Cold-start Playlist Recommendation via Multitask Learning

ABSTRACT

Playlist recommendation concerns producing a sequence of songs that a user might enjoy. We investigate this problem in three different cold-start scenarios: (i) *cold songs*, where we recommend newly released songs to extend existing playlists; (ii) *cold playlists*, where we recommend a set of songs to form a new playlist for an existing user; (iii) *cold users*, where we recommend a set of songs to form a new playlist for a new user.

We propose a flexible multitask learning method to deal with all three settings. The method learns from user-curated playlists, and encourages songs in the playlist to be ranked higher than those are not by minimising a bipartite ranking loss. We formulate the objective as a constrained convex optimisation problem, and show how this may be approximated by an unconstrained objective inspired by an equivalence relationship between bipartite ranking and binary classification. Empirical results on two real music playlist datasets show the proposed approach has good performance for playlist recommendation in cold-start settings.

KEYWORDS

Playlist recommendation, Cold start, Multitask learning

1 INTRODUCTION

Online music streaming services (e.g., Spotify, Pandora, Apple Music) play an increasingly important role in the digital music industry. A key ingredient of these services is the ability to automatically recommend songs to help users explore large collections of music, as well as create an uninterrupted listening experience. Such recommendation is often in the form of a *playlist*, which is simply an ordered sequence of songs.

Conventional recommender systems for books or movies [27, 32] typically learn a score function via matrix factorisation [19], and recommend the item that achieves the highest score. This approach is not suited to deal with *cold-start* settings, where there is no historical data for either users or items. Further, in playlist recommendation, one has to recommend a subset of a large collection of songs instead of only one top ranked song. Enumerating all possible such subsets is intractable; additionally, it is likely that more than one playlist is satisfactory, since users generally maintain more than one playlist when using a music streaming service.

We investigate the problem of recommending a playlist of songs for a given user. If the user is new to the system, we would like to make a decent default recommendation by learning from all available playlists of existing users. On the other hand, if the user already has a few playlists in the system, it is expected the recommender system can also learn from this information and hopefully make better recommendations. The challenge of this task is two-fold:

(i) *Sparsity*. While a large number of playlists are hosted by music streaming services, playlists for each user is still limited. This is exacerbated by the long-tail distribution that most users only have few number of playlists, while a small portion of users have a very large number of playlists. Similarly, a small number of popular songs appeared in a large number of playlists, and the most majority of songs only appear in a few playlists [7].
(ii) *Noise*. There could be noise in metadata [8] or users might randomly choose songs from their music collection when composing playlists [25], which makes it hard to learn the intent of a playlist.

These challenges make it hard to make recommendations better than simply ranking songs according to its popularity [7, 8, 23],

In this paper, we propose a novel approach which ranks songs that could end up being in playlist for the given user above those unlikely. Compared to the Bayesian Personalised Ranking (BPR) approach which requires all songs in playlist ranks higher than songs are not [23, 30], we focus on the lowest ranked song in playlist and penalise any song ranked higher but does not appear in playlist. This approach has been shown to work more efficiently when one cares about only the top ranked items [20].

To learn the parameters, we optimise a multitask objective, which involves a constrained convex optimisation. We show how one can avoid dealing with the large number of constraints using an equivalence relationship between bipartite ranking and binary classification. This results in an unconstrained classification loss that approximates the multitask objective. Experiments on two real playlist datasets show that our proposed approaches work effectively, and the classification approach in particular significantly improves the learning efficiency while maintaining the same level of high performance as the ranking approach.

Our paper is organised as follows. Section 2 illustrate three cold-start settings, and in Section 3, we describe the motivations of the multitask objective, and propose a ranking approach that optimises the objective by solving a constrained optimisation problem. We then describe a classification loss which enables us to approximately optimise the objective by solving an unconstrained optimisation problem. Section 4 discusses previous work that are most relevant to our work. Section 5 details our experiment on two real playlist datasets. Lastly, we summarises the paper in Section 6 and describe future work.

2 PROBLEM STATEMENT

We formulate the problem of recommending music in three cold-start settings in the context of playlist (i.e., a sequence of songs):

(i) *cold songs*, where we recommend newly released songs to extend existing playlists;
(ii) *cold playlists*, where we recommend a set of songs to form a new playlist for an existing user;

RecSys'18, October 2-7, 2018, Vancouver, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

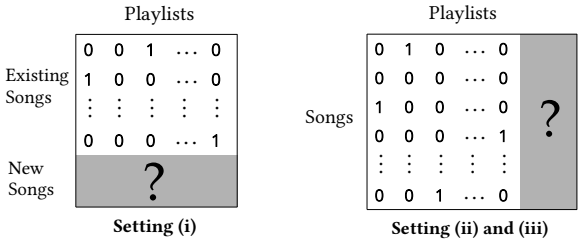

Figure 1: Illustration of three cold-start settings

(iii) *cold users*, where we recommend a set of songs to form a new playlist for a new user.

Suppose we are given a dataset $\mathcal{D}$ with N playlists from U users, where songs in every playlist are from a music collection with M songs. We further assume each user has at least one playlist, and each song in the collection is appeared in at least one playlist. We illustrate the three cold-start settings in Figure 1, where rows represent songs and columns represent playlists. If entry (m, n) is 1 (or 0), it means song m can (or not) be found in playlist n. The shaded rows/columns with ? marks represent the test set, where all entries are unknown. In the setting of *cold songs*, we have a set of playlists (columns) form by existing songs (rows), and the task is to determine whether we should add each new song (row) for an existing playlist. In the setting of *cold playlists*, we have a set of playlists (columns) from existing users, and the task is to recommend a set of songs to form a new playlist (column) for an existing user. Lastly, in the *cold users* setting, which is similar to the *cold playlists* setting except the target user that we will recommend a playlist (formed by a set of songs) is a new user, i.e., the user does not have any playlists in the system.

