International Journal of Multimedia Information Retrieval
https://doi.org/10.1007/s13735-019-00169-8

REGULAR PAPER

Order, context and popularity bias in next-song recommendations

Andreu Vall[1] · **Massimo Quadrana**[3] · **Markus Schedl**[1] · **Gerhard Widmer**[1,2]

Received: 9 July 2018 / Revised: 21 February 2019 / Accepted: 7 March 2019

Abstract
The availability of increasingly larger multimedia collections has fostered extensive research in recommender systems. Instead of capturing general user preferences, the task of *next-item recommendation* focuses on revealing specific session preferences encoded in the most recent user interactions. This study focuses on the music domain, particularly on the task of *music playlist continuation*, a paradigmatic case of next-item recommendation. While the accuracy achieved in next-song recommendations is important, in this work we shift our focus toward a deeper understanding of fundamental playlist characteristics, namely the song order, the song context and the song popularity, and their relation to the recommendation of playlist continuations. We also propose an approach to assess the quality of the recommendations that mitigates known problems of off-line experiments for music recommender systems. Our results indicate that knowing a longer song context has a positive impact on next-song recommendations. We find that the long-tailed nature of the playlist datasets makes simple and highly expressive playlist models appear to perform comparably, but further analysis reveals the advantage of using highly expressive models. Finally, our experiments suggest that the song order is not crucial to accurately predict next-song recommendations.

Keywords Music recommender systems · Music playlist continuation · Sequential recommendation · Collaborative filtering · Recurrent neural networks

1 Introduction

Automated music playlist continuation is a specific task in music recommender systems where the user sequentially receives song recommendations, producing a listening experience similar to traditional radio broadcasting. Sequential recommendation scenarios are in fact very natural in the music domain. This is possibly explained by the short time required to listen to a song, which results in listening sessions typically including not one, but several songs.

✉ Andreu Vall
andreu.vall@jku.at

Massimo Quadrana
mquadrana@pandora.com

Markus Schedl
markus.schedl@jku.at

Gerhard Widmer
gerhard.widmer@jku.at

[1] Institute of Computational Perception, Johannes Kepler University Linz, Linz, Austria

[2] Austrian Research Institute for Artificial Intelligence, Vienna, Austria

[3] Pandora Media Inc., Oakland, CA, USA

According to interviews with practitioners and postings to a dedicated playlist-sharing website, Cunningham et al. [8] identified the choice of songs and the song order as important aspects of the playlist curation process. As we review in Sect. 2, some approaches to automated music playlist continuation take into account the current and previous songs in the playlist and the order of the songs in the playlist to recommend the next song. However, to the best of our knowledge, previous works do not explicitly analyze the impact of exploiting this information for next-song recommendations. We refer to the current and previous songs in a playlist as the "song context" available to the recommender system when it predicts the next song. This terminology is borrowed from language models and should not be confused with the incorporation of user's contextual information into the recommender system.

In this work, we compare four well established and widely used playlist models: a popularity-based model, a song-based collaborative filtering (CF) model, a playlist-based CF model and a model based on recurrent neural networks (RNNs). These playlist models are of increasing complexity and, by design, are able to exploit the song context and the song order to different extents. By analyzing and comparing their performance on different playlist continuation off-line exper-

Published online: 03 April 2019

iments, we derive insights regarding the impact that the song context, the song order and the bias toward popular music have on next-song recommendations. For the evaluation of the off-line experiments, we propose to use metrics derived from complete recommendation lists, instead of from the top K positions of recommendation lists. This provides a more complete view on the performance of the playlist models.

The remainder of this paper is organized as follows. Section 2 reviews the related work on automated music playlist continuation. Section 3 introduces the guidelines for the off-line experiments conducted throughout this work. We describe the recommendation task that the playlist models must fulfill and define the metrics employed to assess their performance on the task. Section 4 describes the four playlist models considered. Section 5 presents the datasets of hand-curated music playlists on which we conduct the off-line experiments. Section 6 elaborates on the results of the off-line experiments and is divided into three parts, which discuss the impact of the song context, the popularity bias and the song order on next-song recommendations, respectively. Conclusions are drawn in Sect. 7.

2 Related work

A well-researched approach to automated music playlist continuation relies on the song content. Pairwise song similarities are computed on the basis of features extracted from the audio signal (possibly enriched with social tags and metadata) and used to enforce content-wise smooth transitions [10,18,21,22,25]. Recommendations based on content similarity are expected to yield coherent playlists. However, pure content-based recommendations cannot capture complex relations and, in fact, it does not hold in general that the songs in a playlist should all sound similar [19].

Playlist continuation has also been regarded as a form of collaborative filtering (CF), making the analogy that playlists are equivalent to user listening histories on the basis of which songs should be recommended. Playlist-based nearest-neighbors CF models and factorization-based CF models exploit the full song context when making next-song recommendations [1,4,11]. Song-based nearest-neighbors CF models [29] are not common in the playlist continuation literature. However, Hidasi et al. [12] show in the domains of e-commerce and video streaming that an item-based CF model that predicts the next item on the basis of only the current item can effectively deal with short histories. In general, CF models disregard the song order, but it is worth noting that the model presented by Aizenberg et al. [1] accounts for neighboring songs, and the model introduced by Rendle et al. [27] for on-line shopping is aware of sequential behavior.

The Latent Markov Embedding introduced by Chen et al. [6] models playlists as Markov chains. It projects songs into a Euclidean space such that the distance between two projected songs represents their transition probability. The importance of the direction of song transitions is evaluated by testing a model on actual playlists and on playlists with reversed transitions, yielding comparable performance in both cases. McFee and Lanckriet [23] also treat playlists as Markov chains, modeled as random walks on song hypergraphs, where the edges are derived from multimodal song features, and the weights are learned from hand-curated music playlists. The importance of modeling song transitions is assessed by learning the hypergraph weights again but treating the playlists as a collection of song singletons. When song transitions are ignored, the performance degrades. These works examine the importance of accounting for song transitions and their order, but the Markovian assumption implies that only adjacent songs are considered.

Hariri et al. [11] represent songs by latent topics extracted from song-level social tags. Sequential pattern mining is performed at the topic level, so that given seed songs, a next topic can be predicted. Re-ranking the results of a CF model with the predicted latent topics is found to outperform the plain CF model. This approach considers the ordering but only at the topic level, which is more abstract than the song level.

Hidasi et al. [12] propose for e-commerce and video streaming an approach to sequential recommendation based on the combination of RNNs with ranking-aware loss functions. This approach has gained attention and has been further improved and extended [13,31]. Jannach and Ludewig [14] have applied it to the task of automated music playlist continuation in a study that compares the performance of RNN models and session-based nearest-neighbors models for sequential recommendation. Among other analyses, Jannach and Ludewig question whether the computational complexity of RNN models is justified. Recommendation models based on RNNs consider the full item context in sequences and are also aware of their order.

For a comprehensive survey on automated music playlist continuation, we point the interested reader to Bonnin and Jannach [4] and Ricci et al. [28, chap.13].

We conducted preliminary studies preceding this work analyzing the importance of the song order and the song context in next-song recommendations [32,33]. This paper further extends these works by incorporating a detailed discussion of the proposed evaluation methodology, an additional playlist model (namely a playlist-based CF model), an analysis of the impact of the popularity bias of music collections in next-song recommendations and more conclusive experiments to determine the importance of the song order. We also provide the full configurations and training details for the playlist models.

