Adversarial Mahalanobis Distance-based Attentive Song Recommender for Automatic Playlist Continuation

Thanh Tran, Renee Sweeney, Kyumin Lee
Department of Computer Science
Worcester Polytechnic Institute
Massachusetts, USA
{tdtran,rasweeney,kmlee}@wpi.edu

ABSTRACT

In this paper, we aim to solve the *automatic playlist continuation* (APC) problem by modeling complex interactions among users, playlists, and songs using only their interaction data. Prior methods mainly rely on dot product to account for similarities, which is not ideal as dot product is not metric learning, so it does not convey the important inequality property. Based on this observation, we propose three novel deep learning approaches that utilize Mahalanobis distance. Our first approach uses user-playlist-song interactions, and combines Mahalanobis distance scores between (i) a target user and a target song, and (ii) between a target playlist and the target song to account for both the user's preference and the playlist's theme. Our second approach measures song-song similarities by considering Mahalanobis distance scores between the target song and each member song (i.e., existing song) in the target playlist. The contribution of each distance score is measured by our proposed *memory metric-based attention mechanism*. In the third approach, we fuse the two previous models into a unified model to further enhance their performance. In addition, we adopt and customize *Adversarial Personalized Ranking* (APR) for our three approaches to further improve their robustness and predictive capabilities. Through extensive experiments, we show that our proposed models outperform eight state-of-the-art models in two large-scale real-world datasets.

ACM Reference Format:
Thanh Tran, Renee Sweeney, Kyumin Lee. 2019. Adversarial Mahalanobis Distance-based Attentive Song Recommender for Automatic Playlist Continuation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331234

1 INTRODUCTION

The *automatic playlist continuation* (*APC*) problem has received increased attention among researchers following the growth of online music streaming services such as Spotify, Apple Music, SoundCloud, *etc.* Given a user-created playlist of songs, *APC* aims to recommend one or more songs that fit the user's preference and match the playlist's theme.

Due to the inconsistency of available side information in public music playlist datasets, we first attempt to solve the *APC* problem using only interaction data. *Collaborative filtering* (CF) methods, which encode users, playlists, and songs in lower-dimensional latent spaces, have been widely used [14, 15, 18, 24]. To account for the extra playlist dimension in this work, the term *item* in the context of *APC* will refer to a song, and the term *user* will refer to either a user or playlist, depending on the model. We will also use the term *member song* to denote an existing song within a target playlist. CF solutions to the *APC* problem can be classified into the following three groups:

Group 1: Recommending songs that are directly relevant to user/playlist taste. Methods in this group aim to measure the conditional probability of a target item given a target user using user-item interactions. In *APC*, this is either $P(s|u)$ – the conditional probability of a target song s given a target user u by taking users and songs within their playlists as implicit feedback input –, or $P(s|p)$ – the conditional probability of a target song s given a target playlist p by utilizing playlist-song interactions. Most of the works in this group measure $P(s|u)$[1] by taking the dot product of the user and song latent vectors [5, 15, 18], denoted by $P(s|u) \propto \overrightarrow{u}^T \cdot \overrightarrow{s}$. With the recent success of deep learning approaches, researchers proposed neural network-based models [14, 27, 58, 62].

Despite their high performance, *Group 1* is limited for the *APC* task. First, although a target song can be very similar to the ones in a target playlist [9, 38], this association information is ignored. Second, these approaches only measure either $P(s|u)$ or $P(s|p)$, which is sub-optimal. $P(s|u)$ omits the playlist theme, causing the model to recommend the same songs for different playlists, and $P(s|p)$ is not personalized for each user.

Group 2: Recommending songs that are similar to existing songs in a playlist. Methods in this group are based on a principle that similar users prefer the same items (user neighborhood design), or users themselves prefer similar items (item neighborhood design). *ItemKNN*[42], *SLIM*[35], and *FISM*[21] solve the *APC* problem by identifying similar users/playlists or similar songs. These works are limited in that they give equal weight to each song in a playlist when calculating their similarity to a candidate song. In reality, certain aspects of a member song, such as the genre or artist, may be more important to a user when deciding whether or not to add

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331234

[1] Measuring $P(s|p)$ is easily obtained by replacing the user latent vector u with the playlist latent vector p.

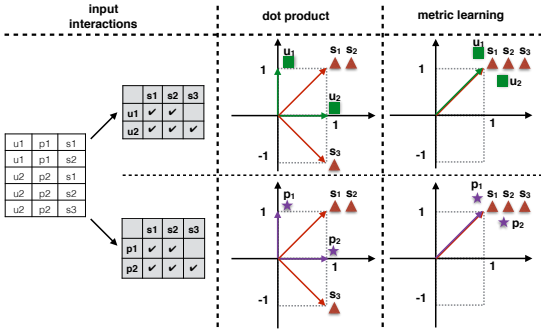


Figure 1: Learning with dot product vs. metric learning.

another song into the playlist. This calls for differing song weights, which are produced by attentive neighborhood-based models.

Recently, [7] proposed a *Collaborative Memory Network* (CMN) that considers both the target user preference on a target item as well as similar users' preferences on that item (i.e., user neighborhood hybrid design). It utilizes a memory network to assign different attentive contributions of neighboring users. However, this approach still does not work well with *APC* datasets due to sparsity issues and less associations among users.

Group 3: Recommending next songs as transitions from the previous songs. Methods in this group are called *sequential recommendation* models, which rely on Markov Chains to capture sequential patterns [41, 54]. In the *APC* domain, these methods make recommendations based on the order of songs added to a playlist. Deep learning-based sequential methods are able to model even more complex transitional patterns using convolutional neural networks (CNNs) [45] or recurrent neural networks (RNNs) [6, 16, 20]. However, *Sequential recommenders* have restrictions in the *APC* domain, namely that playlists are often listened to on shuffle. It means that users typically add songs based on an overarching playlist theme, rather than song transition quality. In addition, added song timestamps may not be available in music datasets.

Motivation. A common drawback of the works listed in the three groups above is that they rely on the dot product to measure similarity. Dot product is not metric learning, so it does not convey the crucial inequality property [17, 39], and does not handle differently scaled input variables well. We illustrate the drawback of dot product in a toy example shown in Figure 1[2], where the latent dimension is size $d = 2$. Assume we have two users u_1, u_2, two playlists p_1, p_2, and three songs s_1, s_2, s_3. We can see that p_1 and p_2 (or u_1 and u_2) are similar (i.e., both liked s_1, and s_2), suggesting that s_3 would be relevant to the playlist p_1. Learning with dot product can lead to the following result: $p_1 = (0, 1), p_2 = (1, 0), s_1 = (1, 1), s_2 = (1, 1), s_3 = (1, -1)$, because $p_1^T s_1 = 1, p_1^T s_2 = 1, p_2^T s_1 = 1, p_2^T s_2 = 1$, $p_2^T s_3$=1 (same for users u_1, u_2). However, the dot product between p_1 and s_3 is -1, so s_3 would not be recommended to p_1. However, if we use metric learning, it will pull similar users/playlists/songs closer together by using the inequality property. In the example, the distance between p_1 and s_3 is rescaled to 0, and s_3 is now correctly portrayed as a good fit for p_1.

There exist several works that adopt metric learning for recommendation. [17] proposed *Collaborative Metric Learning* (CML) which used Euclidean distance to pull positive items closer to a user and push negative items further away. [4, 8, 11] also used Euclidean distance but for modeling transitional patterns. However, these metric-based models still fall into either *Group 1* or *Group 3*, inheriting the limitations that we described previously. Furthermore, as Euclidean distance is the primary metric, these models are highly sensitive to the scales of (latent) dimensions/variables.

Our approaches and main contributions. According to the literature, Mahalanobis distance[3] [57, 59] overcomes the drawback (i.e., high sensitivity) of Euclidean distance. However, Mahalanobis distance has not yet been applied to recommendation with neural network designs.