3 MULTITASK LEARNING FOR MUSIC RECOMMENDATION

In this section, we first introduce a multitask objective that supports the cold-start recommendation tasks described in Section (2). We propose a ranking approach to optimise the multitask objective by solving an convex constrained optimisation problem. We analyse the challenge to deal with a large number of constraints and then propose an classification approach that approximates the multitask objective without any constraints, which results in an efficient optimisation of the multitask objective.

3.1 Multitask learning objective

Let P_u denote the set of playlists from users u. We aim to learn a function $f(u, i, m)$ that measures the affinity between user u, playlist i and song m. Suppose f has a linear form

$$f(u, i, m) = \mathbf{w}_{i,u}^\top \mathbf{x}_m, \tag{1}$$

where $u \in \{1, \dots, U\}$, $i \in P_u$, $m \in \{1, \dots, M\}$, vector $\mathbf{w}_{i,u}$ represents the parameters of playlist i from user u, and $\mathbf{x}_m$ is a feature vector of song m.

We decompose the representation $\mathbf{w}_{i,u}$ of a playlist i from user u in (1) into three components, i.e.,

$$\mathbf{w}_{i.u} = \boldsymbol{\alpha}_i + \boldsymbol{\beta}_u + \boldsymbol{\mu},$$

where $\boldsymbol{\alpha}_i$ is a parameter vector specific for playlist i, $\boldsymbol{\beta}_u$ is the component for user u, and $\boldsymbol{\mu}$ is a representation shared by all users.

The learning task is to minimise the empirical risk of affinity function f on dataset $\mathcal{D}$ over all parameters. Let θ which represents all parameters in $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_u, \boldsymbol{\mu}$, $u \in \{1, \dots, U\}$, $i \in P_u$. Formally, we solve an optimisation problem

$$\min_{\theta} \ \Omega(\theta) + R(f, \mathcal{D}), \tag{2}$$

where $\Omega(\theta)$ is the regularisation term and $R(f, \mathcal{D})$ denotes the empirical risk. We call the objective in problem (2) a multitask learning objective, since jointly learn from multiple tasks where each task is to recommend a set of songs given either a user or a playlist.

We further assume that playlists of the *same* user have *similar* representations, which can be achieved by imposing an L1 regularisation, i.e., $\sum_{u=1}^{U} \sum_{i \in P_u} \|\boldsymbol{\alpha}_i\|_1$ that encourages sparse representations. Lastly, although the representations of different users are different, we assume they nonetheless share a small number of features in their representations. This can be formalised similarly by imposing an additional L1 regularisation term $\|\boldsymbol{\mu}\|_1$.

The regularisation terms in our multitask learning objective can be summarised as

$$\Omega(\theta) = \lambda_1 \sum_{u=1}^{U} \sum_{i \in P_u} \|\boldsymbol{\alpha}_i\|_1 + \lambda_2 \sum_{u=1}^{U} \|\boldsymbol{\beta}_u\|_2^2 + \lambda_3 \|\boldsymbol{\mu}\|_1, \tag{3}$$

where we add an L2 regularisation term to penalise large values in user representations, and $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_+$ are regularisation constants.

Once all parameters have been learned, we can first score each song according to available information (e.g., an existing playlist, an existing user or a new user), then form a playlist by either taking the top-K scored songs or sampling songs proportional to their scores. Specifically, in the cold-start setting (i), where we recommend new songs to extend an existing playlist i from user u, we can score a new song m by

$$\hat{f}(u, i, m) = (\boldsymbol{\alpha}_i + \boldsymbol{\beta}_u + \boldsymbol{\mu})^\top \mathbf{x}_m.$$

Further, in cold-start setting (ii), i.e., recommending a set of songs for an existing user u (to form a playlist), we score song m by

$$\hat{f}(u, \cdot, m) = (\boldsymbol{\beta}_u + \boldsymbol{\mu})^\top \mathbf{x}_m.$$

Finally, in the last cold-start setting (iii) where we recommend a set of songs for a new user, each song m can be scored by

$$\hat{f}(\cdot, \cdot, m) = \boldsymbol{\mu}^\top \mathbf{x}_m.$$

3.2 Recommending the most probable songs

In this section, we describe a ranking approach that learns to rank songs in playlist above those not in it, which aims to rank the most probable songs above those unlikely when making a recommendation. The corresponding loss function of this approach is known as the Bottom-Push loss [31] in bipartite ranking literature. Formally, we minimise the number of songs not in playlist but ranked above the lowest ranked song in playlist, i.e., the (normalised) empirical risk is

$$R_{\text{RANK}}(\theta) = \frac{1}{N} \sum_{u=1}^{U} \sum_{i \in P_u} \frac{1}{M_-^i} \sum_{m': y_{m'}^i = 0} [\![\min_{m: y_m^i = 1} f(u, i, m) \le f(u, i, m')]\!], \tag{4}$$

where M^i_- denotes the number of songs that are not in playlist i, and $[\![\cdot]\!]$ is the indicator function that represents the 0-1 loss.