3 Evaluation methodology

We propose an evaluation methodology based on the ability of playlist models at retrieving withheld playlist continuations. We conduct the next-item recommendation experiment proposed by Hidasi et al. [12] and then propose approaches to interpreting the obtained results and to comparing the performance of different playlist models.

We are aware that off-line evaluation approaches are approximations of the actual recommendation task, and they may not be able to fully estimate the final user satisfaction. However, the aim of this work is to understand the importance of the song context, the song order and the bias toward popular music on next-song recommendations. In this sense, the proposed off-line evaluation methodology serves this purpose well, because it allows the systematic comparison of different playlist models under controlled conditions.

3.1 Next-song recommendation experiment

A collection of music playlists is split into training and test playlists. A trained playlist model is evaluated by repeating the following procedure over all the test playlists, which, for clarity, we describe alongside the example depicted in Fig. 1. We consider a test playlist (e.g., $p = (s_3, s_5, s_2)$). In the first step, we show the model the first song in the playlist (s_3). The model ranks all the songs in the dataset according to their likelihood to be the second song in the playlist. We keep track of the rank attained by the actual second song in the playlist (s_5 attains rank 3). We also keep track of the fact that this is a prediction for a song in second position. In the second step, we show the model the first and the second actual songs in the playlist (s_3, s_5). The model ranks all the songs in the dataset according to their likelihood to be the third song in the playlist. We keep track of the rank attained by the actual third song in the playlist (s_2 attains rank 1), etc. In this way, we progress until the end of the playlist, always keeping track of the rank attained by the actual next song

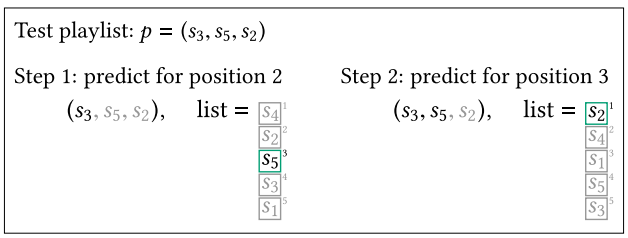


Fig. 1 Illustration of the evaluation methodology. The playlist model is evaluated on the test playlist $p = (s_3, s_5, s_2)$. It progresses through p and ranks all the songs in the dataset according to their likelihood to be the next song. The actual second song, s_5, attains rank 3. The actual third song, s_2, attains rank 1

in the playlist and the position in the playlist for which the prediction is made.

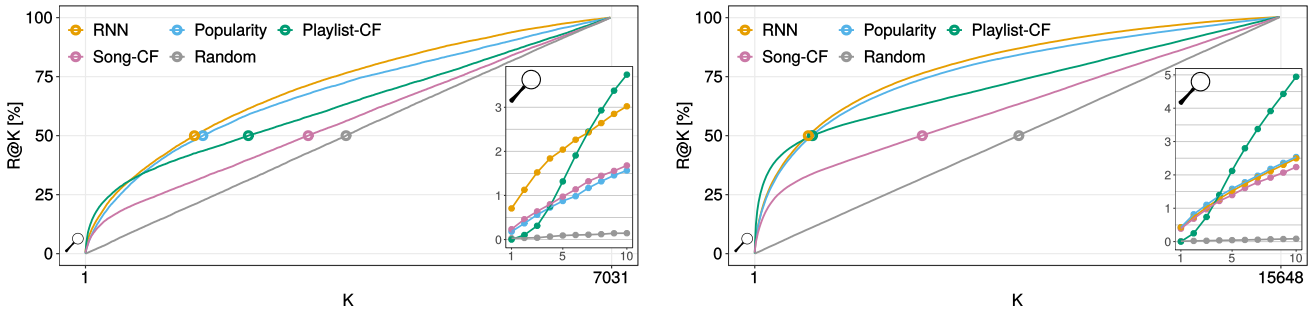
We index the ordered list of next-song candidates from 1 (most likely) until N (least likely), where N is the number of unique songs in the dataset. A good playlist model is expected to rank the actual next song in top positions (rank values close to 1). On the other hand, a poor model would rank the actual next song on bottom positions (large rank values). A random model would, on average, rank the actual next song on positions around $N/2$.

3.2 Assessing the quality of the recommendations

Previous research in automated music playlist continuation has summarized the distribution of attained ranks using metrics derived from the top K positions of the ordered lists of next-song candidates. For example, the recall at K (also named "hit rate" at K) is defined as the proportion of times that the actual next songs in the test playlists attain a rank lower than K [4,11,14,15]. The rationale behind fixing the length K, typically to a small value, is that, in practice, only the top K results are of interest to the end user of the recommender system. We claim that this approach has two important limitations: (1) values of K that are reasonable for on-line systems (with actual end users) are not necessarily reasonable for off-line evaluation (without end users); (2) arbitrarily fixing a value of K provides partial and potentially misleading information.

We first discuss the first limitation. A playlist may be extended by a number of potentially relevant songs, but off-line experiments only accept the exact match to the actual next song. The rank attained by the actual next song can be overly pessimistic, because the ordered list of next-song candidates can actually contain relevant results in better positions [22,24]. Therefore, the results of off-line evaluation approaches need to be understood as approximations of the expected performance of the playlist models. They cannot be interpreted in absolute terms but as a means to compare the relative performance of different models. In particular, values of K meaningful for on-line systems should not be literally transferred to off-line experiments.

We now address the second limitation. Even though the playlist models and the datasets will be presented in Sects. 4 and 5, respectively, we advance here some results for the sake of illustration. Figure 2 shows the recall curves of several playlist models for values of K ranging from 1 to the maximum number of song candidates. If we chose to focus on a fix value of K, this would correspond to only observing a one-point cut of these recall curves, which is very partial information. Furthermore, as we have discussed, choosing a specific value of K can become arbitrary in off-line experiments, where the user feedback is missing. Finally, the information provided by a fix value of K is potentially mis-



(a) AotM-2011 dataset

(b) 8tracks dataset

Fig. 2 Recall curves for values of K ranging from 1 to the maximum number of songs in each playlist dataset. The circles indicate the length K where each playlist model achieves a recall at K of 50% and correspond to the median rank achieved by each model. The results on the top 10 positions are detailed in the boxes, where dots are superimposed only to remind of the discrete nature of the displayed values (the lines just connect the different recall values)

leading because the recall curves of different playlist models cross each other at different values of K. That is, the best performing playlist model would depend on the chosen value of K.

For these reasons, we propose to assess playlist models by examining their whole lists of ordered next-song candidates, as opposed to focusing only on an arbitrary number of top K positions in the lists. This provides a more complete view of the performance of the playlist models. Even though the complete recall curves displayed in Fig. 2 are informative, we instead propose to directly compare the whole distribution of ranks attained by each playlist model. We report the distribution of attained ranks by means of boxplots that represent the minimum, first quartile, median, third quartile, and maximum rank values (see, e.g., Fig. 3). Alternatively, we report only the median rank value if this facilitates the interpretation of the results (Fig. 5).

4 Playlist models

We describe the four playlist models considered in our experiments. By design, the models are of increasing complexity and are able to exploit the song context and the song order to different extents (Table 1). Hyperparameter tuning, if necessary, is performed on validation playlists withheld from the training playlists.