By overcoming the limitations of existing recommendation models, we propose three novel deep learning approaches in this paper that utilize Mahalanobis distance. Our first approach, *Mahalanobis Distance Based Recommender* (MDR), belongs to *Group 1*. Instead of modeling either $P(s|p)$ or $P(s|u)$, it measures $P(s|u, p)$. To combine both a user' preference and a playlist' theme, *MDR* measures and combines the Mahalanobis distance between the target user and target song, and between the target playlist and target song. Our second approach, *Mahalanobis distance-based Attentive Song Similarity recommender* (MASS), falls into *Group 2*. Unlike the prior works, *MASS* uses Mahalanobis distance to measure similarities between a target song and member songs in the playlist. MASS incorporates our proposed *memory metric-based attention mechanism* that assigns attentive weights to each distance score between the target song and each member song in order to capture different influence levels. Our third approach, *Mahalanobis distance based Attentive Song Recommender* (MASR), combines *MDR* and *MASS* to merge their capabilities. In addition, we incorporate customized *Adversarial Personalized Ranking* [13] into our three models to further improve their robustness.

We summarize our contributions as follows:

- We propose three deep learning approaches (*MDR*, *MASS*, and *MASR*) that fully exploit Mahalanobis distance to tackle the *APC* task. As a part of *MASS*, we propose the memory metric-based attention mechanism.
- We improve the robustness of our models by applying *adversarial personalized ranking* and customizing it with a flexible noise magnitude.
- We conduct extensive experiments on two large-scale *APC* datasets to show the effectiveness and efficiency of our approaches.

2 OTHER RELATED WORK

Music recommendation literature has frequently made use of available metadata such as: lyrics [33], tags [19, 28, 33, 50, 51], audio features [28, 33, 50, 51, 56], audio spectrograms [36, 52, 55], song/artist/playlist names [1, 20, 22, 37, 46], and Twitter data [19]. Deep learning and hybrid approaches have made significant progress against traditional collaborative filtering music recommenders [36, 50, 51, 55]. [56] uses multi-arm bandit reinforcement learning for interactive music recommendation by leveraging novelty and music audio content. [28] and [52] perform weighted matrix factorization

[2]This Figure is inspired by [17]

[3]https://en.wikipedia.org/wiki/Mahalanobis_distance

using latent features pre-trained on a CNN, with song tags and Mel-frequency cepstral coefficients (MFCCs) as input, respectively. Unlike these works, our proposed approaches do not require or incorporate side information.

Recently, attention mechanisms have shown their effectiveness in various machine learning tasks including document classification [61], machine translation [3, 29], recommendation [30, 31], *etc.* So far, several attention mechanisms are proposed such as: *general attention* [29], *dot* attention [29], *concat* attention [3, 29], hierarchical attention [61], *scaled dot attention* and *multi-head* attention [53], *etc.*. However, to our best of knowledge, most of previously proposed attention mechanisms leveraged dot product for measuring similarities which is not optimal in our Mahalanobis distance-based recommendation approaches because of the difference between dot product space and metric space. Therefore, we propose a memory metric-based attention mechanism for our models' designs.

3 PROBLEM DEFINITION

Let $U = \{u_1, u_2, u_3, ..., u_m\}$ denote the set of all users, $P = \{p_1, p_2, p_3, ..., p_n\}$ denote the set of all playlists, $S = \{s_1, s_2, s_3, ..., s_v\}$ denote the set of all songs. Bolded versions of these variables, which we will introduce in the following sections, denote their respective embeddings. *m, n, v* are the number of users, playlists, and songs in a dataset, respectively. Each user $u_i \in U$ has created a set of playlists $T^{(u_i)} = \{p_1, p_2, ..., p_{|T^{(u_i)}|}\}$, where each playlist $p_j \in T^{(u_i)}$ contains a list of songs $T^{(p_j)} = \{s_1, s_2, ..., s_{|T^{(p_j)}|}\}$. Note that $T^{(u_1)} \cup T^{(u_2)} \cup ... \cup T^{(u_m)} = \{p_1, p_2, p_3, ..., p_n\}$, $T^{(p_1)} \cup T^{(p_2)} \cup ... \cup T^{(p_n)} = \{s_1, s_2, s_3, ..., s_v\}$, and the song order within each playlist is often not available in the dataset. The *Automatic Playlist Continuity (APC)* problem can then be defined as recommending new songs $s_k \notin T^{(p_j)}$ for each playlist $p_j \in T^{(u_i)}$ created by user u_i.

4 MAHALANOBIS DISTANCE PRELIMINARY

Given two points $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$, the Mahalanobis distance between x and y is defined as:

$$d_M(x, y) = \|x - y\|_M = \sqrt{(x - y)^T M (x - y)} \tag{1}$$

where $M \in \mathbb{R}^{d \times d}$ parameterizes the Mahalanobis distance metric to be learned during model training. To ensure that Eq. (1) produces a mathematical metric[4], M must be symmetric positive semi-definite ($M \succeq 0$). This constraint on M makes the model training process more complicated, so to ease this condition, we rewrite $M = A^T A$ ($A \in \mathbb{R}^{d \times d}$) since $M \succeq 0$. The Mahalanobis distance between two points $d_M(x, y)$ now becomes:

$$\begin{aligned} d_M(x, y) = \|x - y\|_A &= \sqrt{(x - y)^T A^T A (x - y)} \\ &= \sqrt{\Big(A(x - y)\Big)^T \Big(A(x - y)\Big)} \\ &= \|A(x - y)\|_2 = \|Ax - Ay\|_2 \end{aligned} \tag{2}$$

where $\|\cdot\|_2$ refers to the Euclidean distance. By rewriting Eq. (1) into Eq. (2), the Mahalanobis distance can now be computed by measuring the Euclidean distance between two linearly transformed points $x \rightarrow Ax$ and $y \rightarrow Ay$. This transformed space encourages the model to learn a more accurate similarity between x and y.

[4]https://en.wikipedia.org/wiki/Metric_(mathematics)

$d_M(x, y)$ is generalized to basic Euclidean distance $d(x, y)$ when A is the identity matrix. If A in Eq. (2) is a diagonal matrix, the objective becomes learning metric A such that different dimensions are assigned different weights. Our experiments show that learning diagonal matrix A generalizes well and produces slightly better performance than if A were a full matrix. Therefore in this paper we focus on only the diagonal case. Also note that when A is diagonal, we can rewrite Eq. (2) as:

$$d_M(x, y) = \|A(x - y)\|_2 = \|diag(A) \odot (x - y)\|_2 \tag{3}$$

where $diag(A) \in \mathbb{R}^n$ returns the diagonal of matrix A, and $\odot$ denotes the element-wise product. Therefore, we can parameterize $B = diag(A) \in \mathbb{R}^n$ and learn the Mahalanobis distance by simply computing $\|B \odot (x - y)\|_2$.

In our models' calculations, we will adopt squared Mahalanobis distance, since quadratic form promotes faster learning.

5 OUR PROPOSED MODELS

In this section, we delve into design elements and parameter estimation of our three proposed models: *Mahalanobis Distance based Recommender (MDR), Mahalanobis distance-based Attentive Song Similarity recommender (MASS),* and the combined model *Mahalanobis distance based Attentive Song Recommender (MASR).*

5.1 Mahalanobis Distance based Recommender (MDR)

As mentioned in Section 1, *MDR* belongs to the *Group 1. MDR* takes a target user, a target playlist, and a target song as inputs, and outputs a distance score reflecting the direct relevance of the target song to the target user's music taste and to the target playlist's theme. We will first describe how to measure each of the conditional probabilities – $P(s_k|u_i)$, $P(s_k|p_j)$, and finally $P(s_k|u_i, p_j)$ – using Mahalanobis distance. Then we will go over *MDR*'s design.

