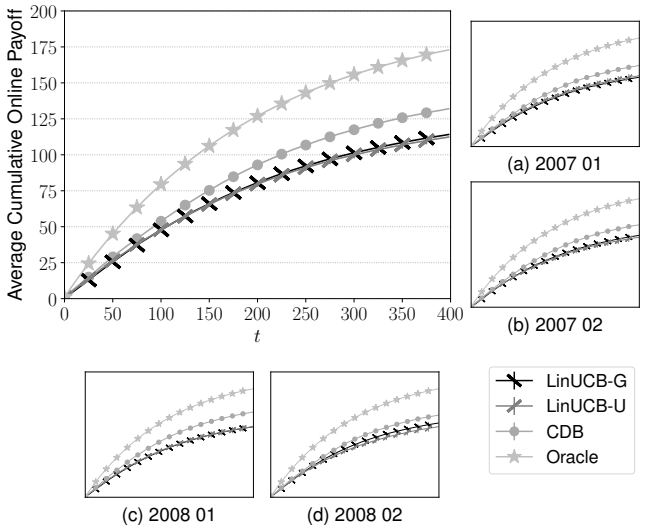

Figure 5: Online test results (average cumulative online payoff) of LinUCB-G, LinUCB-U, CDB, and the Oracle after t interactions with the user.

5 EXPERIMENTAL RESULTS

In this section, we validate our proposed online learning to rank approach for sequential music recommendation by addressing the four research questions stated in Section 4. All results are given in terms of average cumulative online (Equation (1)) and offline (Equation (2)) payoffs. Significance with respect to the LinUCB baselines is verified using a paired t-test with $\alpha = 0.01$.

5.1 Learning Speed

Our first set of experiments aims to answer *Q1*: How fast can CDB learn? To this end, we compare CDB with the two variants of LinUCB described in Section 4.4 and also with an Oracle recommender, which recommends only the correct song to the user at each time step, and hence provides an upper bound for the effectiveness of the other recommenders. Figure 5 shows the average cumulative online payoff across users at each time step t (Equation (1)), for each of the four splits (smaller plots) as well for the average of the four splits (larger plot). The curved shape of the average payoffs are caused by the discount factor λ, which produces marginal gains as the user progresses through the listening session.

From Figure 5, we observe that, for every time step t, CDB is at least as effective as its baselines. In particular, for $t \geq 150$, CDB has significantly better results than both LinUCB variants. From another perspective, CDB is able to achieve each average payoff p_t on the y-axis with fewer user interactions compared to its baselines. These results are further corroborated by those in the top half of Table 1, which shows the average cumulative online payoff at the end of the users' sessions ($t = 400$) in each split. From the table, we note that LinUCB-U is slightly outperformed by its non-personalized counterpart LinUCB-G, which suggests that the scarcity of feedback in personalized settings may hinder the ability of LinUCB to perform exploration in a high-dimensional space of

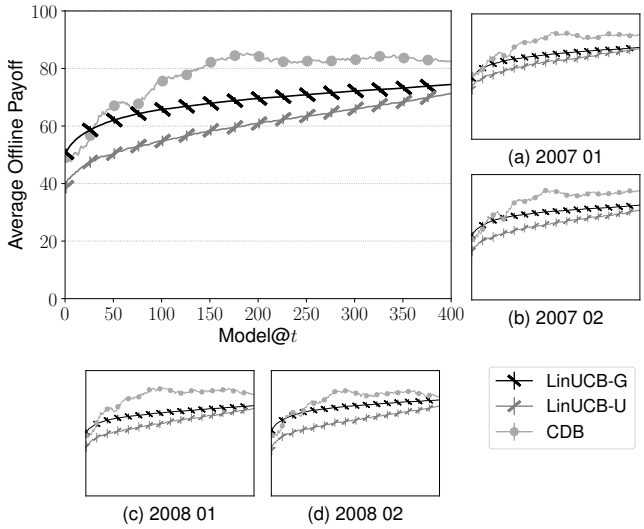

Figure 6: Offline test results (avg. offline payoff) of LinUCB-G/U, and CDB, after t interactions with the user.

songs. In contrast, CDB explores multiple candidate recommendation models in a lower dimensional feature space at each user interaction, which results in significant improvements compared to both LinUCB variants. Recalling *Q1*, these results attest the ability of CDB to learn fast, which is a key trait for an improved user experience in live online recommendation deployments.