As a remark, it is easy to see that the order of songs in a specific playlist does not affect the empirical risk (4) as long as they are scored higher than the lowest scored song in that playlist. The optimisation problem (2) now become the following regularised risk minimisation:

$$\min_{\theta} \ \Omega(\theta) + R_{\text{RANK}}(\theta). \tag{5}$$

There are two challenges to optimise the above objective, namely, the non-differentiable 0-1 loss and the *min* operator in $R_{\text{RANK}}(\theta)$. To address these challenges, we first replace the 0-1 loss with one of its convex surrogate $\ell(f, y)$, (e.g., the exponential loss $\ell(f, y) = e^{-fy}$, the logistic loss $\ell(f, y) = \log(1 + e^{-fy})$, or the squared hinge loss $\ell(f, y) = [\max(0,\ 1 - fy)]^2$).

3.3 Solving a constrained optimisation problem

Suppose we use the exponential loss to upper-bound the 0-1 loss in R_{RANK}, the multitask learning objective (5) can be optimised by

$$\min_{\theta} \ \Omega(\theta) + \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M^i_-}\sum_{m':y^i_{m'}=0}\exp\left(f(u,i,m') - \min_{m:y^i_m=1} f(u,i,m)\right). \tag{6}$$

To deal with the challenge imposed by the *min* operator in the exponential function, we reformulate problem (6) into a constrained optimisation problem by introducing slack variables ξ_i, $i \in P_u$, $u \in \{1, \ldots, U\}$,

$$\begin{aligned} \min_{\theta} \ & \Omega(\theta) + \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M^i_-}\sum_{m':y^i_{m'}=0} e^{f(u,i,m')-\xi_i} \\ s.t. \ & \xi_i \le f(u,i,m),\ \forall m \in \{1,\ldots,M\} \text{ and } y^i_m = 1. \end{aligned} \tag{7}$$

One may observe that the objective of problem (7) is convex but not differentiable due to the L1 regularisation terms in $\Omega(\theta)$, nonetheless, we can use its sub-gradient. Further, the number of constraints in problem (7) is

$$\sum_{u=1}^{U}\sum_{i\in P_u}\sum_{m=1}^{M}[\![y^i_m = 1]\!],$$

in other words, the number of constraints equals the total number of songs played in all playlists. Asymptotically, it is of order $O(\bar{L}N)$ where $\bar{L}$ is the average number of songs in playlists. Although $\bar{L}$ is dataset dependent, and is typically less than 100, the total number of playlists N can be very large in production systems (e.g., Spotify hosts more than 2 billion playlists [29]), which imposes significant challenge when optimising problem (7).

We want to mention two techniques that can be employed to alleviate this issue: the cutting-plane [4] method and the sub-gradient method. We have found both approaches converge extremely slow in practice for this problem, in particular, the cutting plane method is required to deal with constrained optimisation problems with at least N constraints, which is still challenging when N is large.

In this paper, we address this issue by formulating an unconstrained optimisation problem which approximates the objective in problem (7), by leveraging an equivalence relationship between bipartite ranking and binary classification [14], we describe this approach in the following section.

3.4 Unconstrained optimisation with classification loss

It has been known that binary classification and bipartite ranking are closely related [14, 26]. In particular, Ertekin and Rudin [14] have shown that the P-Norm Push bipartite ranking loss [31] is equivalent to the P-Classification loss [14] when using the exponential surrogate. Further, the P-Norm Push loss is an approximation of the Infinite-Push loss [2], or equivalently, the Top-Push loss [20], which focuses on the highest ranked negative example instead of of the lowest ranked positive example as in (4).

Inspired by these connections, we first seek a bipartite ranking loss that approximates the Bottom-Push loss in (4), then propose a classification loss that is equivalent to this bipartite ranking loss. The reason not to directly optimise the bipartite ranking loss is due to computation cost, and classification loss can be optimised more efficiently in general [14].

We can approximate the *min* operator by utilising the well known Log-sum-exp approximation of the *max* operator [9, p. 72],

$$\min_i z_i = -\max_i(-z_i) \approx -\frac{1}{p}\log\sum_i e^{-pz_i}, \tag{8}$$

where $p > 0$ is a parameter that trades off the approximation precision. This approximation becomes precise when $p \to \infty$.

By Eq. (4) and (8), the empirical risk R_{RANK} can be approximated (with the exponential surrogate) by $\widetilde{R}_{\text{RANK}}$ defined as

$$\widetilde{R}_{\text{RANK}}(\theta) = \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M^i_-}\left(\sum_{m:y^i_m=1}\left(\sum_{m':y^i_{m'}=0} e^{-(f(u,i,m)-f(u,i,m'))}\right)^p\right)^{\frac{1}{p}}. \tag{9}$$

One may note that $\widetilde{R}_{\text{RANK}}$ can be transformed into the standard P-Norm Push loss by swapping the positives ($m : y^i_m = 1$) and negatives ($m' : y^i_{m'} = 0$). Inspired by this observation, we swap the positives and negatives in the P-Classification loss (by taking care of signs), which results in the following classification risk:

$$R_{\text{CLF}}(\theta) = \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\left(\frac{1}{pM^i_+}\sum_{m:y^i_m=1} e^{-pf(u,m,i)} + \frac{1}{M^i_-}\sum_{m':y^i_{m'}=0} e^{f(u,m',i)}\right). \tag{10}$$

We have the following lemma:

Lemma 3.1. *Let* $\theta^* \in \operatorname{argmin}_\theta R_{CLF}$ *(assuming minimisers exist), then* $\theta^* \in \operatorname{argmin}_\theta \widetilde{R}_{RANK}$.