4.1 Song popularity ("Popularity")

This is a unigram model that computes the popularity of a song s according to its relative frequency in the training playlists, i.e.,

$$pop(s) = \frac{|P_{\text{tr}}(s)|}{|P_{\text{tr}}|}, \tag{1}$$

Table 1 Summary of the playlist models

Playlist model	Context length	Order awareness
Popularity	0	✗
Song-CF	1	✗
Playlist-CF	n	✗
RNN	n	✓

The context length is the number of songs considered by the model to predict the next song (n means all the songs shown to the model). Order awareness indicates if the model regards the order of songs in playlists

where P_{tr} is the set of training playlists, $P_{\text{tr}}(s)$ is the subset of training playlists that contain the song s, and $|\cdot|$ denotes the number of playlists in each set. Given a test playlist, the next-song candidates are ranked by their popularity, disregarding the previous songs and their order. Despite its simplicity, the popularity-based model is a competitive playlist model [4,6].

4.2 Song-based collaborative filtering ("Song-CF")

This is a CF model based on song-to-song similarities. A song s is represented by a binary vector $\mathbf{p}_s$ that indicates the training playlists to which it belongs. The similarity of a pair of songs s, t is computed as the cosine between $\mathbf{p}_s$ and $\mathbf{p}_t$, i.e.,

$$\text{sim}(s,t) = \cos(\mathbf{p}_s, \mathbf{p}_t) = \frac{\mathbf{p}_s \cdot \mathbf{p}_t}{\|\mathbf{p}_s\| \|\mathbf{p}_t\|}.$$

Two songs are similar if they co-occur in training playlists, regardless of the positions they occupy in the playlists. We follow Hidasi et al. [12] and implement the song-based CF model such that next-song candidates are ranked according to their similarity only to the current song in the playlist, ignoring previous songs. This approach is relatively simple, but Hidasi et al. show its competitive performance for sequential recommendation on short sessions.

4.3 Playlist-based collaborative filtering ("Playlist-CF")

This is a CF model based on playlist-to-playlist similarities. A playlist p is represented by a binary vector $\mathbf{s}_p$ indicating the songs that it includes. The similarity of a pair of playlists p, q is computed as the cosine between $\mathbf{s}_p$ and $\mathbf{s}_q$, i.e.,

$$\text{sim}(p, q) = \cos(\mathbf{s}_p, \mathbf{s}_q) = \frac{\mathbf{s}_p \cdot \mathbf{s}_q}{\|\mathbf{s}_p\| \|\mathbf{s}_q\|}. \tag{2}$$

The score assigned to a song s as a candidate to extend a test playlist p is computed as

$$\text{score}(s, p) = \sum_{q \in P_{\text{tr}}(s)} \text{sim}(p, q), \tag{3}$$

where $P_{\text{tr}}(s)$ is the subset of training playlists that contain the song s. This model considers a song to be a suitable continuation for playlist p if it has occurred in training playlists that are similar to p. The similarity of a playlist pair (Eq. 2) and the score assigned to a candidate song to extend a playlist (Eq. 3) depend on the full playlist p, i.e., on the full song context, but they disregard the song order.

Playlist-based CF has proven to be a competitive playlist model [4,11,14,15]. It usually has an additional parameter defining the number of most similar training playlists on which Eq. 3 is calculated. We use all the training playlists because we find that this yields best performance in our experiments (Appendix A.3).

4.4 Recurrent neural networks ("RNN")

Recurrent neural networks are a class of neural network models particularly suited to learn from sequential data. They have a hidden state that accounts for the input at each time step while recurrently incorporating information from previous hidden states. We point the interested reader to Lipton et al. [20] for a review of RNN models.

We adopt the approach and implementation[1] proposed by Hidasi et al. [12], where an RNN model with one layer of gated recurrent units (GRU) [7] is combined with a loss function designed to optimize the ranking of next-item recommendations. The model hyperparameters and architecture are detailed in Appendix A.4.

Given a test playlist, the RNN model considers the full song context and the song order and outputs a vector of song scores used to rank the next-song candidates.

[1] https://github.com/hidasib/GRU4Rec

5 Datasets

We evaluate the four playlist models on two datasets of hand-curated music playlists derived from the on-line playlist-sharing platforms "Art of the Mix"[2] and "8tracks."[3] Both platforms allow music aficionados to publish their playlists on-line. Moreover, the Art of the Mix platform hosted forums and blogs for discussion about playlist curation, as well as social functionalities such as favoriting, or providing direct feedback to a user.[4] The 8tracks platform also provides social functionalities, such as following users, liking, or commenting on specific playlists. Previous works in the automated music playlist continuation literature have chosen to work with collections derived from the Art of the Mix and the 8tracks databases because of their presumably careful curation process [4,11,15,22,23]. As an illustration of the users' engagement, we refer the interested reader to the study presented by Cunningham et al. [8], that analyzes posts to the Art of the Mix forums requesting advice on, for example, the choice of songs, or song ordering rules.

The "AotM-2011" dataset [23] is a publicly available playlist collection derived from the Art of the Mix database. Each playlist is represented by song titles and artist names, linked to the corresponding identifiers of the Million Song Dataset[5] (MSD) [3], where available. The "8tracks" dataset is a private playlists collection derived from 8tracks. Each playlist is represented by song titles and artist names. Since we find multiple spellings for the same song–artist pairs, we use fuzzy string matching to resolve the song titles and artist names against the MSD, adapting the code released by Jansson et al. [16] for a very similar task.

We use the MSD as a common name space to correctly identify song–artist pairs. In both datasets, the songs that could not be resolved against the MSD are discarded, with one of two possible approaches. The first approach consists in simply removing the non-matched songs. The original playlists are preserved but with skips within them, which we ignore. The second approach consists in breaking up the original playlists into segments of consecutive matched songs, yielding shorter playlists without skips. We show results obtained on playlists derived from the first approach, but experiments on playlists derived from the second approach yielded equivalent conclusions.

We keep only the playlists with at least 3 unique artists and with a maximum of 2 songs per artist. This is to discard artist- or album-themed playlists, which may correspond to book-

[2] http://www.artofthemix.org

[3] https://8tracks.com

[4] Publishing playlists and interacting with individual users are still active services on the Art of the Mix, but the forums and blogs seem to be discontinued.

[5] https://labrosa.ee.columbia.edu/millionsong

Table 2 Descriptive statistics of the filtered AotM-2011 and 8tracks playlist datasets. We report the distribution of playlist lengths, number of artists per playlist and song frequency in the datasets (i.e., the number of playlists in which each song occurs)

Dataset	Statistic	min	1q	med	3q	max
AotM-2011	Playlist length	5	6	7	8	34
	Artists per playlist	3	5	7	8	34
	Song frequency	1	8	12	20	249
8tracks	Playlist length	5	5	6	7	46
	Artists per playlist	3	5	6	7	41
	Song frequency	1	9	15	30	2320

marking favorite artists, or saving full albums as playlists. While these are also valid criteria, we prefer to exclude them in this work. We also keep only the playlists with at least 5 songs to ensure a minimum playlist length. Songs occurring in less than 10 playlists are removed to ensure that the models have sufficient observations for each song.

We randomly assign 80% of the playlists to training and the remaining 20% to test. As in any recommendation task blind to item content, the songs that occur only in test playlists need to be removed because they cannot be modeled at training time. This affects the final playlist length and song frequency of the playlist datasets.

The filtered AotM-2011 dataset has 17,178 playlists with 7032 unique songs by 2208 artists. The filtered 8tracks dataset has 76,759 playlists with 15,649 unique songs by 4290 artists. Table 2 reports the distribution of playlist lengths, unique artists per playlist and song frequency in the datasets.