Table 1: Online and offline results (average cumulative online payoff and average offline payoff, respectively) of CDB and LinUCB at time step $t = 400$.

	Algorithm	2007 01	2007 02	2008 01	2008 02
Online	LinUCB-G	$0.28_{\pm 0.01}$	$0.29_{\pm 0.01}$	$0.28_{\pm 0.01}$	$0.30_{\pm 0.01}$
	LinUCB-U	$0.28_{\pm 0.01}$	$0.28_{\pm 0.01}$	$0.28_{\pm 0.01}$	$0.28_{\pm 0.01}$
	CDB	$\mathbf{0.32}^{\blacktriangle\blacktriangle}_{\pm 0.01}$	$\mathbf{0.33}^{\blacktriangle\blacktriangle}_{\pm 0.01}$	$\mathbf{0.34}^{\blacktriangle\blacktriangle}_{\pm 0.02}$	$\mathbf{0.33}^{\blacktriangle\blacktriangle}_{\pm 0.01}$
Offline	LinUCB-G	$66.75_{\pm 0.53}$	$69.40_{\pm 0.50}$	$65.93_{\pm 0.45}$	$70.58_{\pm 0.51}$
	LinUCB-U	$61.47_{\pm 0.71}$	$60.22_{\pm 0.74}$	$58.36_{\pm 0.70}$	$59.87_{\pm 0.74}$
	CDB	$\mathbf{76.79}^{\blacktriangle\blacktriangle}_{\pm 0.91}$	$\mathbf{78.94}^{\blacktriangle\blacktriangle}_{\pm 1.00}$	$\mathbf{78.54}^{\blacktriangle\blacktriangle}_{\pm 0.92}$	$\mathbf{76.46}^{\blacktriangle\blacktriangle}_{\pm 0.92}$

5.2 Learning Effectiveness

The results in the previous section demonstrated the ability of our proposed CDB to learn fast. However, the online test results only tell part of the story, as CDB might be only exploring, and a trade-off between exploitation and exploration can yield a better model at the end. To investigate this further, we address *Q2*: How effective is CDB at convergence? To this end, we present offline test results in Figure 6 in terms of the average offline payoff across users attained by each model learned online up until time step t. Once again, smaller plots show results per test split, whereas the larger plot shows global results across all test splits.

From Figure 6, we observe that, during the initial interactions, the performance of CDB models is slightly lower than the models produced by LinUCB-G and slightly higher than the models produced by LinUCB-U. However, for $t \geq 100$, CDB consistently outperforms both LinUCB variants. The ups and downs of the average offline payoff attained by CDB can be explained by the intense exploration of several candidate recommendation models performed during its execution. Nevertheless, despite these initial fluctuations, CDB converges to a significantly higher performance level than LinUCB. This observation is further confirmed by the results at the bottom half of Table 1, which shows the average offline payoff obtained by CDB and LinUCB at the end of the learning process ($t = 400$). In particular, CDB substantially outperforms both LinUCB variants. Recalling *Q2*, these results attest the ability of CDB to learn increasingly improved models along the user's listening session, with a highly effective performance at convergence.