Proof. This theorem can be proved by swapping the positives and negatives in the proof of the equivalence between P-Norm Push loss and P-Classification loss [14]. □

We can therefore create an unconstrained optimisation problem using the classification risk R_{CLF}:

$$\min_{\theta} \quad \Omega(\theta) + R_{\text{CLF}}(\theta). \tag{11}$$

The objective in problem (11) is convex but not differentiable due to the L1 regularisation terms in $\Omega(\theta)$, but it can be efficiently solved using the Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) [3] L-BFGS variant.

We refer the approach that solves problem (11) as *Multitask Classification* in experiment. As a remark, the optimal solutions of problem (11) are not necessarily the optimal solutions of problem

$\min_\theta\ \Omega(\theta) + \widetilde{R}_{\text{RANK}}(\theta)$ due to the regularisation terms, however, when parameters θ are small (which is generally the case when using regularisation), the two objectives can nonetheless approximate each other in acceptable level.

4 RELATED WORK

We summarise recent work most related to recommending music for playlist according the problem settings, the recommendation methods, as well as the information being utilised.

Problem settings. There are three typical settings: playlist generation, next song recommendation, and playlist continuation. There is rich collection of recent literature on the playlist generation or prediction problem [5, 11, 24, 25, 28]. Typical work is to generate a complete playlist given some seed, for example, the AutoDJ system [28] generates playlists given one or more seed songs, a playlist for a specific user can be generated by Groove Radio given a seed artist [5], or a seed location in hidden space, where all songs are embedded, should be specified in order to generate a complete playlist [11]. There are also work that focus on evaluating the learned playlist model, without concretely generate playlist [24, 25]. More details of playlist generation can be found in this recent survey [8].

Next song recommendation [7, 16, 17] is to predict the next song a user might play after observing some context, for example, the most recent sequence of songs a user interacted with the system was used to infer the contextual information, which was further used to rank the next possible song with regards to a topic-based sequential patterns learned from user playlists [16]. The artists appeared in user's listening history can be used as context, which, together with the popularity of song or frequency of artists collocations, were further used to score the next song [7, 23]. It is obvious that next song recommendation techniques can also be used to generate a complete playlist by picking the next song sequentially [5, 7].

Playlist continuation is to add one or more songs to a playlist, given the added songs serve the same purpose of the original playlist [29, 33]. One can immediately notice that playlist generation and next song recommendation are special cases of playlist continuation, which means techniques developed for playlist generation and next song recommendation can both be used for playlist continuation.

In the collaborative filtering literature, the cold-start setting has primarily been addressed through suitable regularisation of matrix factorisation parameters based on exogenous user- or item-features [1, 10, 21]. Another popular approach involves explicitly mapping such features to the latent embeddings [15].

Methods. Ranking approaches such as popularity-based ranking have been shown to work surprisingly well [7, 8, 23]. The reason is believed to be the long-tail distribution of songs in playlists [7, 12]. This approach has been further improved by taking into account artist information in addition to song popularity, which creates a comparably strong baseline that outperforms many sophisticated approaches such as Bayesian Personalised Ranking based approach, neighbourhood methods, and approaches making use of association rules and sequential patterns [7, 23]. Ranking method has also been used as a post-processing component in more sophisticated approaches, where a subset of songs were selected by scoring [17] or matching [16] before ranking which optimises specific characteristics of the generated playlists.

Markov chains and related approaches were also widely used for playlist generation by casting the task as language modelling problem [24], random walks in a hyper-graph [25] where songs were first grouped (by genre, year of release, or social tags etc.) to serve as edges in the hyper-graph, or samples of Markov chains in latent space where songs are embedded as (pairs of) points [11]. Other techniques including playlist generation as sequential classification based on context [5], Gaussian process regression for user preference learning [28], topic models for sequential pattern mining and the well-known matrix factorisation techniques for learning latent representations of songs, artists and users [5, 11, 25].

A diverse set of information has been used for music recommendation, such as song metadata (e.g., song title, artist name, era, genre, albums etc.) [16, 28], content data (e.g., lyrics, low level audio features etc.) [5, 17, 24, 25], artists information (e.g., artist popularity, collocation of artists etc.) [5, 7], user listening history and social interactions (e.g., social tags) as well as usage statistics (e.g., song popularity, song co-occurrence etc.) [5, 7, 16, 17, 25]. There are a few work that make use of latent representations of song, user and artist which are learned from existing playlists, or user-song and user-artist interactions [5, 11].

The sequential order of songs in playlist has not been well understood [33], some work suggest that song order and song-to-song transitions are important for playlist quality [18, 25], while other work have shown that the ensemble of songs in playlist do matter, but the song order seems to be negligible [36, 38]. In this work, we treat a playlist as a set of songs by discarding the sequential order, and leave the investigation of using song order to assist playlist generation as future work.

It has been known that binary classification and bipartite ranking are closely related [14, 26]. In particular, Ertekin and Rudin [14] have shown that the P-Norm Push bipartite ranking loss [31] is equivalent to the P-Classification loss [14] when using the exponential surrogate. Further, the P-Norm Push loss is an approximation of the Infinite-Push loss [2], or equivalently, the Top-Push loss [20], which focuses on the highest ranked negative example instead of of the lowest ranked positive example as in (4).