5.1.1 Measuring $P(s_k|u_i)$. Given a target user u_i, a target playlist p_j, a target song s_k, and the Mahalanobis distance $d_M(u_i, s_k)$ between u_i and s_k, $P(s_k|u_i)$ is measured by:

$$P(s_k|u_i) = \frac{\exp(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}}))}{\sum_l \exp(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}}))} \tag{4}$$

where β_{s_k}, β_{s_l} are bias terms to capture their respective song's overall popularity [23]. User bias is not included in Eq.(4) because it is independent of $P(s_k|u_i)$ when varying candidate song s_k. The denominator $\sum_l \exp(-d_M(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}})$ is a normalization term shared among all candidate songs. Thus, $P(s_k|u_i)$ is measured as:

$$P(s_k|u_i) \propto -\Big(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}}\Big) \tag{5}$$

Note that training with Bayesian Personalized Ranking (BPR) will only require calculating Eq. (5), since for every pair of observed song k^+ and unobserved song k^- , we model the pairwise ranking $P(s_{k^+}|u_i) > P(s_{k^-}|u_i)$. Using Eq. (4), this inequality is satisfied only if $d_M^2(\boldsymbol{u_i}, \boldsymbol{s_{k^+}}) + \beta_{s_{k^+}} < d_M^2(\boldsymbol{u_i}, \boldsymbol{s_{k^-}}) + \beta_{s_{k^-}}$, which leads to Eq. (5).

5.1.2 Measuring $P(s_k|p_j)$. Given a target playlist p_j, a target song s_k, and the Mahalanobis distance $d_M(p_j, s_k)$ between p_j and s_k, $P(s_k|p_j)$ is measured by:

$$P(s_k|p_j) = \frac{\exp(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma_{s_k}}))}{\sum_l \exp(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_l}) + \boldsymbol{\gamma_{s_l}}))} \tag{6}$$

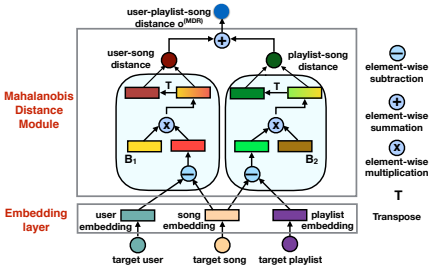


Figure 2: Architecture of our MDR.

where γ_{s_k} and γ_{s_l} are song bias terms. Similar to $P(s_k|u_i)$, we shortly measure $P(s_k|p_j)$ by:

$$P(s_k|p_j) \propto -(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma_{s_k}}) \tag{7}$$

5.1.3 Measuring $P(s_k|u_i, p_j)$. $P(s_k|u_i, p_j)$ is computed using the Bayesian rule under the assumption that u_i and p_j are conditionally independent given s_k:

$$\begin{aligned} P(s_k|u_i, p_j) &\propto P(u_i|s_k)P(p_j|s_k)P(s_k) \\ &= \frac{P(s_k|u_i)P(u_i)}{P(s_k)} \frac{P(s_k|p_j)P(p_j)}{P(s_k)} P(s_k) \\ &\propto P(s_k|u_i)P(s_k|p_j)\frac{1}{P(s_k)} \end{aligned} \tag{8}$$

In Eq. (8), $P(s_k)$ represents the popularity of target song s_k among the song pool. For simplicity in this paper, we assume that selecting a random candidate song follows a uniform distribution instead of modeling this popularity information. $P(s_k|u_i, p_j)$ then becomes proportional to: $P(s_k|u_i, p_j) \propto P(s_k|u_i)P(s_k|p_j)$. Using Eq. (4, 6), we can approximate $P(s_k|u_i, p_j)$ as follows:

$$\begin{aligned} &P(s_k|u_i, p_j) \propto \\ &\frac{\exp\left(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}})\right)}{\sum_l \exp\left(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}})\right)} \times \frac{\exp\left(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma s_k})\right)}{\sum_l \exp\left(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_l}) + \boldsymbol{\gamma s_l})\right)} \\ =&\frac{\exp\left(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}}) - (d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma_{s_k}})\right)}{\sum_l \exp\left(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}})\right) \sum_{l'} \exp\left(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_{l'}}) + \boldsymbol{\gamma_{s_{l'}}})\right)} \end{aligned} \tag{9}$$

Since the denominator of Eq. (9) is shared by all candidate songs (i.e., normalization term), we can shortly measure $P(s_k|u_i, p_j)$ by:

$$\begin{aligned} P(s_k|u_i, p_j) &\propto -\left(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k})\right) - \left(\boldsymbol{\beta_{s_k}} + \boldsymbol{\gamma_{s_k}}\right) \\ &= -\left(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\theta_{s_k}}\right) \end{aligned} \tag{10}$$

With $P(s_k|u_i, p_j)$ now established in Eq. (10), we can move on to our *MDR* model.

5.1.4 MDR Design. The *MDR* architecture is depicted in Figure 2. It has an Input, Embedding Layer, and Mahalanobis Distance Module.

Input: *MDR* takes a target user u_i (user ID), a target playlist p_j (playlist ID), and a target song s_k (song ID) as input.

Embedding Layer: *MDR* maintains three embedding matrices of users, playlists, and songs. By passing user u_i, playlist p_j, and song s_k through the embedding layer, we obtain their respective embedding vectors $\boldsymbol{u_i} \in \mathbb{R}^d$, $\boldsymbol{p_j} \in \mathbb{R}^d$, and $\boldsymbol{s_k} \in \mathbb{R}^d$, where d is the embedding size.

Mahalanobis Distance Module: As depicted in Figure 2, this module outputs a distance score $o^{(MDR)}$ that indicates the relevance of candidate song s_k to both user u_i's music preference and playlist

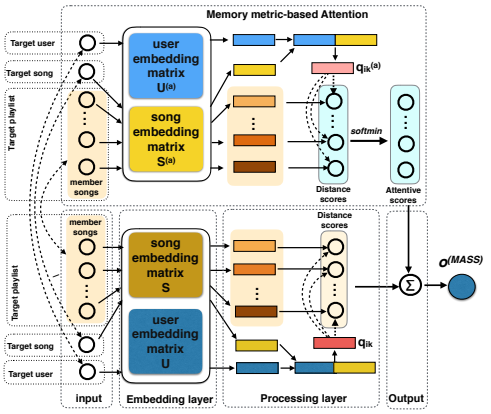


Figure 3: Architecture of our MASS.

p_j's theme. Intuitively, the lower the distance score is, the more relevant the song is. $o^{(MDR)}(\boldsymbol{u_i}, \boldsymbol{p_j}, \boldsymbol{s_k})$ is computed as follows:

$$o^{(MDR)} = o(\boldsymbol{u_i}, \boldsymbol{s_k}) + o(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\theta_{s_k}} \tag{11}$$

where $\boldsymbol{\theta_{s_k}}$ is song s_k's bias, and $o(\boldsymbol{u_i}, \boldsymbol{s_k}), o(\boldsymbol{p_j}, \boldsymbol{s_k})$ are quadratic Mahalanobis distance scores between user u_i and song s_k, and between playlist p_j and song s_k, shown in the following two equations. $\boldsymbol{B_1} \in \mathbb{R}^d$ and $\boldsymbol{B_2} \in \mathbb{R}^d$ are two metric learning vectors. And,

$$o(\boldsymbol{u_i}, \boldsymbol{s_k}) = \left(\boldsymbol{B_1} \odot (\boldsymbol{u_i} - \boldsymbol{s_k})\right)^T \left(\boldsymbol{B_1} \odot (\boldsymbol{u_i} - \boldsymbol{s_k})\right)$$

$$o(\boldsymbol{p_j}, \boldsymbol{s_k}) = \left(\boldsymbol{B_2} \odot (\boldsymbol{p_j} - \boldsymbol{s_k})\right)^T \left(\boldsymbol{B_2} \odot (\boldsymbol{p_j} - \boldsymbol{s_k})\right)$$

5.2 Mahalanobis distance-based Attentive Song Similarity recommender (MASS)

As stated in the Section 1, *MASS* belongs to *Group 2*, where it measures attentive similarities between the target song and member songs in the target playlist. An overview of *MASS*'s architecture is depicted in Figure 3. *MASS* has five components: Input, Embedding Layer, Processing Layer, Attention Layer, and Output.

5.2.1 Input: The inputs to our *MASS* model include a target user u_i, a candidate song s_k for a target playlist p_j, and a list of l member songs within the playlist, where l is the number of songs in the largest playlist (i.e., containing the largest number of songs) in the dataset. If a playlist contains less than l songs, we pad the list with *zeroes* until it reaches length l.