5.3 Impact of the Initialization Strategy

The previous sections demonstrated the ability of CDB to learn faster and better compared to a state-of-the-art contextual bandit approach. To further understand the role of different components of CDB on its effectiveness, we address *Q3*: How is CDB impacted by model initialization? To answer this question, we evaluate two alternative initialization strategies: (i) CDB *Fixed*, which initializes all weights of the model $\mathbf{w}$ with a constant γ tuned on the validation data in each split; and (ii) CDB *Random*, which initializes $\mathbf{w}$ randomly. Figure 7 shows the average performance of the two proposed configurations in the online tests. The results of CDB Fixed are slightly better than random initialization for each time step t in all splits. However, the results also show that, despite starting from a completely randomly generated model, CDB Random is able to learn models almost as effective as the ones produced by CDB Fixed. To complement this investigation, Figure 7 also shows the average offline payoff attained with each initialization strategy over time. Indeed, the first models produced by CDB Random present lower effectiveness than CDB Fixed in all splits. Nevertheless, after just a few user interactions (for $t \geq 50$), CDB Random is able to learn significantly better models than its initial ones and begins to produce recommendations with similar or even greater effectiveness than the recommendations from CDB Fixed. Recalling *Q3*, these observations suggest that setting aside validation data for initializing model weights may be unnecessary, which could facilitate the deployment of CDB in other sequential recommendation domains.

5.4 Impact of the Number of Duels

To complement the analysis in the previous section, we address *Q4*: How is CDB impacted by the number of duels? The expectation is that a larger number of duels m will accelerate the learning process and consequently improve the effectiveness of the models produced after each interaction with the user. On the other hand, as we increase the number of duels, we also increase the time consumption of CDB. In particular, the time complexity of CDB can be expressed by $O(m \times d \times k)$, where m is the number of duels, d is the dimensionality of the underlying feature space (22 in our experiments), and k is the cardinality of the set of candidate songs (100 in our experiments). Therefore, establishing an optimal setting

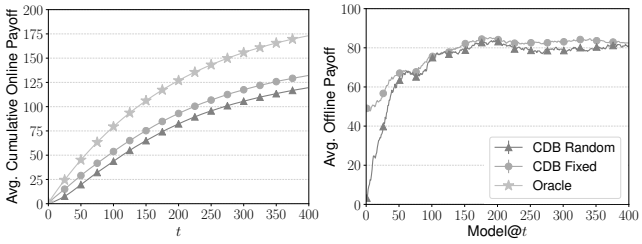

Figure 7: Online and offline test results (avg. cumulative online payoff and avg. offline payoff, respectively) of CDB Random/Fixed and Oracle after t user interactions.

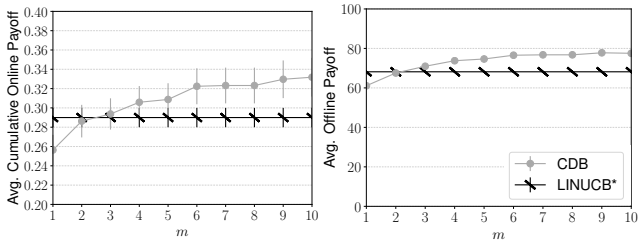

Figure 8: Online and offline test results (avg. cumulative online payoff and avg. offline payoff, respectively) of CDB by increasing the number m of duels.

for m is crucial for balancing the effectiveness and efficiency of the produced recommendations. To this end, we tested values of $m \in \{1, \cdots, 10\}$, tuning the remaining hyperparameters of CDB, namely γ and δ, on the validation data. Note that $m = 1$ equates to performing a single duel at each time step, in a similar fashion to the standard DBGD algorithm [40].

Figure 8 shows the average cumulative online payoff attained by CDB when we increase the value of the parameter m. From the figure, we observe that the use of multiple duels instead of a single duel ($m = 1$) indeed improves the effectiveness of CDB. Also, as expected, the results show a positive impact on effectiveness as the parameter m increases. However, we also note that the gain in performance tails off for higher values of m (for instance, for $m > 6$) compared to the gains observed for the lower range of m values. For instance, the improvement from $m = 1$ to $m = 2$ is the highest in all the splits. Moreover, for $m \geq 5$, CDB outperforms the best LinUCB variant in all splits.