6 Results

We assess the ability of the four considered playlist models, Popularity, Song-CF, Playlist-CF and RNN, to recover withheld playlist continuations as described in Sect. 3. By comparing the performance of the different playlist models on the same experiment, or the performance of the same model on different experiments, we reason about the importance of considering the song context and the song order for next-song recommendations. Furthermore, we study the impact of the song popularity on the performance of the different models. As a reference, all the results include the performance of a dummy model that ranks next-song candidates at random (we call this model "Random").

6.1 Song context

Recall that Popularity predicts the next song disregarding the current and previous songs, i.e., it has no context. Song-CF predicts the next song on the basis of the current song but disregards the previous ones, i.e., it has a context of 1 song. Playlist-CF and RNN predict the next song on the basis of the full playlist, i.e., they have full song context.

Figure 3 reports the rank distribution of each playlist model. They are split by the position in the playlist for which the next-song prediction is made.[6] We consider only predictions up to position 8, which represent roughly the 90% of all the next-song predictions made in the AotM-2011 and the 8tracks datasets. From position 9 onward, the number of predictions quickly decreases and the results become less reliable.

The results in Fig. 3 show that Popularity and Song-CF do not systematically improve their predictions as they progress through the playlists. This is the expected result because Popularity has no context, and Song-CF has a constant context of 1 song. Their rank distributions remain overall stable with fluctuations easily explained by the fact that at each position the models deal with different songs. On the other hand, Playlist-CF and RNN are aware of the full song context. The results in Fig. 3 show that the performance of Playlist-CF clearly improves as it progresses through the playlists, and the performance of RNN improves slightly but steadily. This indicates that Playlist-CF and RNN benefit from increasingly longer song contexts.

In terms of absolute model performance, Song-CF is the least competitive model, slightly better but not clearly different than the random reference. Popularity and RNN show the most competitive overall performances. Playlist-CF has difficulties when the song context is short, but it consistently improves as it gains more context, until it eventually outperforms Popularity.

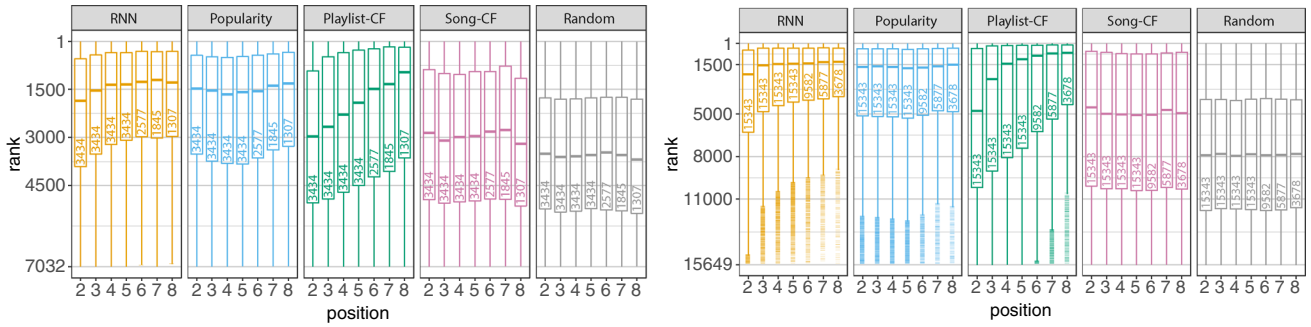
Summary of main observations:

- *Playlist-CF and RNN, aware of the full song context, improve their performance as the song context grows.*
- *Despite its simplicity, Popularity compares to RNN and, except for long contexts, outperforms Playlist-CF.*
- *Song-CF exhibits a poor performance.*

6.2 Popularity bias

The previous results pose an apparent contradiction: Popularity, unaware of the song context, performs comparably to RNN and, overall, slightly better than Playlist-CF, both aware of the full song context. Is it then important or not to exploit the song context? Furthermore, as discussed by Jannach and

[6] The position in the playlist for which the next-song prediction is made must not be confused with the song context length of the playlist model. For example, making a next-song prediction for a song in position 5, the playlist-based CF model has a context of 4 songs, while the song-based CF still has a context of 1 song (Table 1).



(a) AotM-2011 dataset

(b) 8tracks dataset

Fig. 3 Song context experiments. Distribution of ranks attained by the actual next songs in the test playlists (closer to 1 is better) for the AotM-2011 and the 8tracks datasets. Each panel corresponds to a playlist model. The x-axis indicates the position in the playlist for which a prediction is made. The y-axis indicates the attained ranks, and its scale relates to the number of songs in each dataset. The boxplots report the distribution of attained ranks. Outliers are indicated with small horizontal marks. The number of next-song predictions made at every position is annotated in the boxplots

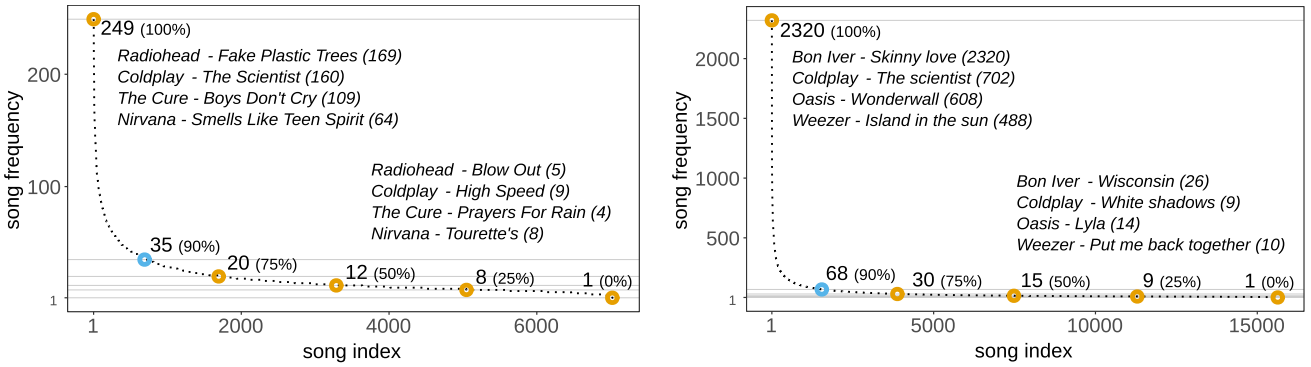
Ludewig [14], do marginal performance gains of RNN over Playlist-CF and Popularity justify its higher computational complexity?

To shed light on these two questions we deem it important to analyze the possible impact of the playlist datasets being biased toward popular songs, which is in fact a bias ubiquitous in the music consumption domain [5]. Within our study, we identify the popularity of a song with its frequency in each of the datasets, that is, with the number of playlists in which it occurs in each dataset. Table 2 and Fig. 4 show the song frequency distribution of the AotM-2011 and the 8tracks datasets. The AotM-2011 and the 8tracks datasets present a clear popularity bias, with a vast majority of songs occurring in few playlists and a few songs occurring in many playlists.

We consider again in Fig. 5 the performance of the four playlist models, but this time we distinguish whether the actual next songs in the test playlists were popular or not. We precisely define the popularity of a song as its relative frequency in the training playlists, as given by Eq. (1). The left panels report the median rank obtained when all the next-song predictions are considered. The central panels report the median rank obtained when the actual next songs belong to the 10% most popular songs in the datasets. The right panels report the median rank obtained when the actual next songs belong to the 90% least popular songs in the datasets (which we refer to as the "long tail"). In this particular case we report only the median rank instead of the whole rank distribution to obtain a more compact figure that facilitates the comparison of the playlist models across the different song-popularity levels.