5.2.2 Embedding Layer: This layer holds two embedding matrices: a user embedding matrix $\mathbf{U} \in \mathbb{R}^{m \times d}$ and a song embedding matrix $\mathbf{S} \in \mathbb{R}^{v \times d}$. By passing the input target user u_i and target song s_k through these two respective matrices, we obtain their embedding vectors $\boldsymbol{u_i} \in \mathbb{R}^d$ and $\boldsymbol{s_k} \in \mathbb{R}^d$. Similarly, we acquire the embedding vectors for all l member songs in p_j, denoted by $\boldsymbol{s_1}, \boldsymbol{s_2}, ..., \boldsymbol{s_l}$.

5.2.3 Processing Layer: We first need to consolidate $\boldsymbol{u_i}$ and $\boldsymbol{s_k}$. Following widely adopted deep multimodal network designs [43], we concatenate the two embeddings, and then transform them into a new vector $\boldsymbol{q}_{ik} \in \mathbb{R}^d$ via a fully connected layer with weight matrix $\boldsymbol{W}_1 \in \mathbb{R}^{2d \times d}$, bias term $\boldsymbol{b} \in \mathbb{R}$, and activation function ReLU. We formulate this process as follows:

$$\boldsymbol{q}_{ik} = \text{ReLU}\left(\mathbf{W}_1 \begin{bmatrix} \boldsymbol{u_i} \\ \boldsymbol{s_k} \end{bmatrix} + \boldsymbol{b}_1\right) \tag{12}$$

Note that $\boldsymbol{q}_{ik}$ can be interpreted as a search query in QA systems [2, 60]. Since we combined the target user u_i with the query song

s_k (to add to the user's target playlist), the search query $\boldsymbol{q}_{ik}$ is personalized. The ReLU activation function models a non-linear combination between these two target entities, and was chosen over *sigmoid* or *tanh* due to its encouragement of sparse activations and proven non-saturation [10], which helps prevent overfitting.

Next, given the embedding vectors $\boldsymbol{s_1}, \boldsymbol{s_2}, ..., \boldsymbol{s_l}$ of the l member songs in target playlist p_j, we approximate the conditional probability $P(s_k|u_i, s_1, s_2, ..., s_l)$ by:

$$P(s_k|u_i, s_1, s_2, ..., s_l) \propto -\left(\sum_{t=1}^{l} \boldsymbol{\alpha_{ikt}} d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) + \boldsymbol{b_{s_k}} \right) \quad (13)$$

where $d_M(\cdot)$ returns the Mahalanobis distance between two vectors, b_{s_k} is the song bias reflecting its overall popularity, and $\boldsymbol{\alpha}_{ikt}$ is the attention score to weight the contribution of the partial distance between search query $\boldsymbol{q_{ik}}$ and member song $\boldsymbol{s_t}$. We will show how to calculate $d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t})$ below, and $\boldsymbol{\alpha_{ikt}}$ in *Attention Layer* at 5.2.4.

As indicated in Eq. (3), we parameterize $\boldsymbol{B_3} \in \mathbb{R}^d$, which will be learned during the training phase. The Mahalanobis distance between the search query $\boldsymbol{q_{ik}}$ and each member song $\boldsymbol{s_t}$, treating $\boldsymbol{B_3}$ as an edge-weight vector, is measured by:

$$d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) = \left\| \boldsymbol{e}_{\boldsymbol{ikt}}^T \boldsymbol{e_{ikt}} \right\|_2^2 \quad \text{where} \quad \boldsymbol{e_{ikt}} = \boldsymbol{B_3} \odot (\boldsymbol{q_{ik}} - \boldsymbol{s_t}) \quad (14)$$

Calculating Eq. (14) for every member song $\boldsymbol{s_t}$ yields the following l-dimensional vector:

$$\begin{bmatrix} d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_1) \\ d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_2) \\ \cdots \\ d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_l) \end{bmatrix} = \begin{bmatrix} \left\| \boldsymbol{e}_{ik1}^T \boldsymbol{e}_{ik1} \right\|_2^2 \\ \left\| \boldsymbol{e}_{ik2}^T \boldsymbol{e}_{ik2} \right\|_2^2 \\ \cdots \\ \left\| \boldsymbol{e}_{ikl}^T \boldsymbol{e}_{ikl} \right\|_2^2 \end{bmatrix} \quad (15)$$

Note that $\boldsymbol{B_3}$ is shared across all Mahalanobis measurement pairs. Now we go into detail of how to calculate the attention weights $\boldsymbol{\alpha_{ikt}}$ using our proposed Attention Layer.

5.2.4 Attention Layer: With l distance scores obtained in Eq. (15), we need to combine them into one distance value to reflect how relevant the target song is *w.r.t* the target playlist's member songs. The simplest approach is to follow the well-known item similarity design [21, 35] where the same weights are assigned for all l distance scores. This is sub-optimal in our domain because different member song can relate to the target song differently. For example, given a country playlist and a target song of the same genre, the member songs that share the same artist with the target song would be more similar to the target song than the other member songs in the playlist. To address this concern, we propose a novel *memory metric-based attention mechanism* to properly allocate different attentive scores to the distance values in Eq. (15). Compared to existing attention mechanisms, our attention mechanism maintains its own embedding memory of users and songs (i.e., memory-based property), which can function as an external memory. It also computes attentive scores using Mahalanobis distance (i.e., metric-based property) instead of traditional dot product. Note that the memory-based property is also commonly applied to question-answering in NLP, where memory networks have utilized external memory [44] for better memorization of context information [25, 34]. Our attention mechanism has one external memory containing user and song embedding matrices. When the user and song embedding matrices of our attention mechanism are identical to those in the embedding layer, it is the same as looking up the embedding vectors of target users, target songs, and member songs in the embedding layer (Section 5.2.2). Therefore, using external memory will make room for more flexibility in our models.

The attention layer features an external user embedding matrix $\mathbf{U^{(a)}} \in \mathbb{R}^{m \times d}$ and external song embedding matrix $\mathbf{S^{(a)}} \in \mathbb{R}^{v \times d}$. Given the following inputs – a target user u_i, a target song s_k, and all l member songs in playlist p_j – by passing them through the corresponding embedding matrices, we obtain the embedding vectors of u_i, s_k, and all the member songs, denoted as $\boldsymbol{u_i^{(a)}}$, $\boldsymbol{s_k^{(a)}}$, and $\boldsymbol{s_1^{(a)}}, \boldsymbol{s_2^{(a)}}, ..., \boldsymbol{s_l^{(a)}}$, respectively.

We then forge a personalized search query $\boldsymbol{q_{ik}^{(a)}}$ by combining $\boldsymbol{u_i^{(a)}}$ and $\boldsymbol{s_k^{(a)}}$ in a multimodal design as follows:

$$\boldsymbol{q_{ik}^{(a)}} = \text{ReLU}\left(\mathbf{W}_2 \begin{bmatrix} \boldsymbol{u_i^{(a)}} \\ \boldsymbol{s_k^{(a)}} \end{bmatrix} + \boldsymbol{b}_2 \right) \quad (16)$$

where $\mathbf{W}_2 \in \mathbb{R}^{2d \times d}$ is a weight matrix and $\boldsymbol{b}_2$ is a bias term. Next, we measure the Mahalanobis distance (with an edge weight vector $\boldsymbol{B_4} \in \mathbb{R}^d$) from $\boldsymbol{q_{ik}^{(a)}}$ to a member song's embedding vector $\boldsymbol{s_t^{(a)}}$ where $t \in \overline{1, l}$:

$$d_M^2(\boldsymbol{q_{ik}^{(a)}}, \boldsymbol{s_t^{(a)}}) = \left\| \left(\boldsymbol{e_{ikt}^{(a)}} \right)^T \boldsymbol{e_{ikt}^{(a)}} \right\|_2^2 \quad \text{where} \quad \boldsymbol{e_{ikt}^{(a)}} = \boldsymbol{B_4} \odot \left(\boldsymbol{q_{ik}^{(a)}} - \boldsymbol{s_t^{(a)}} \right) \quad (17)$$