We repeated the same testing methodology in the offline test setting, as shown in Figure 8. Again, the results show a positive impact on effectiveness as we increase the value of m, but with decreasing gains in performance as m approaches the highest tested values. The improvement of using $m = 7$, for instance, is around 30% when compared to CDB using only one duel per feedback ($m = 1$). Besides, for $m \geq 3$, CDB produces better results than the best LinUCB variant in all splits. Recalling *Q4*, these results attest the benefit of making the most of every single feedback provided by the user and demonstrate the effectiveness of comparing multiple recommendation models via duels to this end.

6 CONCLUSIONS

In this paper, we proposed CDB, a novel online learning to rank approach for sequential music recommendation. CDB leverages implicit feedback signals (plays and skips) to dynamically drive the user's listening session. To enable the exploration of multiple candidate recommendation models when only one feedback is received at each point in time, we devise a counterfactual estimation of the effectiveness of each model based on how it would have ranked the song actually consumed by the user at that time. To the best of our knowledge, our approach is the first to perform online learning to rank in a personalized and feedback-scarce scenario. We thoroughly evaluated our approach in contrast to a state-of-the-art contextual bandit approach in a sequential music recommendation scenario using enriched listening sessions from Last.fm. The results showed that CDB learns more effective recommendation models while demanding fewer user interactions. Further analyses demonstrated its robustness to initialization conditions and the benefit of exploring multiple directions at each time step.

In the future, we plan to investigate the suitability of CDB for learning personalized recommendations in other domains, such as news recommendation. We also plan to investigate alternative counterfactual estimation approaches for determining duel winners under scarce feedback conditions, as well as alternatives to complement the available feedback with weak supervision, which could enable the learning of non-linear, data-hungry recommendation models. Lastly, we plan to further assess the effectiveness of our approach through a live experiment with real users.

ACKNOWLEDGMENTS

This work was partially funded by project MASWeb (FAPEMIG/PRONEX APQ-01400-14) and by the authors' individual grants from CNPq and FAPEMIG.

REFERENCES

[1] Claudio Baccigalupo and Enric Plaza. 2006. Case-based Sequential Ordering of Songs for Playlist Recommendation. In *ECCBR*. 286–300.
[2] Linas Baltrunas and Xavier Amatriain. 2009. Towards Time-Dependant Recommendation Based on Implicit Feedback. In *Workshop on CARS*.
[3] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Proc. of NIPS*. 2546–2554.
[4] Dmitry Bogdanov, Joan Serrà, Nicolas Wack, Perfecto Herrera, and Xavier Serra. 2011. Unifying low-level and high-level music similarity measures. *IEEE Transactions on Multimedia* 13, 4 (2011), 687–701.
[5] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *ACM Comput. Surv.* 47, 2 (2014), 26:1–26:35.
[6] Klaas Bosteels and Etienne E. Kerre. 2009. A Fuzzy Framework for Defining Dynamic Playlist Generation Heuristics. *Fuzzy Sets and Systems* 160, 23 (2009).
[7] Klaas Bosteels, Elias Pampalk, and Etienne E. Kerre. 2009. Evaluating and Analysing Dynamic Playlist Generation Heuristics using Radio Logs and Fuzzy Set Theory. In *Proc. of ISMIR*. 351–356.
[8] Oscar Celma. 2010. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space* (1st ed.).
[9] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist Prediction via Metric Embedding. In *Proc. of KDD*. 714–722.
[10] Zhiyong Cheng, Shen Jialie, and Steven CH Hoi. 2016. On Effective Personalized Music Retrieval by Exploring Online User Behaviors. In *Proc. of SIGIR*. 125–134.
[11] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan S Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation.. In *Proc. of IJCAI*, Vol. 17. 3654–3660.
[12] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. 2008. Playlist Generation Using Start and End Songs. In *Proc. of ISMIR*. 173–178.
[13] Claudio Gentile, Shuai Li, Purushottam Kar, Alexandros Karatzoglou, Giovanni Zappella, and Evans Etrue. 2017. On Context-Dependent Clustering of Bandits. In *Proc. of ICML*. 1253–1262.