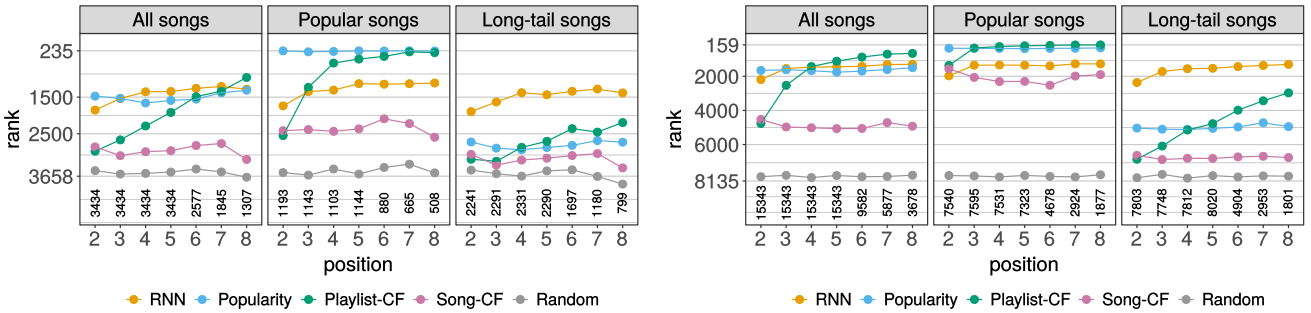
The results in Fig. 5 show that Popularity performs outstandingly well on the most popular songs, but it makes poor predictions for songs in the long tail. This is the natural consequence of its very design (Sect. 4.1). Playlist-CF performs reasonably well on the most popular songs, and it shows a quick improvement as the song context grows. Its performance on long-tail songs is poorer, but it shows a slight improvement as it gains song context, until given a context of at least 5 songs, it outperforms Popularity. The good performance of Playlist-CF on popular songs is not surprising because the scoring Eq. (3) favors songs occurring in many training playlists. However, the rather poor performance on long-tail songs is less expected, especially if we remember that our implementation of Playlist-CF considers all the training playlists as neighbors (Sect. 4.3), which should help to counteract the large amount of non-popular songs in the playlist datasets. Song-CF also performs better on popular songs than on long-tail songs, especially in the 8tracks dataset, where the popularity bias is stronger (Table 2, Fig. 4). RNN is competitive, and most importantly, in contrast to the other playlist models, its performance is largely unaffected by the popularity of the actual next songs in the test playlists.

Focusing on the performance of the playlist models on all next-song predictions (left panels in Fig. 5), Popularity seems comparable to the more sophisticated RNN. Given enough song context, Playlist-CF also seems to compete with RNN. However, as we have just seen, the overall strong performance of Popularity and Playlist-CF is the result of aggregating the accurate predictions made for a few popular songs (central panels in Fig. 5) with the rather poor predictions made for a vast majority of non-popular songs (right panels in Fig. 5). On the contrary, the performance of RNN is not affected by the song popularity. This observation must be taken into consideration to judge whether the higher computational complexity of the RNN model is justified, also considering the particular use case and target users of each recommender system. For example, the robustness of RNN to the popularity bias would be crucial to assist users interested in discovering long-tail music.



(a) AotM-2011 dataset **(b)** 8tracks dataset

Fig. 4 Unique songs in the AotM-2011 and the 8tracks datasets, sorted by frequency, i.e., by the number of playlists in which they occur. The colored dots correspond to songs located at specific percentile positions, with their absolute and percentile frequencies annotated (the latter in parentheses). Furthermore, examples of frequent and infrequent songs in the datasets are provided, with their absolute frequency annotated in parentheses



(a) AotM-2011 dataset **(b)** 8tracks dataset

Fig. 5 Popularity bias experiments. Median rank attained by the actual next songs in the test playlists (closer to 1 is better) for the AotM-2011 and the 8tracks datasets. Left: all songs are considered. Center: only the 10% most popular songs in the dataset are considered. Right: only the 90% least popular (long-tail) songs in the dataset are considered. The x-axis indicates the position in the playlist for which a prediction is made. The y-axis indicates the attained ranks, and its scale relates to the number of songs in each dataset. The number of next-song predictions made at every position is annotated

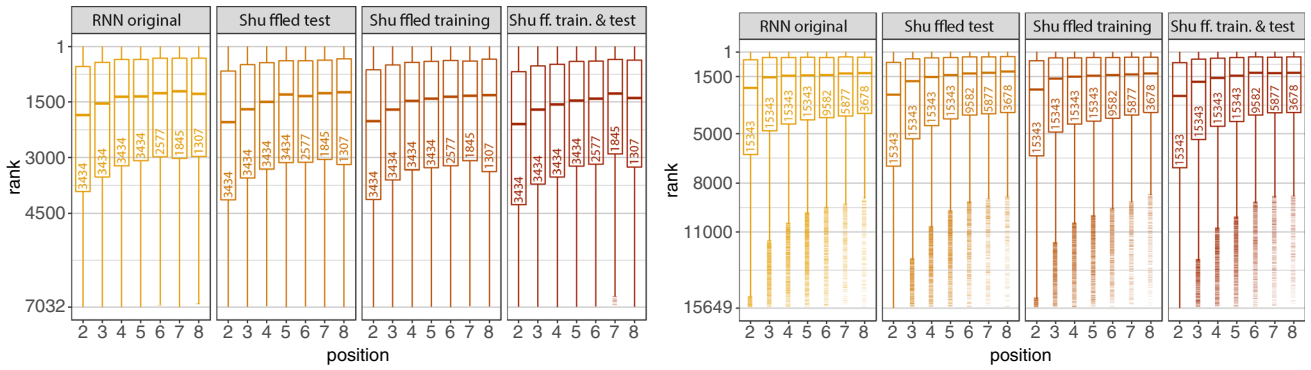
Summary of main observations:

- *RNN exhibits a competitive performance which is not affected by the popularity of the actual next songs.*
- *Popularity, Song-CF and Playlist-CF exhibit a considerable performance gap depending on the popularity of the actual next songs.*
- *Despite its overall poor performance on non-popular songs, Playlist-CF can exploit the song context to eventually outperform Popularity.*

6.3 Song order

RNN is the most complex of the four playlist models considered, and it is the only one aware of the song order. Furthermore, we have shown its good performance and robustness to dealing with infrequent music. We now investigate the importance of considering the song order by comparing the performance of RNN when it deals with original playlists, and when it deals with playlists where the song order has been manipulated. The rationale behind this experiment is the following: if the playlist datasets exhibit a consistent song order that RNN exploits to predict next-song recommendations, then we should observe a performance degradation when the song order is deliberately broken.