Using Eq. (17), we generate l distance scores between each of l member songs and the candidate song. Then we apply *softmin* on l distance scores in order to obtain the member songs' attentive scores[5]. Intuitively, the lower the distance between a search query and a member song vector, the higher its contribution level is *w.r.t* the candidate song.

$$\boldsymbol{\alpha_{ikt}} = \frac{\exp\left(- \left\| \left(\boldsymbol{e_{ikt}^{(a)}} \right)^T \boldsymbol{e_{ikt}^{(a)}} \right\|_2^2 \right)}{\sum_{t'=1}^{l} \exp\left(- \left\| \left(\boldsymbol{e_{ikt'}^{(a)}} \right)^T \boldsymbol{e_{ikt'}^{(a)}} \right\|_2^2 \right)} \quad (18)$$

5.2.5 Output: We output the total attentive distances $o^{(MASS)}$ from the target song s_k to target playlist p_j's existing songs by:

$$\mathbf{o}^{(\text{MASS})} = -\left(\sum_{t=1}^{l} \boldsymbol{\alpha_{ikt}} d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) + \boldsymbol{b_{s_k}} \right) \quad (19)$$

where $\boldsymbol{\alpha_{ikt}}$ is the attentive score from Eq. (18), $d_M(\boldsymbol{q_{ik}}, \boldsymbol{s_t})$ is the personalized Mahalanobis distance between target song s_k and a member song s_t in user u_i's playlist (Eq. (15)), $\boldsymbol{b_{s_k}}$ is the song bias.

5.3 Mahalanobis distance based Attentive Song Recommender (MASR = MDR + MASS)

We enhance our performance on the *APC* problem by combining our *MDR* and *MASS* into a *Mahalanobis distance based Attentive Song Recommender (MASR)* model. *MASR* outputs a cumulative distance score from the outputs of *MDR* and *MASS* as follows:

$$\mathbf{o}^{(\text{MASR})} = \alpha \mathbf{o}^{(\text{MDR})} + (1 - \alpha) \mathbf{o}^{(\text{MASS})} \quad (20)$$

where $\mathbf{o}^{(\text{MDR})}$ is from Eq. (11), $\mathbf{o}^{(\text{MASS})}$ is from Eq. (19), and $\alpha \in [0, 1]$ is a hyperparameter to adjust the contribution levels of *MDR* and *MASS*. α can be tuned using a development dataset. However, in the following experiments, we set $\alpha = 0.5$ to receive equal contribution from *MDR* and *MASS*. We pretrain *MDR* and *MASS* first, then fix *MDR* and *MASS*'s parameters in *MASR*. There are two benefits

[5] Note that attentive scores of padded items are 0.

of this design. First, if MASR is learnable with pretrained MDR and MASS initialization, MASR would have too high a computational cost to train. Second, by making *MASR* non-learnable, *MDR* and *MASS* in *MASR* can be trained separately and in parallel, which is more practical and efficient for real-world systems.

5.4 Parameter Estimation

5.4.1 Learning with Bayesian Personalized Ranking (BPR) loss. We apply BPR loss as an objective function to train our *MDR, MASS, MASR* as follows:

$$\mathcal{L}(\mathcal{D}|\Theta) = \text{argmin}_{\Theta}\Big(-\sum_{(i,j,k^+,k^-)} \log \sigma(\mathbf{o}_{ijk^-} - \mathbf{o}_{ijk^+}) + \lambda_{\Theta}\|\Theta\|^2 \Big) \quad (21)$$

where (i, j, k^+, k^-) is a quartet of a target user, a target playlist, a positive song, and a negative song which is randomly sampled. $\sigma(\cdot)$ is the *sigmoid* function; $\mathcal{D}$ denotes all training instances; Θ are the model's parameters (for instance, $\Theta = \{\mathbf{U}, \mathbf{S}, \mathbf{U^{(a)}}, \mathbf{S^{(a)}}, \mathbf{W_1}, \mathbf{W_2}, \mathbf{B_3}, \mathbf{B_4}, \mathbf{b}\}$ in the *MASS* model); λ_{Θ} is a regularization hyper-parameter; and $\mathbf{o_{ijk}}$ is the output of either *MDR*, *MASS*, or *MASR*, which is measured in Eq. (11), (19), and (20), respectively.

5.4.2 Learning with Adversarial Personalized Ranking (APR) loss. It has been shown in [13] that BPR loss is vulnerable to adversarial noise, and *APR* was proposed to enhance the robustness of a simple matrix factorization model. In this work, we apply *APR* to further improve the robustness of our *MDR*, *MASS*, and *MASR*. We name our *MDR*, *MASS*, and *MASR* trained with *APR* loss as *AMDR*, *AMASS*, *AMASR* by adding an "adversarial (A)" term, respectively. Denote δ as adversarial noise on the model's parameters Θ. The *BPR* loss from adding adversarial noise δ to Θ is defined by:

$$\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta) = \underset{\Theta=\hat{\Theta}+\delta}{\text{argmax}}\Big(-\sum_{(i,j,k^+,k^-)} \log \sigma(\mathbf{o}_{ijk^-} - \mathbf{o}_{ijk^+}) \Big) \quad (22)$$

where $\hat{\Theta}$ is optimized in Eq. (21) and fixed as constants in Eq. (22). Then, training with *APR* aims to play a minimax game as follows:

$$\arg\min_{\Theta} \max_{\delta, \|\delta\| \le \epsilon s(\hat{\Theta})} \mathcal{L}(\mathcal{D}|\Theta) + \lambda_{\delta}\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta) \quad (23)$$

where ϵ is a hyper-parameter to control the magnitude of perturbations δ. In [13], the authors fixed ϵ for all the model's parameters, which is not ideal because different parameters can endure different levels of perturbation. If we add too large adversarial noise, the model's performance will downgrade, while adding too small noise does not guarantee more robust models. Hence, we multiply ϵ with the standard deviation $s(\hat{\Theta})$ of the targeting parameter $\hat{\Theta}$ to provide a more flexible noise magnitude. For instance, the adversarial noise magnitude on parameter $\mathbf{B_3}$ in *AMASS* model is $\epsilon \times s(\mathbf{B_3})$. If the values in $\mathbf{B_3}$ are widely dispersed, they are more vulnerable to attack, so the adversarial noise applied during training must be higher in order to improve robustness. Whereas if the values are centralized, they are already robust, so only a small noise magnitude is needed.

Learning with *APR* follows 4 steps: **Step 1**: unlike [13] where parameters are saved at the last training epoch, which can be overfitted parameter values (e.g. some thousands of epoches for matrix factorization in [13]), we first learn our models' parameters by minimizing Eq. (21) and save the best checkpoint based on evaluating on a development dataset. **Step 2:** with optimal $\hat{\Theta}$ learned in *Step 1*, in Eq. (22), we set $\Theta = \hat{\Theta}$ and fix Θ to learn δ. **Step 3:** with optimal $\hat{\delta}$ learned in Eq. (22), in Eq. (23) we set $\delta = \hat{\delta}$ and fix δ to learn new values for Θ. **Step 4**: We repeat *Step 2* and *Step 3* until a maximum number of epochs is reached and save the best checkpoint based on evaluation on a development dataset. Following [13, 26], the update rule for δ is obtained by using the fast gradient method as follows:

$$\delta = \epsilon \times s(\hat{\Theta}) \times \frac{\nabla_{\delta}(\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta))}{\left\|\nabla_{\delta}(\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta))\right\|_2} \quad (24)$$

Note that update rules of parameters in Θ can be easily obtained by computing the partial derivative *w.r.t* each parameter in Θ.