Inspired by these connections, we seek a classification loss that is equivalent to a bipartite ranking loss (under a few assumptions), which can approximate the risk with Bottom-Push loss in (4). This will make it possible to formulate an unconstrained objective that approximates the empirical loss $\mathcal{R}_{\text{RANK}}$.

5 EXPERIMENTS

In this section, we describe experiments that empirically evaluate the proposed method and a number of well known baseline approaches in the three cold-start settings described in Section 2.

5.1 Dataset

We make use of two publicly available playlist datasets: the 30Music [37] and AotM-2011 [25] playlist dataset. The Million Song Dataset [6] serves as an underlying dataset where all songs in playlists are intersected with, and it also provides a few song features which we will detail later.

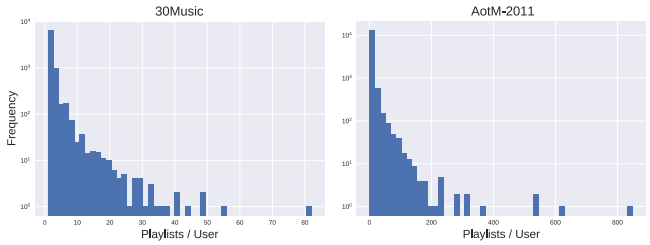

Figure 2: Histogram of the number of playlists per user

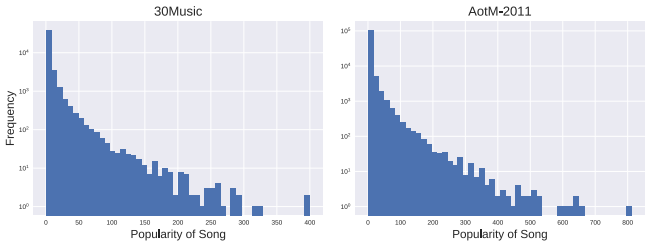

Figure 3: Histogram of song popularity

Million Song Dataset (MSD) is a collection of one million songs, in which information of each song such as the name, artist, year of release are available. It also provides acoustic features computed from a sample section of audio file of each song.

30Music Dataset is a collection of listening events and playlists retrieved from Last.fm[1]. We utilise the playlists data by first intersecting with the MSD, leveraging the Last.fm dataset [13] which matched songs from Last.fm with those in MSD, then filtering out playlists with less than 5 songs. This results in roughly 17K playlists over 45K songs from 8K users.

AotM-2011 Dataset is a collection of playlists shared by users[2] ranging from 1998 to 2011, where songs in the dataset have been matched to those in the Million Song Dataset (MSD). We filtered out playlists with less than 5 songs, which results in roughly 84K playlists over 114K songs from 14K users.

Table 1 summarises the two playlist datasets used in this work. The histograms of the number of playlists per user as well as song popularity (i.e., the number of occurrences of a song in all playlists) of the two datasets are shown in Figure 2 and Figure 3, respectively. We can see from Figure 2 and 3 that both the number of playlists per user and song popularity follow a long-tail distribution, which imposes further challenge to the learning task as the amount of data for learning is limited for users (or songs) at the tail.

5.2 Experimental setup

We describe how playlist data are split into training and test set for all three cold-start settings, the features of songs used to learn the multitask objective, as well as baseline approaches and evaluation metrics.

Dataset split. To empirically evaluate the performance of recommending new songs to extend existing playlists (setting (i)), we hold 5K of the latest released songs in 30Music dataset, which results in about 8K playlists with songs in both training and test set. In the larger AotM-2011 dataset, we hold 10K of the latest released songs, which leads to about 19K playlists with songs in both training and test set. Playlists where all songs have been held are removed from both training and test set. Table 2 summarises the statistics of this training/test split.

For the task of recommending songs to form playlists for existing users (setting (ii)), we hold playlists from about 20% users in both datasets for test, and all other playlists are used for training. To make sure each song in the test set also appears in the training set, and all users in test set also have a few playlists in training set, playlists in the test set are sampled from users that have at least one playlist where each song has also been included in four other playlists among the whole dataset, which results in a test set with about 2.1K playlists from 1.6K users in 30Music dataset, and a test set with about 9.2K playlists from 2.7K users in AotM-2011 dataset. The statistics of this training/test split are shown in Table 3.

To evaluate the last cold-start setting (iii), i.e., recommending songs to form playlist for new users, we sampled 30% of all users and hold all their playlists as test set in both datasets. Similarly, to make sure songs in test set also exist in the training set, a user will not be included in the test set if holding all of her playlists breaks this requirement. This results in a test set with about 3.4K playlists from 2.4K users in 30Music dataset, and a test set with about 8.2K playlists from 4.2K users in AotM-2011 dataset. Table 4 describes the statistics of this training/test split.

Features. Song features used in the experiments including metadata, audio data, genre and artist information, and song/artist popularity. The metadata of songs (e.g., duration, year of release) and audio features (e.g., loudness, mode, tempo) are provided by MSD. We use genre data from the Top-MAGD genre dataset [34] and tagtraum genre annotations for MSD [35] via one-hot encoding. If the genre data of a song is not available, we apply the mean imputation using genre counts of other songs in training set. To encode artist information as features, we trained a word2vec[3] model using sequences of artist identifiers in playlists.

In the task of recommending new songs to extend existing playlists (setting (i)), where song popularity is not available, we use artist popularity (we use the number of occurrences of all songs from a artist in training playlists as a proxy of her popularity). Finally, we add a constant feature (with value 1) for each song to account for bias.