We devise three song order manipulation experiments. Firstly, we evaluate RNN on shuffled test playlists. This can be regarded as a *weak* check, because RNN could still have learned patterns based on the song order at training time. Secondly, we train another instance of the RNN model but using shuffled training playlists. We name it "RNN_{sh}." We re-tune its hyperparameters to make sure that the performance



(a) AotM-2011 dataset **(b)** 8tracks dataset

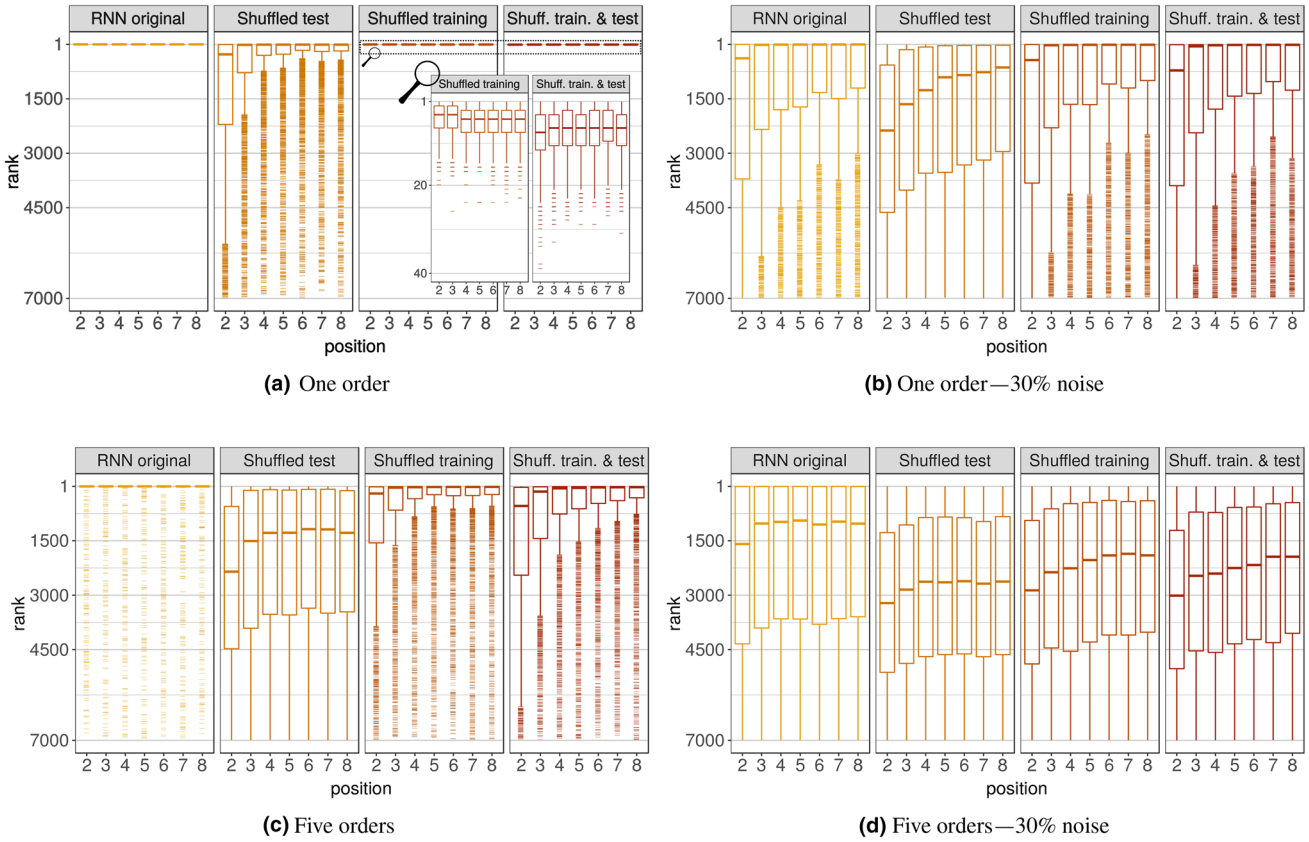
Fig. 6 Song order experiments. Distribution of ranks attained by the actual next songs in the test playlists (closer to 1 is better) for the AotM-2011 and the 8tracks datasets. The panels report the results of RNN and RNN_{sh} evaluated on original and shuffled test playlists. The x-axis indicates the position in the playlist for which a prediction is made. The y-axis indicates the attained ranks, and its scale relates to the number of songs in each dataset. The boxplots report the distribution of attained ranks. Outliers are indicated with small horizontal marks. The number of next-song predictions made at every position is annotated in the boxplots

is not compromised as a consequence of modifying the training playlists, but eventually we keep the same configuration because others do not yield consistent improvements. Then, we evaluate RNN_{sh} on original test playlists. This is a *strong* check, because we now make sure that RNN_{sh} cannot exploit the song order at training time. For completeness, we also evaluate RNN_{sh} on shuffled test playlists.

Figure 6 reports the rank distribution for each song order randomization experiment. As a reference, we also include the performance of RNN evaluated on original test playlists. The rank distributions are split by the position in the playlist for which the next-song prediction is made. As before, we consider only predictions up to position 8, which represent roughly the 90% of all the next-song predictions made in the AotM-2011 and the 8tracks datasets (Sect. 6.1). Surprisingly, the rank distributions are comparable across all song order randomization experiments, regardless of whether the song order is original, broken at test time, broken at training time, or broken both at training and at test time. This result provides an indication that the song order may not be an essential feature for next-song recommendations, and it would agree with similar findings derived from user experiments [17]. Alternatively, even though RNN models are the state of the art in many sequential tasks, this result could be explained by the incapability of this specific RNN model to properly exploit the song order.

To further investigate this question, we create synthetic playlist datasets where the song order is controlled. We start creating a playlist dataset where the song order within playlists is strictly ruled by an arbitrary but fixed universal song order. This dataset, which we name "One order," will let us determine whether the considered RNN model (Sect. 4.4) is actually able to capture order information. In any case, we already presume that such a definite song order behavior will not occur in real situations. We hypothesize with two possible sources of variation that may better respond to how natural playlists are organized: firstly, instead of a universal song order, there may exist several song orders corresponding to, for example, different underlying music taste profiles; secondly, one or several orders may exist, but they may be followed in a non-strict manner. We create three additional synthetic datasets according to these variations. We create a playlist dataset where the song order within playlists is strictly ruled by one of five arbitrary but fixed song orders, with the same number of playlists following each of the five orders. We refer to this dataset as "Five orders." We further create *noisy* versions of "One order" and "Five orders" such that the song order within the playlists is followed but in a non-strict manner. To achieve this, we copy the original datasets but replace a randomly chosen 30% of the songs of each playlist by unordered, randomly sampled songs from outside the playlist. We name the resulting datasets "One order—30% noise" and "Five orders—30% noise."

Each of the synthetic datasets has 15,000 playlists with 7000 unique songs, and each playlist has a length of exactly 10 songs. These specific values are chosen so that the synthetic datasets have similar characteristics to a natural collection like AotM-2011. The concept of *artist* does not exist, and the song orders are defined arbitrarily. Using more than five song orders, or a noise factor higher than 30%, yielded very challenging datasets that were not as illustrative as the created ones. The synthetic datasets are split into training and test playlists as described in Sect. 5. We then use them to conduct the song order randomization experiments described



(a) One order

(b) One order—30% noise

(c) Five orders

(d) Five orders—30% noise

Fig. 7 Randomized song order experiments with synthetic datasets. Distribution of ranks attained by the actual next songs in the synthetic test playlists (closer to 1 is better) for the AotM-2011 and the 8tracks datasets. The panels report the results of RNN and RNN_{sh} evaluated on original and shuffled test playlists. The x-axis indicates the position in the playlist for which a prediction is made. The y-axis indicates the attained ranks, and its scale relates to the number of songs in each dataset. The boxplots report the distribution of attained ranks. Outliers are indicated with small horizontal marks. The number of next-song predictions made at every position is always exactly 3000

at the beginning of this section. Figure 7 reports the rank distribution for each song order randomization experiment, for each synthetic playlist dataset.