5.5 Time Complexity

Let Ω denote the total number of training instances ($= \sum_j N(p_j)$ where $N(p_j)$ refers to the number of songs in training playlist p_j). $\omega = max(N(p_j)), \forall j = \overline{1, n}$ denotes the maximum number of songs in all playlists. For each forward pass, *MDR* takes $O(d)$ to measure $\mathbf{o}^{(MDR)}$ (in Eq. (11)) for a positive training instance, and another forward pass with $O(d)$ to calculate $\mathbf{o}^{(MDR)}$ for a negative instance. The backpropagation for updating parameters take the same complexity. Therefore, the time complexity of *MDR* is $O(\Omega d)$. Similarly, for each positive training instance, *MASS* takes (i) $O(2d^2)$ to make each query in Eq. (12) and Eq. (16); (ii) $O(\omega d)$ to calculate ω distance scores from ω member songs to the target song in Eq. (15); and (iii) $O(\omega d)$ to measure attention scores in Eq. (18). Since embedding size d is often small, $O(\omega d)$ is a dominant term and *MASS*'s time complexity is $O(\Omega\omega d)$. Hence, both *MDR* and *MASS* scale linearly to the number of training instances and can run very fast, especially with sparse datasets. When training with *APR*, updating δ in Eq. (24) with fixed $\hat{\Theta}$ needs one forward and one backward pass. Learning Θ in Eq. (23) requires one forward pass to measure $\mathcal{L}(\mathcal{D}|\Theta)$ in Eq. (21), one forward pass to measure $\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta)$ in Eq. (22), and one backward pass to update Θ in Eq. (23). Hence, time complexity when training with *APR* is h times higher (h is small) compared to training with *BPR* loss.

6 EMPIRICAL STUDY

6.1 Datasets

To evaluate our proposed models and existing baselines, we used two publicly accessible real-world datasets that contain user, playlist, and song information. They are described as follows:

- 30Music [49]: This is a collection of playlists data retrieved from Internet radio stations through Last.fm[6]. It consists of 57K playlists and 466K songs from 15K users.
- AOTM [32]: This dataset was collected from the Art of the Mix[7] playlist database. It consists of 101K playlists and 504K songs from 16K users, spanning from Jan 1998 to June 2011.

For data preprocessing, we removed duplicate songs in playlists. Then we adopted a widely used *k-core* preprocessing step [12, 47] (with *k-core* = 5), filtering out playlists with less than 5 songs. We also removed users with an extremely large number of playlists, and extremely large playlists (i.e., containing thousands of songs). Since the datasets did not have song order information for playlists (i.e., which song was added to a playlist first, then next, and so on), we randomly shuffled the song order of each playlist and used it in

[6] https://www.last.fm
[7] http://www.artofthemix.org/

Table 1: Statistics of datasets.

Statistics	30Music	AOTM
# of users	12,336	15,835
# of playlists	32,140	99,903
# of songs	276,142	504,283
# of interactions	666,788	1,966,795
avg. # of playlists per user	2.6	6.3
avg. & max # of songs per playlist	18.75 & 63	17.69 & 58
Density	0.008%	0.004%

the sequential recommendation baseline models to compare with our models. The two datasets are implicit feedback datasets. The statistics of the preprocessed datasets are presented in Table 1.

6.2 Baselines

We compared our proposed models with **eight** strong state-of-the-art models in the *APC* task. The baselines were trained by using *BPR* loss for a fair comparison:

- **Bayesian Personalized Ranking (MF-BPR)** [40]: It is a pairwise matrix factorization method for implicit feedback datasets.
- **Collaborative Metric Learning (CML)** [17]: It is a collaborative metric-based method. It adopted Euclidean distance to measure a user's preference on items.
- **Neural Collaborative Filtering (NeuMF++)** [14]: It is a neural network based method that models non-linear user-item interactions. We pretrained two components of NeuMF to obtain its best performance (i.e., NeuMF++).
- **Factored Item Similarity Methods (FISM)** [21]: It is a item neighborhood-based method. It ranks a candidate song based on its similarity with member songs using dot product.
- **Collaborative Memory Network (CMN++)** [7]: It is a user-neighborhood based model using a memory network to assign attentive scores for similar users.
- **Personalized Ranking Metric Embedding (PRME)** [8]: It is a sequential recommender that models a personalized first-order Markov behavior using Euclidean distance.
- **Translation-based Recommendation (Transrec)** [11]: It is one of the best sequential recommendation methods. It models the third order between the user, the previous song, and the next song where the user acts as a translator.
- **Convolutional Sequence Embedding Recommendation (Caser)** [45]: It is a CNN based sequential recommendation. It embeds a sequence of recent songs into an "image" in time and latent spaces, then learns sequential patterns as local features of the image using different horizontal and vertical filters.

We did not compare our models with baselines that performed worse than above listed baselines like *item-KNN*[42], *SLIM*[35], *etc.*

MF-BPR, CML, and NeuMF++ used only user/playlist-song interaction data to model either users' preferences over songs $P(s|u)$ or playlists' tastes over songs $P(s|p)$. We ran the baselines both ways, and report the best results. Two neighborhood-based baselines utilized neighbor users/playlists (i.e., CMN++) or member songs (i.e., FISM) to recommend the next song based on user/playlist similarities or song similarities (i.e., measure $P(s|u, s_1, s_2, ..., s_l)$ and $P(s|p, s_1, s_2, ..., s_l)$, of which we report the best results).

Table 2: Performance of the baselines, and our models. The last two lines show the relative improvement of MASR and AMASR compared to the best baseline.

	Methods	30Music		AOTM	
		hit@10	ndcg@10	hit@10	ndcg@10
(a)	MF-BPR	0.450	0.315	0.699	0.473
(b)	CML	0.600	0.452	0.735	0.481
(c)	NeuMF++	0.623	0.461	0.741	0.498
(d)	FISM	0.544	0.346	0.686	0.446
(e)	CMN++	0.536	0.397	0.722	0.505
(f)	PRME	0.426	0.260	0.570	0.354
(g)	Transrec	0.570	0.417	0.710	0.450
(h)	Caser	0.458	0.289	0.681	0.448
Ours	MDR	0.705	0.524	0.820	0.631
	MASS	0.670	0.500	0.834	0.639
	MASR	**0.731**	**0.564**	**0.854**	**0.654**
	AMDR	0.764	0.581	0.850	0.658
	AMASS	0.753	0.581	0.856	0.659
	AMASR	**0.785**	**0.604**	**0.874**	**0.677**
Imprv. (%)	MASR	**+17.34**	**+22.34**	**+13.36**	**+28.24**
	AMASR	**+26.00**	**+31.02**	**+17.95**	**+34.19**

6.3 Experimental Settings

Protocol: We use the widely adopted *leave-one-out* evaluation setting [14]. Since both the 30Music and AOTM datasets do not contain timestamps of added songs for each playlist, we randomly sample two songs per playlist–one for a positive test sample, and one for a development set to tune hyper-parameters–while the remaining songs in each playlist make up the training set. We follow [14, 48] and uniformly random sample 100 non-member songs as negative songs, and rank the test song against those negative songs.

Evaluation metrics: We evaluate the performance of the models with two widely used metrics: Hit Ratio (*hit@N*), and Normalized Discounted Cumulative Gain (*NDCG@N*). The *hit@N* measures whether the test item is in the recommended list or not, while the *NDCG@N* takes into account the position of the *hit* and assigns higher scores to hits at top-rank positions. For the test set, we measure both metrics and report the average scores.

Hyper-parameters settings: Models are trained with the *Adam* optimizer with learning rates from {0.001, 0.0001}, regularization terms λ_Θ from {0, 0.1, 0.01, 0.001, 0.0001}, and embedding sizes from {8, 16, 32, 64}. The maximum number of epochs is 50, and the batch size is 256. The number of hops in *CMN++* are selected from {1, 2, 3, 4}. In *NeuMF++*, the number of MLP layers are selected from {1, 2, 3}. The number of negative samples per one positive instance is 4, similar to [14]. The Markov order L in *Caser* is selected from {4, 5, 6, 7, 8, 9, 10}. For *APR* training, the number of *APR* training epochs is 50, the noise magnitude ϵ is selected from {0.5, 1.0}, and the adversarial regularization λ_δ is set to 1, as suggested in [13]. Adversarial noise is added only in training process, and are initialized as *zero*. All hyper-parameters are tuned by using the development set. Our source code is available at *https://github.com/thanhdtran/MASR.git*.

6.4 Experimental Results

6.4.1 Performance comparison. Table 2 shows the performance of our proposed models and baselines on each dataset. *MDR* and baselines (a)-(c) are in *Group 1*, but *MDR* shows much better performance

Table 3: Performance of variants of our MDR and MASS. RI indicates relative average improvement over the corresponding method.