Baselines. We compare the performance of our proposed approach (referred as *Multitask Classification*) with a number of baseline methods for the task of recommending music to form playlists:

[1]https://www.last.fm
[2]http://www.artofthemix.org

Table 1: Music playlist dataset

Dataset	Songs	Playlists	Users	Songs/Playlist	Playlists/User
30Music	45,468	17,457	8,070	16.3	2.2
AotM-2011	114,428	84,710	14,182	10.1	6.0

[3]https://github.com/dav/word2vec

Table 2: Dataset for setting (i)

Dataset	Songs		Playlists	
	Train	Test	Train	Test
30Music	40,468	5,000	17,342	8,215
AotM-2011	104,428	10,000	84,646	19,504

Table 3: Dataset for setting (ii)

Dataset	Playlists		Users	
	Train	Test	Train	Test
30Music	15,262	2,195	8,070	1,644
AotM-2011	75,477	9,233	14,182	2,722

Table 4: Dataset for setting (iii)

Dataset	Playlists		Users	
	Train	Test	Train	Test
30Music	14,067	3,390	5,649	2,421
AotM-2011	76,450	8,260	9,928	4,254

- The *Popularity Ranking* method scores each song using only its popularity in training set. When song popularity is not available (setting (i)), we use artist popularity.
- The *Same Artists - Greatest Hits* (SAGH) [23] method scores scores each song by its popularity if the artist of the song has appeared in the given user's playlist in training set; otherwise the song is scored zero. Similarly, we replace song popularity with artist popularity in the task of recommending new songs (setting (i)).
- The *Collocated Artists - Greatest Hits* (CAGH) [7] method is a variant of SAGH. It scores each song using its popularity, but weighted by the frequency of the collocation between the artist of the song and those artists that appeared in the given user's playlist in training set. In the task of recommending new songs (setting (i)), we again replace the song popularity in this method with artist popularity. When recommending songs for new users (setting (iii)), we use the top 10 most popular artists instead of artists in the user's listening history as required by this method.
- The *Logistic Regression* baseline is specific for the task of recommending new songs (setting (i)), where we independenly train an logistic regression classifier for each playlist, which is used to classify whether we should add each new song to this playlist. This method is also known as binary relevance in multi-label classification.

Evaluation. We evaluate the performance of all approaches using two metrics that are commonly employed in playlist recommendation tasks: Hit-Rate@K [16] and Area under the ROC curve (AUC) [22]. Hit-Rate@K (or Hit Ratio) is the number of correctly recommended songs among the top-K recommendations over L, where K is the number of recommendations, and L is the number of songs in the ground truth playlist. It has been employed to evaluate several playlist generation and next song recommendation methods [7, 8, 16, 17]. This metric is also known as Recall@K [33]. The area under the ROC curve (AUC) is widely used in measuring performance of classifiers, it has also been used for evaluating performance of playlist generation method when the task is cast as a (sequence of) classification problems [5].

5.3 Results and discussion

We can see from Table 5 that learning based methods (i.e., **Multitask classification** and *Logistic Regression* perform significantly better than other baselines, in the setting of recommending new songs for extending playlists (setting (i)). Which suggests learning from existing playlists helps the task of recommending new songs. Among the (artist) popularity based approaches, simply ranking songs by its artist popularity achieves decent performance, and take into account artist collocation information does not seem to improve the performance. Further, the SAGH method which considers only songs from the top 10 most popular artists, is the worst performer in terms of HitRate@100 and AUC.

Figure 4 shows the hit rates when the number of recommendations K varies from 0 to 1000. We can see that the performance of all methods improves as the number of recommendations increase, and learning based methods are always performing better than ranking based artist informations. It is interesting that both *Multitask Classification* and *Logistic Regression* perform almost the same when K is small (100 on 30Music and 50 on AotM-2011 dataset), and the former only performance better when K is big than that. Another interesting observation is that SAGH and CAGH performs similar on AotM-2011 dataset when K is less 50, and both outperforms *Popularity Ranking*, however, SAGH suffer from slow improvement as K increases, this effect is observed on both datasets.

Table 5: Performance in setting (i)

Method	30Music		AotM-2011	
	HitRate@100 %	AUC %	HitRate@100 %	AUC %
SAGH	4.8	51.6	7.8	53.6
CAGH	10.9	69.2	11.9	77.5
Popularity Ranking	12.2	70.9	8.7	76.5
Logistic Regression	35.4	82.5	23.1	81.0
Multitask Classification	**36.2**	**86.6**	**24.4**	**84.3**

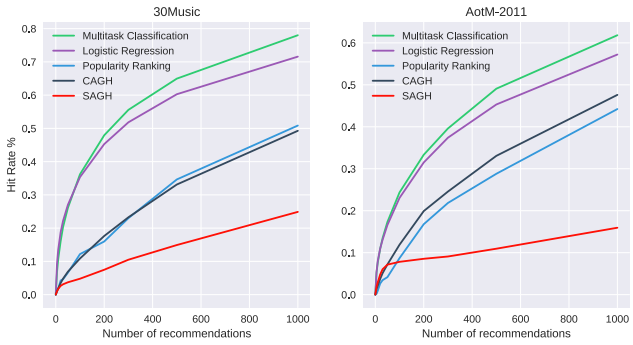

Figure 4: Hit rates in setting (i)