We first analyze the results obtained on "One order" (Fig. 7a), which should let us determine whether the considered RNN model is actually able to capture order information. The performance of RNN on original test playlists is perfect, with all ranks equal to 1. This shows that the considered RNN model would be able to capture a universal song order if there were one. Consequently, we can conclude that the playlists in the AotM-2011 and the 8tracks datasets are not strictly ordered; otherwise, RNN would have been able to perfectly extend them. Again in Fig. 7a, precisely because RNN learned the song order strictly, its performance on shuffled test playlists is comparatively very poor. We now move on to RNN_{sh}. The performance of RNN_{sh} is not perfect but very good on both original and shuffled test playlists. This suggests that RNN_{sh} follows a different strategy than RNN. Since RNN_{sh} is trained on shuffled playlists, the strict song order is not anymore enforced. Instead, RNN_{sh} learns to predict songs in the *proximity* of the current song, meaning songs that are few positions before or after the current song within the universal song order. In other words, training the RNN model on shuffled playlists works as a regularization that favors learning song proximity rather than strict song order.

We have found evidence that the playlists in the AotM-2011 and the 8tracks datasets are not ruled by a strict, universal song order. In fact, since we train dedicated instances of the RNN model for each dataset, we know that the playlists are also not ruled by a strict, *dataset-specific* song order. However, this does not imply that the playlists are, on the other end, completely unordered. There may exist intermediate song order rules, which in real situations may further be affected by different sources of uncertainty. We examine the results obtained on the remaining synthetic datasets (Fig. 7b–d) as an approximation to the more complex song order patterns that may underlie the playlists of the AotM-2011 and the 8tracks datasets.

Figure 7b, c shows an overall, noticeable performance degradation compared to Fig. 7a. Still, RNN evaluated on

original test playlists performs almost perfectly for "Five orders" and competitively for "One order—30% noise." The performance degrades strongly when RNN is evaluated on shuffled test playlists, which suggests that it had indeed captured song order patterns. RNN_{sh} again shows that training on shuffled playlists provides a regularization effect, because its performance on original and shuffled test playlists is comparable. Figure 7d reports the results on the most challenging of the synthetic datasets and exhibits a generalized, clear performance degradation. While we cannot derive further conclusions regarding the nature of the playlists in the AotM-2011 and the 8tracks datasets, the comparison between Figs. 6 and 7 suggests that real playlists may indeed be subject to complex, noisy song order rules.

We know that the playlists in the AotM-2011 and the 8tracks datasets do not follow simple song order patterns, and thus they could be either completely unordered or ruled by complex, noisy song order rules. In both cases, the song order experiments on the natural datasets show that RNN performs equivalently to RNN_{sh} (Fig. 6), which could be the result of RNN adopting the same strategy of RNN_{sh}, that is, relying on song proximity patterns rather than on complex or inexistent song order patterns. Since the AotM-2011 and the 8tracks datasets do not have a strict song order, the concept of *proximity* could be understood as *fitness*, meaning that RNN and RNN_{sh} could be predicting next songs that fit well the playlist being extended. This could also explain why RNN and RNN_{sh} improve their performance as the song context grows, namely because a longer song context better specifies the playlist under consideration. A longer song context would not necessarily translate into performance improvements if RNN could rely on clear song order patterns, in which case knowing a single song would suffice to accurately predict the next one (see the performance of RNN evaluated on original playlists in Fig. 7a, c).

Summary of main observations:

- *RNN achieves comparable performance on original and on shuffled test playlists. It also compares to* RNN_{sh}*, which is completely unaware of the song order.*
- *Experiments on strictly ordered synthetic datasets show that RNN can learn song order patterns.*
- *From the previous, we conclude that the AoM-2011 and the 8tracks datasets are not strictly ordered.*
- *Further experiments on synthetic datasets suggest that the AotM-2011 and the 8tracks datasets might be ruled by complex, noisy song order rules.*
- *When RNN relies on song fitness patterns rather than on song order patterns, it benefits from longer song contexts, which better identify the playlist being extended.*

7 Conclusion

We have explicitly investigated the impact of the song order, the song context and the popularity bias in music playlists for the task of predicting next-song recommendations. We have conducted dedicated off-line experiments on two datasets of hand-curated music playlists comparing the following playlist models: a popularity-based model, a song-based CF model, a playlist-based CF model, and an RNN model. These models are well established and widely used and exploit the song context and the song order to different extents. Our results indicate that the playlist-based CF model and the RNN model, which can consider the full song context, do benefit from increasingly longer song contexts. However, we observe that a longer song context does not necessarily translate into outperforming the simpler popularity-based model, which is unaware of the song context. This is explained by the popularity bias in the datasets, i.e., the coexistence of few, popular songs with many, non-popular songs. Failing to take into account the popularity bias masks important performance differences: the popularity-based model, the song-based CF model and the playlist-based CF model exhibit considerable differences in performance depending on the popularity of the actual next songs in the test playlists. On the contrary, the more complex RNN model has a stable performance regardless of the song popularity. This effect must be taken into account in the design of playlist models for specific use cases and target users. The RNN model is the only of the considered playlist models aware of the song order. We have found that its performance on original and shuffled playlists is comparable, suggesting either that the song order is not crucial for next-song recommendations, or that the RNN model is unable to fully exploit it. We have further investigated this question by evaluating the RNN model on synthetic datasets with controlled song orders. We have found that the RNN model is able to capture a universal song order if there is one. This implies that the natural playlists datasets considered do not follow a strict song order, although they might be ruled by complex, noisy song order rules. Finally, and regarding the evaluation methodology, we have proposed an approach to assess the quality of the recommendations that observes the complete recommendation lists instead of focusing on the top K recommendations. Doing so provides a more complete view on the performance of the playlist models.

Acknowledgements We thank Bruce Ferwerda, Rainer Kelz, Rocío del Río Lorenzo, and David Sears for their valuable feedback. This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 670035 (Con Espressione). Open access funding provided by Johannes Kepler University Linz.

A Model configurations

A.1 Song popularity

This model computes the popularity of all the unique songs in the dataset, that is, 7032 songs for the AotM-2011 dataset and 15,649 songs for the 8tracks dataset.

A.2 Song-based collaborative filtering

This model computes pairwise similarities for all the unique songs in the dataset, that is, 7032 songs for the AotM-2011 dataset and 15,649 songs for the 8tracks dataset.

A.3 Playlist-based collaborative filtering

This model computes the similarity of each test playlist to all the training playlists in the dataset, that is, 13,744 playlists for the AotM-2011 dataset and 61,416 playlists for the 8tracks dataset. We also experimented using 100, 500 and 1000 training playlists but did not achieve better results.