Methods	30Music		AOTM		RI(%)
	hit@10	ndcg@10	hit@10	ndcg@10	
MDR_us	0.684	0.500	0.815	0.594	**+3.68**
MDR_ps	0.654	0.476	0.746	0.547	**+10.79**
MDR_ups (i.e., MDR)	**0.705**	**0.524**	**0.818**	**0.613**	
MASS_ups	0.651	0.479	0.789	0.581	**+4.12**
MASS_ps	0.621	0.450	0.764	0.523	**+10.82**
MASS_us (i.e., MASS)	**0.670**	**0.500**	**0.820**	**0.631**	

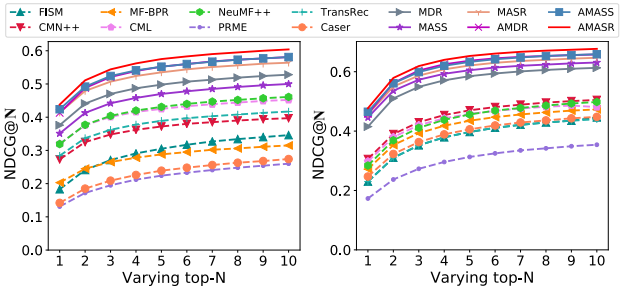
compared to the (a)-(c) baselines, improving at least 11.14% *hit@10* and 18.81% *NDCG@10* on average. *CML* simply adopts Euclidean distance between users/playlists and positive songs, but has nearly equal performance with NeuMF++, which utilizes a neural network to learn non-linear relationships between users/playlists and songs. This result shows the effectiveness of using metric learning over dot product in recommendation. *MDR* outperforms *CML* by 19.04% on average. This confirms the effectiveness of Mahalanobis distance over Euclidian distance for recommendation.

MASS outperforms both *FISM* and *CMN++*, improving *hit@10* by 18.4%, and *NDCG@10* by 25.5% on average. This is because *FISM* does not consider the attentive contribution of different neighbors. Even though *CMN++* can assign attention scores for different user/playlist neighbors, it bears the flaws of *Group 1* by considering only either neighbor users or neighbor playlists. More importantly, *MASS* uses a novel attentive metric design, while dot product is utilized in *FISM* and *CMN++*. Sequential models, (f)-(h) baselines, do not work well. In particular, *MASS* outperforms the (f)-(h) baselines, improving 24.6% on average compared to the best model in (f)-(h).

MASR outperforms both *MDR* and *MASS*, indicating the effectiveness of fusing them into one model. Particularly, *MASR* improves *MDR* by 5.0%, and *MASS* by 6.7% on average. Performances of *MDR, MASS, MASR* are boosted when adopting *APR* loss with a *flexible* noise magnitude. *AMDR* improves *MDR* by 7.7%, *AMASS* improves *MASS* by 9.4%, and *AMASR* improves *MASR* by 5.8%. We also compare our *flexible* noise magnitude with a fixed noise magnitude used in [13] by varying the fixed noise magnitude in {0.5, 1.0} and setting $\lambda_\delta = 1$. We observe that *APR* with a *flexible* noise magnitude performs better with an average improvement of 7.53%.

Next, we build variants of our *MDR* and *MASS* models by removing either playlist or user embeddings, or using both of them. Table 3 presents an ablation study of exploiting playlist embeddings. *MDR_us* is the *MDR* that uses only user-song interactions (i.e., ignore playlist-song distance $o(p_j, s_k)$ in Eq. (11)). *MDR_ps* is the *MDR* that uses only playlist-song interactions (i.e., ignores user-song distance $o(u_i, s_k)$ in Eq. (11)). *MDR_ups* is our proposed *MDR* model. Similarly, *MASS_ups* is the *MASS* model but considers both user-song distances and playlist-song distances in its design. The *Embedding Layer* and *Attention Layer* of *MASS_ups* have additional playlist embedding matrices $\mathbf{P} \in \mathbb{R}^{n \times d}$ and $\mathbf{P}^{(\mathbf{a})} \in \mathbb{R}^{n \times d}$, respectively. *MASS_ps* is the *MASS* model that replaces user embeddings with playlist embeddings. *MASS_us* is our proposed *MASS* model.

MDR (i.e., *MDR_ups*) outperforms its derived forms (*MDR_us* and *MDR_ps*), improving by 3.7~10.8% on average. This result shows



(a) 30Music. (b) AOTM.

Figure 4: Performance of our models and the baselines when varying *N* (or *top-N* recommendation list) from [1, 10].

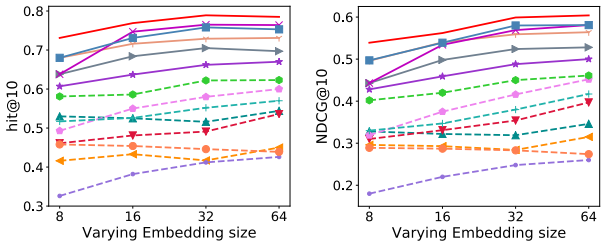


Figure 5: Performance of all models when varying the embedding size *d* from {8, 16, 32, 64} in 30Music dataset.

the effectiveness of modeling both users' preferences and playlists' themes in *MDR* design. *MASS* (i.e., *MASS_us*) outperforms its two variants (*MASS_ups* and *MASS_ps*), improving *MASS_ups* by 3.7%, and *MASS_ps* by 10.8% on average. It makes sense that using additional playlist embeddings in *MASS_ups* is redundant since the member songs have already conveyed the playlist's theme, and ignoring user embeddings in *MASS_ps* neglects user preferences.

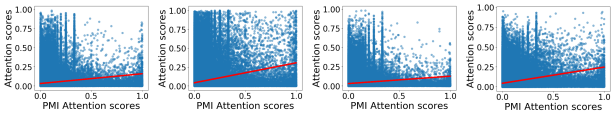
6.4.2 Varying top-N recommendation list and embedding size. Figure 4 shows performances of all models when varying *top-N* recommendation from 1 to 10. We see that all models gain higher results when increasing *top-N*, and all our proposed models outperform all baselines across all *top-N* values. On average, *MASR* improves 26.3%, and *AMASR* improves 33.9% over the best baseline's performance.

Figure 5[8] shows all models' performances when varying the embedding size d from {8, 16, 32, 64} for the *30Music* dataset. Note that the *AOTM* dataset also shows similar results but is omitted due to the space limitations. We observe that most models tend to have increased performance when increasing embedding size. *AMDR* does not improve *MDR* when $d = 8$ but does so when increasing d. This phenomenon was also reported in [13] because when $d = 8$, *MDR* is too simple and has a small number of parameters, which is far from overfitting the data and not very vulnerable to adversarial noise. However, for more complicated models like *MASS* and *MASR*, even with a small embedding size $d = 8$, *APR* shows its effectiveness in making the models more robust, and leads to an improvement of *AMASS* by 12.0% over *MASS*, and an improvement of *AMASR* by 7.5% over *MASR*. The improvements of *AMDR, AMASS, AMASR* over their corresponding base models are higher for larger d due to the increase of model complexity.

[8]Figure 5 shares the same legend with Figure 4 for saving space.

Table 4: Performance of MASS using various attention mechanisms.

Attention Types	30Music		AOTM		RI(%)
	hit@10	ndcg@10	hit@10	ndcg@10	
non-mem + dot	0.630	0.454	0.785	0.574	**+8.51**
non-mem + metric	0.660	0.490	0.803	0.601	**+3.43**
mem + dot	0.659	0.475	0.791	0.585	**+5.40**
mem + metric	**0.670**	**0.500**	**0.834**	**0.639**	



(a) ρ=0.153 **(b) ρ=0.215** **(c) ρ=0.171** **(d) ρ=0.254**

Figure 6: Scatter plots of PMI attention scores vs. attention weights learned by various attention mechanisms, showing corresponding Pearson correlation score ρ). (a)non-mem + dot, (b)non-mem + metric, (c)mem + dot, (d)mem + metric.