[14] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-Aware Music Recommendation Based on Latent Topic Sequential Patterns. In *Proc. of RecSys*. 131–138.
[15] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2015. Adapting to User Preference Changes in Interactive Recommendation. In *Proc. of ICAI*. 4268–4274.
[16] David B Hauver and James C French. 2001. Flycasting: Using Collaborative Filtering to Generate a Playlist for Online Radio. In *Proc. of IEEE Web Delivering of Music*. 123–130.
[17] Katja Hofmann. 2013. *Fast and Reliable Online Learning to Rank for Information Retrieval*. Ph.D. Dissertation. University of Amsterdam.
[18] Gawesh Jawaheer, Matin Szomszor, and Patty Kostkova. 2010. Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service. In *Proc. of HetRec*. 47–51.
[19] Diane Kelly and Jaime Teevan. 2003. Implicit Feedback for Inferring User Preference: A Bibliography. In *Proc. of SIGIR*. 18–28.
[20] James King and Vaiva Imbrasaitė. 2015. Generating Music Playlists With hierarchical Clustering and Q-Learning. In *Advances in Information Retrieval*. 315–326.
[21] Paul B Lamere. 2012. I've got 10 Million Songs in my Pocket: Now what?. In *Proc. of RecSys*. 207–208.
[22] John Langford and Tong Zhang. 2008. The Epoch-Greedy Algorithm for Multi-Armed Bandits with Side Information. In *Proc. of NIPS*. 817–824.
[23] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proc. of WWW*. 661–670.
[24] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-Bandit-Based News Article Recommendation Algorithms. In *Proc. of WSDM*. 297–306.
[25] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. 2015. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proc. of AAMAS*. 591–599.
[26] Beth Logan. 2002. Content-Based Playlist Generation: Exploratory Experiments. In *Proc. of ISMIR*.
[27] François Maillet, Douglas Eck, Guillaume Desjardins, Paul Lamere, et al. 2009. Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists.. In *Proc. of ISMIR*. 345–350.
[28] Daan Odijk and Anne Schuth. 2017. Online Learning to Rank for Recommender Systems. In *Proc. of RecSys (RecSys '17)*. 348–348.
[29] Elias Pampalk, Tim Pohle, and Gerhard Widmer. 2005. Dynamic Playlist Generation Based on Skipping Behavior. In *Proc. of ISMIR*. 634–637.
[30] Snickars Pelle. 2017. More of the Same–On Spotify Radio. *Culture Unbound. Journal of Current Cultural Research* 9, 2 (2017), 184–211.
[31] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4 (2018), 66.
[32] Robert Ragno, Christopher J. C. Burges, and Cormac Herley. 2005. Inferring Similarity between Music Objects with Application to Playlist Generation. In *Proc. of SIGMM*. 73–80.
[33] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Introduction to Recommender Systems Handbook* (2nd ed.). Springer.
[34] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave Gradient Descent for Fast Online Learning to Rank. In *Proc. of WSDM*. 457–466.
[35] Yading Song, Simon Dixon, and Marcus Pearce. 2012. A Survey of Music Recommendation Systems and Future Perspectives. In *Proc. of CMMR*. 395–410.
[36] Andreu Vall. 2015. Listener-Inspired Automated Music Playlist Generation. In *Proc. of RecSys*. 387–390.
[37] Maksims N. Volkovs and Guang Wei Yu. 2015. Effective Latent Models for Binary Feedback in Recommender Systems. In *Proc. of SIGIR*. 313–322.
[38] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. 2014. Exploration in Interactive Personalized Music Recommendation: A Reinforcement Learning Approach. *ACM TOMM* 11, 1 (2014), 1–22.
[39] Zhe Xing, Xinxi Wang, and Ye Wang. 2014. Enhancing Collaborative Filtering Music Recommendation by Balancing Exploration and Exploitation. In *Proc. of ISMIR*. 445–450.
[40] Yisong Yue and Thorsten Joachims. 2009. Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem. In *Proc. of ICML*. 1201–1208.