Table 6 and Figure 5 show the performance on the task of recommending a set of songs for an existing user (setting (ii)). First, we can see that methods based on song popularity and artist information achieve good performance, which is in line with discoveries in [7, 8, 17] which show ranking based on popularity is a strong baseline. Further, methods which make use of both song popularity and artist information, outperform *Popularity Ranking* (except SAGH on 30Music), which shows artist information is helpful for

Table 6: Performance in setting (ii)

Method	30Music		AotM-2011	
	HitRate@100 %	AUC %	HitRate@100 %	AUC %
SAGH	18.1	64.4	9.2	79.6
CAGH	19.3	95.2	7.7	94.2
Popularity Ranking	12.0	94.0	5.8	93.8
Multitask Classification	**20.8**	**95.9**	**9.9**	**95.4**

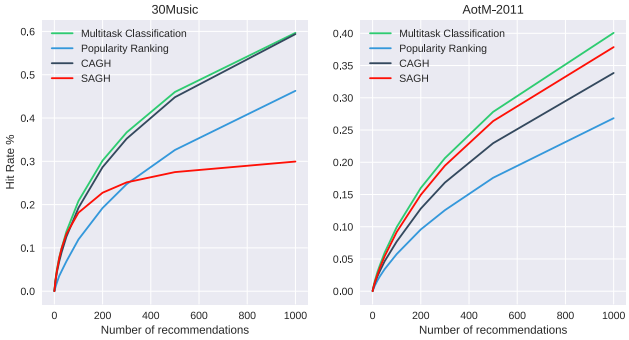

Figure 5: Hit rates in setting (ii)

Table 7: Performance in setting (iii)

Method	30Music		AotM-2011	
	HitRate@100 %	AUC %	HitRate@100 %	AUC %
SAGH	7.3	54.5	5.0	53.7
CAGH	9.7	87.3	5.8	89.4
Popularity Ranking	**11.0**	88.3	7.4	**91.8**
Multitask Classification	**11.0**	**88.6**	**7.5**	**91.8**

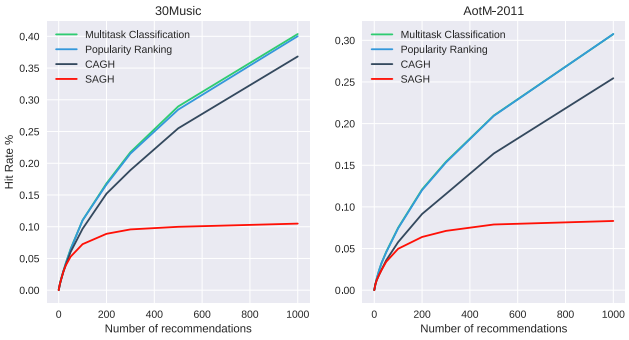

Figure 6: Hit rates in setting (iii)

music recommendation, which is consistent with results reported in [7, 8]. It is interesting to observe that SAGH is one of the best performer on AotM-2011 dataset (Figure 5), while it performs the worst in terms of HitRate@K on 30Music dataset, when K is greater than 300.

Finally, the performance in the setting of recommending music to form playlists for new users (setting (iii)), however, is quite different from the other two cold-start settings. We can see from Table 7 and Figure 6 that *Popularity Ranking* is one of the best performing methods, outperforming both baselines that make use of song popularity and artist information (i.e., SAGH and CAGH). The performance of *Multitask Classification* is almost identical to that of *Popularity Ranking* in terms of all metrics, which suggests *Multitask Classification* might degenerate to simply rank songs using its popularity. Another interesting observation is that, although both SAGH and CAGH make use of song popularity and artist information, the later performs significantly better than the former in terms of AUC and HitRate@K (when K is greater than 100). This suggests the way to make use of artist information matters, and collocation of artists could be more helpful for music recommendation than simply filtering out songs that are not from popular artists.

Last but not the least, our proposed method *Multitask Classification*, unlike any other method in consideration, has always been one of the best performer in all three cold-start settings on both datasets, Which suggests our propose approach can work effectively for the cold-start settings considered in this paper. As a remark, the performance of all methods in terms of HitRate@K improve as the number of recommendations increase, and the rate of improvement decreases as more songs are recommended. In particular, the performance of SAGH tends to saturate much earlier than other methods (except in setting (ii) on AotM-2011 dataset), which might suggest that, as more songs are recommended, artists in users' listening history (or popular artists) become less informative, and other factors such as song popularity regains its dominance for the task of music recommendation.

6 CONCLUSION

We propose a flexible multitask learning method to deal with all three settings. The method learns from user-curated playlists, and encourages songs in the playlist to be ranked higher than those are not by minimising a bipartite ranking loss. We formulate the objective as a constrained convex optimisation problem, and show how this may be approximated by an unconstrained objective inspired by an equivalence relationship between bipartite ranking and binary classification. Empirical results on two real music playlist datasets show the proposed approach has good performance for playlist recommendation in cold-start settings.

We investigate the problem of recommending playlists to users in cold-start settings. In the cold songs setting, we recommend newly released songs to extend existing playlists; in the cold playlists setting, we recommend a set of songs to form a new playlist for an existing user; and in the cold users setting, we recommend a set of songs to form a new playlist for a new user. We deal with all three settings using a multitask learning method which encourages songs in playlist to be ranked higher than those are not by minimising a bipartite ranking loss. We formulate the objective as a constrained convex optimisation problem, and further approximates it by an unconstrained objective inspired by an equivalence relationship between bipartite ranking and binary classification. Empirical results on two real music playlist datasets show the proposed approach has good performance for playlist recommendation in cold-start settings.