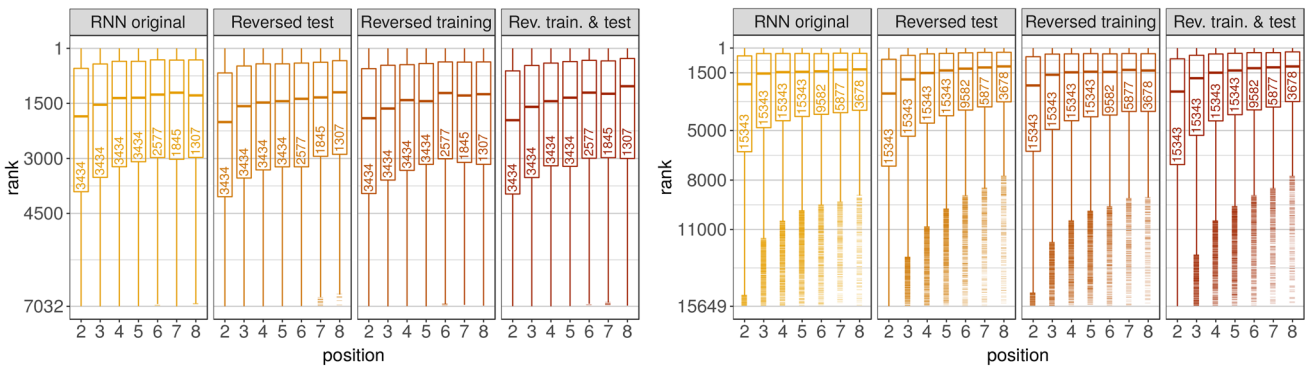
A.4 Recurrent neural networks

We experiment with different loss functions, namely categorical cross-entropy, Bayesian pairwise ranking (BPR) [26] and TOP-1 [12]. The RNN is optimized using AdaGrad [9] with momentum and $L2$-regularization. We also experiment with dropout [30] in the recurrent layer, but none of the final configurations use it. We tune the number of units, the learning rate, the batch size, the amount of momentum, the $L2$-regularization weight and the dropout probability on a withheld validation set, by running 100 random search experiments [2] for each of the loss functions mentioned above. The final model configuration is chosen according to the validation recall at 100 (i.e., the proportion of times that the actual next song is included within the top 100 ranked candidates), which we consider a proxy of the model's ability to rank the actual next songs in top positions. The number of training epochs is chosen on the basis of the validation loss.

For the AotM-2011 dataset, the final model uses the TOP-1 loss and it has 200 hidden units. It is trained on mini-batches of 16 playlists, with a learning rate of 0.01, a momentum coefficient of 0.5 and an $L2$-regularization weight of 0.1. For the 8tracks dataset, the final model uses the TOP-1 loss and it has 200 hidden units. It is trained on mini-batches of 64 playlists, with a learning rate of 0.025, a momentum coefficient of 0.3 and an $L2$-regularization weight of 0.02. For both datasets, the hyperparameters and architecture of the RNN models trained on shuffled and reversed playlists were re-tuned, but since other configurations did not yield clearly better results, we decided to use the same settings for consistency.

B Additional experiments

We conduct, for completeness, a related set of experiments consisting in reversing the song order instead of randomizing it. A similar experiment was proposed by Chen et al. [6] to investigate the importance of the "directionality" in next-song recommendations. Chen et al. found only small performance differences evaluating the Latent Markov



(a) AotM-2011 dataset

(b) 8tracks dataset

Fig. 8 Reversed song order experiments. Distribution of ranks attained by the actual next songs in the test playlists (closer to 1 is better) for the AotM-2011 and the 8tracks datasets. The panels include the predictions of the RNN on the original playlists and on the different reversed song order experiments. The x-axis indicates the position in the playlist for which a prediction is made. The y-axis indicates the attained ranks, and its scale relates to the number of songs in each dataset. The boxplots report the distribution of attained ranks. Outliers are indicated with small horizontal marks. The number of next-song predictions made at every position is annotated in the boxplots

Embedding model on original and reversed playlists. We replicate the different settings from our previous experiments: we first train the RNN model on original playlists and evaluate it on reversed playlists. Then, we train the RNN model on reversed playlists and evaluate it on original playlists. Finally, we train and evaluate the RNN model on reversed playlists. Figure 8 reports the rank distributions under each reversed song order experiment. As expected, the results are comparable to those reported in Fig. 6. That is, the distribution of ranks is comparable for all the reversed song order experiments.

References

1. Aizenberg N, Koren Y, Somekh O (2012) Build your own music recommender by modeling internet radio streams. In: Proceedings of WWW, pp 1–10
2. Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(1):281–305
3. Bertin-Mahieux T, Ellis DP, Whitman B, Lamere P (2011) The million song dataset. In: Proceedings of ISMIR, pp 591–596
4. Bonnin G, Jannach D (2014) Automated generation of music playlists: survey and experiments. ACM Comput Surv 47(2):1–35
5. Celma O (2010) Music recommendation and discovery. Springer, Berlin
6. Chen S, Moore JL, Turnbull D, Joachims T (2012) Playlist prediction via metric embedding. In: Proceedings of SIGKDD, pp 714–722
7. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv preprint arXiv:1406.1078
8. Cunningham SJ, Bainbridge D, Falconer A (2006) "More of an art than a science": supporting the creation of playlists and mixes. In: Proceedings of ISMIR
9. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
10. Flexer A, Schnitzer D, Gasser M, Widmer G (2008) Playlist generation using start and end songs. In: Proceedings of ISMIR, pp 173–178
11. Hariri N, Mobasher B, Burke R (2012) Context-aware music recommendation based on latent topic sequential patterns. In: Proceedings of RecSys, pp 131–138
12. Hidasi B, Karatzoglou A, Baltrunas L, Tikk D (2016) Session-based recommendations with recurrent neural networks. In: Proceedings of ICLR
13. Hidasi B, Quadrana M, Karatzoglou A, Tikk D (2016) Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Proceedings of RecSys, pp 241–248
14. Jannach D, Ludewig M (2017) When recurrent neural networks meet the neighborhood for session-based recommendation. In: Proceedings of RecSys, pp 306–310
15. Jannach D, Lerche L, Kamehkhosh I (2015) Beyond "hitting the hits": generating coherent music playlist continuations with the right tracks. In: Proceedings of RecSys, pp 187–194
16. Jansson A, Raffel C, Weyde T (2015) This is my jam data dump. In: Proceedings of ISMIR
17. Kamehkhosh I, Jannach D, Bonnin G (2018) How automated recommendations affect the playlist creation behavior of users. In: Joint proceedings of IUI workshops, Tokyo, Japan
18. Knees P, Pohle T, Schedl M, Widmer G (2006) Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In: Proceedings of international workshop on multimedia IR, pp 147–154
19. Lee JH, Bare B, Meek G (2011) How similar is too similar? Exploring users' perceptions of similarity in playlist evaluation. In: Proceedings of ISMIR, pp 109–114
20. Lipton ZC, Berkowitz J, Elkan C (2015) A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019
21. Logan B (2002) Content-based playlist generation: exploratory experiments. In: Proceedings of ISMIR
22. McFee B, Lanckriet GR (2011) The natural language of playlists. In: Proceedings of ISMIR, pp 537–542
23. McFee B, Lanckriet GR (2012) Hypergraph models of playlist dialects. In: Proceedings of ISMIR, pp 343–348
24. Platt JC, Burges CJ, Swenson S, Weare C, Zheng A (2002) Learning a Gaussian process prior for automatically generating music playlists. In: Proceedings of NIPS, pp 1425–1432
25. Pohle T, Pampalk E, Widmer G (2005) Generating similarity-based playlists using traveling salesman algorithms. In: Proceedings of DAFx, pp 220–225
26. Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of UAI, pp 452–461
27. Rendle S, Freudenthaler C, Schmidt-Thieme L (2010) Factorizing personalized Markov chains for next-basket recommendation. In: Proceedings of WWW, pp 811–820
28. Ricci F, Rokach L, Shapira B (2015) Recommender systems handbook, 2nd edn. Springer, New York
29. Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. In: Proceedings of WWW, pp 285–295
30. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
31. Tan YK, Xu X, Liu Y (2016) Improved recurrent neural networks for session-based recommendations. In: Proceedings of DLRS@RecSys, pp 17–22
32. Vall A, Schedl M, Widmer G, Quadrana M, Cremonesi P (2017) The importance of song context in music playlists. In: RecSys 2017 poster proceedings, Como, Italy
33. Vall A, Quadrana M, Schedl M, Widmer G (2018) The importance of song context and song order in automated music playlist generation. In: Proceedings of ICMPC-ESCOM, Graz, Austria