We are aware of a few limitations of the proposed approach, which we leave as future work. Specifically, additional data sources (e.g., music information shared on social media) or song/user features (e.g., lyrics, user profile), as well as the sequential order of

songs which could provide additional information to help make better recommendations. Further, non-linear models such as deep neural networks have shown strong performance in a wide arrange of tasks, and the linear model with sparse parameters in this work could potentially be more compact if non-linear objective were employed.

Finally, as a remark, we want to mention the challenge of evaluating the recommended results. While metrics in information retrieval are commonly used, recommender system is more like a generative process than a information retrieval task. Fortunately, this challenge has been noticed and been attacked in many ways [24, 25, 33], we believe that promising automatic evaluation methods that accepted by the (majority of) community is one premise of significant progress in music recommendation.

REFERENCES

[1] Deepak Agarwal and Bee-Chung Chen. 2009. Regression-based Latent Factor Models. In *KDD*. 10.

[2] Shivani Agarwal. 2011. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 839–850.

[3] Galen Andrew and Jianfeng Gao. 2007. Scalable training of L 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*. ACM, 33–40.

[4] Mordecai Avriel. 2003. *Nonlinear programming: analysis and methods*. Courier Corporation.

[5] Shay Ben-Elazar, Gal Lavee, Noam Koenigstein, Oren Barkan, Hilik Berezin, Ulrich Paquet, and Tal Zaccai. 2017. Groove Radio: A Bayesian Hierarchical Model for Personalized Playlist Generation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 445–453.

[6] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

[7] Geoffray Bonnin and Dietmar Jannach. 2013. Evaluating the quality of playlists based on hand-crafted samples. In *ISMIR*. 263–268.

[8] Geoffray Bonnin and Dietmar Jannach. 2015. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)* 47, 2 (2015), 26.

[9] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[10] Bin Cao, Nathan N. Liu, and Qiang Yang. 2010. Transfer Learning for Collective Link Prediction in Multiple Heterogenous Domains. In *ICML*.

[11] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 714–722.

[12] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 39–46.

[13] Last.fm dataset. 2011. The official song tags and song similarity collection for the Million Song Dataset. http://labrosa.ee.columbia.edu/millionsong/lastfm.

[14] Şeyda Ertekin and Cynthia Rudin. 2011. On equivalence relationships between classification and ranking algorithms. *Journal of Machine Learning Research* 12, Oct (2011), 2905–2929.

[15] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. 2010. Learning Attribute-to-Feature Mappings for Cold-Start Recommendations. In *ICDM*. 10.

[16] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the 6th ACM Conference on Recommender Systems*. ACM, 131–138.

[17] Dietmar Jannach, Lukas Lerche, and Iman Kamehkhosh. 2015. Beyond hitting the hits: Generating coherent music playlist continuations with the right tracks. In *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, 187–194.

[18] Iman Kamehkhosh, Dietmar Jannach, and Geoffray Bonnin. 2018. How Automated Recommendations Affect the Playlist Creation Behavior of Users. (2018).

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[20] Nan Li, Rong Jin, and Zhi-Hua Zhou. 2014. Top rank optimization in linear time. In *Advances in neural information processing systems*. 1502–1510.

[21] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. 2008. SoRec: Social Recommendation Using Probabilistic Matrix Factorization. In *CIKM*. 10.

[22] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[23] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. 2012. The Million Song Dataset Challenge. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 909–916.

[24] Brian McFee and Gert RG Lanckriet. 2011. The Natural Language of Playlists. In *ISMIR*, Vol. 11. 537–542.

[25] Brian McFee and Gert RG Lanckriet. 2012. Hypergraph Models of Playlist Dialects. In *ISMIR*, Vol. 12. 343–348.

[26] Aditya Krishna Menon and Robert C Williamson. 2016. Bipartite ranking: a risk-theoretic perspective. *Journal of Machine Learning Research* 17, 195 (2016), 1–102.

[27] Netflix. 2006. Netflix Prize. http://www.netflixprize.com/.

[28] John C Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. 2002. Learning a gaussian process prior for automatically generating music playlists. In *Advances in neural information processing systems*. 1425–1432.

[29] RecSys. 2018. ACM RecSys Challenge. http://www.recsyschallenge.com/2018/.

[30] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[31] Cynthia Rudin. 2009. The P-Norm Push: A simple convex ranking algorithm that concentrates at the top of the list. *Journal of Machine Learning Research* 10, Oct (2009), 2233–2271.

[32] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms. In *WWW '01*. 285–295.

[33] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi. 2017. Current Challenges and Visions in Music Recommender Systems Research. *ArXiv e-prints* (2017). https://arxiv.org/abs/1710.03208

[34] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. 2012. Facilitating Comprehensive Benchmarking Experiments on the Million Song Dataset. In *ISMIR*. 469–474.

[35] Hendrik Schreiber. 2015. Improving Genre Annotations for the Million Song Dataset. In *ISMIR*. 241–247.

[36] Nava Tintarev, Christoph Lofi, and Cynthia Liem. 2017. Sequences of Diverse Song Recommendations: An exploratory study in a commercial system. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*. ACM, 391–392.

[37] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music Listening and Playlists Dataset. In *RecSys Posters*.

[38] Andreu Vall, Markus Schedl, Gerhard Widmer, Massimo Quadrana, and Paolo Cremonesi. 2017. The Importance of Song Context in Music Playlists. In *Proceedings of the Poster Track of the 11th ACM Conference on Recommender Systems*.