6.4.3 Is our memory metric-based attention helpful? To answer this question, we evaluate how *MASS*'s performance changed when varying its attention mechanism as follows:

- *non-memory + dot product* (*non-mem + dot*): It is the popular *dot attention* introduced in [29].
- *non-memory + metric* (*non-mem + metric*): It is our proposed attention with Mahalanobis distance but no external memory.
- *memory + dot product* (*mem + dot*): It is the *dot attention* but exploiting external memory.
- *memory + metric* (*mem + metric*): It is our proposed attention mechanism.

We do not compare with the *no-attention* case because literature has already proved the effectiveness of the attention mechanism [53]. Table 4 shows the performance of *MASS* under the variations of our proposed attention mechanism. We have some key observations. First, *non-mem + metric* attention outperforms *non-mem + dot* attention with an improvement of 4.9% on average. Similarly, *mem + metric* attention improves the *mem + dot* attention design by 5.4% on average. This enhancement comes from different nature of metric space and dot product space. Moreover, these results confirm that metric-based attention designs fit better into our proposed Mahalanobis distance based model. Second, *memory* based attention works better than *non-mem* attention. Particularly, on average, *mem + dot* improves *non-mem + dot* by 2.98%, and *mem + metric* improves *non-mem + metric* by 3.43%. Overall, the performance order is *mem + metric* > *non-mem + metric* > *mem + dot* > *non-mem + dot*, which confirms that our proposed attention performs the best and improves 3.43~8.51% compared to its variations.

6.4.4 Deep analysis on attention. To further understand how attention mechanisms work, we connect attentive scores generated by attention mechanisms with *Pointwise Mutual Information* scores. Given a target song k and a member song t, the *PMI* score between them is defined as: $PMI(k,t) = log\frac{P(k,t)}{P(k)\times P(t)}$. Here, *PMI*(k,t) score indicates how likely two songs k and t co-occur together, or how likely a target song k will be added into song t's playlist.

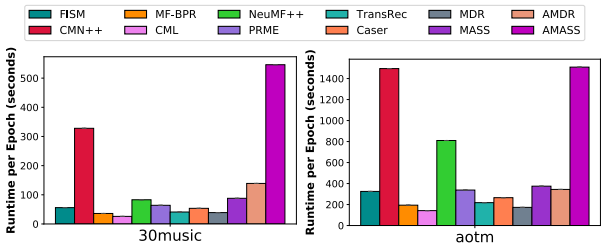


Figure 7: Runtime of all models in 30Music and AOTM.

Given a playlist that has a set of l member songs, we measure *PMI* scores between the target song k and each of l member songs. Then, we apply $softmax$ to those *PMI* scores to obtain *PMI attentive scores*. Intuitively, the member song t that has a higher *PMI* score with candidate song k (i.e., co-occurs more with song k) will have a higher *PMI attentive score*. We draw scatter plots between *PMI attentive scores* and attentive scores generated by our proposed attention mechanism and its variations. Figure 6 shows the experimental results. We observe that the Pearson correlation ρ between the *PMI attentive scores* and the attentive scores generated by our attention mechanism is the highest (0.254). This result shows that our proposed attention tends to give higher scores to co-occurred songs, which is what we desire. The Pearson correlation results are also consistent with what was reported in Table 4.

6.4.5 Runtime comparison. To compare model runtimes, we used a Nvidia GeForce GTX 1080 Ti with a batch size of 256 and embedding size of 64. We do not report *MASR* and *AMASR*'s runtimes because their components are pretrained and fixed (i.e., there is no learning process/time). Figure 7 shows the runtimes (seconds per epoch) of our models and the baselines for each dataset. *MDR* only took 39 and 173 seconds per epoch in *30Music* and *AOTM*, respectively, while *MASS* took 88 and 375 seconds. *MDR*, one of the fastest models, was also competitive with *CML* and *MF-BPR*.

7 CONCLUSION

In this work, we proposed three novel recommendation approaches based on Mahalanobis distance. Our *MDR* model used Mahalanobis distance to account for both users' preferences and playlists' themes over songs. Our *MASS* model measured attentive similarities between a candidate song and member songs in a target playlist through our proposed memory metric-based attention mechanism. Our *MASR* model combined the capabilities of *MDR* and *MASR*. We also adopted and customized *Adversarial Personalized Ranking* (APR) loss with proposed flexible noise magnitude to further enhance the robustness of our three models. Through extensive experiments against eight baselines in two real-world large-scale *APC* datasets, we showed that our *MASR* improved 20.3%, and *AMASR* using *APR* loss improved 27.3% on average over the best baseline. Our runtime experiments also showed that our models were not only competitive, but fast as well.

ACKNOWLEDGMENT

This work was supported in part by NSF grant CNS-1755536, Google Faculty Research Award, Microsoft Azure Research Award, AWS Cloud Credits for Research, and Google Cloud. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. 2012. Build your own music recommender by modeling internet radio streams. In *WWW*. 1–10.

[2] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *ICCV*. 2425–2433.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[4] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *SIGKDD*. 714–722.

[5] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *SIGKDD*. 189–198.

[6] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential user-based recurrent neural network recommendations. In *RecSys*. 152–160.

[7] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative Memory Network for Recommendation Systems. *arXiv preprint arXiv:1804.10862* (2018).

[8] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation.. In *IJCAI*. 2069–2075.

[9] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. 2008. Playlist Generation using Start and End Songs.. In *ISMIR*, Vol. 8. 173–178.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *AISTATS*. 315–323.

[11] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys*. 161–169.

[12] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. 507–517.

[13] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *SIGIR*. 355–364.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[15] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*. 549–558.

[16] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM*. 843–852.

[17] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *WWW*. 193–201.

[18] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM*. 263–272.

[19] Dietmar Jannach, Iman Kamehkhosh, and Lukas Lerche. 2017. Leveraging multi-dimensional user models for personalized next-track music recommendation. In *SIGAPP*. 1635–1642.

[20] How Jing and Alexander J Smola. 2017. Neural survival recommender. In *WSDM*. 515–524.

[21] Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *SIGKDD*. 659–667.

[22] Iman Kamehkhosh and Dietmar Jannach. 2017. User perception of next-track music recommendations. In *UMAP*. 113–121.

[23] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *SIGKDD*. 447–456.

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[25] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*. 1378–1387.

[26] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[27] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. *arXiv preprint arXiv:1802.05814* (2018).

[28] Dawen Liang, Minshu Zhan, and Daniel PW Ellis. 2015. Content-Aware Collaborative Music Recommendation Using Pre-trained Neural Networks.. In *ISMIR*. 295–301.

[29] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[30] Chen Ma, Peng Kang, Bin Wu, Qinglong Wang, and Xue Liu. 2018. Gated Attentive-Autoencoder for Content-Aware Recommendation. *arXiv preprint arXiv:1812.02869* (2018).

[31] Chen Ma, Yingxue Zhang, Qinglong Wang, and Xue Liu. 2018. Point-of-Interest Recommendation: Exploiting Self-Attentive Autoencoders with Neighbor-Aware Influence. In *CIKM*. 697–706.

[32] B. McFee and G. R. G. Lanckriet. 2012. Hypergraph models of playlist dialects. In *ISMIR*.

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

[34] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126* (2016).

[35] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *ICDM*. 497–506.

[36] Sergio Oramas, Oriol Nieto, Mohamed Sordo, and Xavier Serra. 2017. A deep multimodal approach for cold-start music recommendation. *arXiv preprint arXiv:1706.09739* (2017).

[37] Martin Pichl, Eva Zangerle, and Günther Specht. 2017. Improving context-aware music recommender systems: beyond the pre-filtering approach. In *ICMR*. 201–208.

[38] Tim Pohle, Elias Pampalk, and Gerhard Widmer. 2005. Generating similarity-based playlists using traveling salesman algorithms. In *DAFx*. 220–225.

[39] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *SIGKDD*. 931–939.

[40] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.

[41] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. 811–820.

[42] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*. 285–295.

[43] Nitish Srivastava and Ruslan R Salakhutdinov. 2012. Multimodal learning with deep boltzmann machines. In *NIPS*. 2222–2230.

[44] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *NIPS*. 2440–2448.

[45] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[46] Irene Teinemaa, Niek Tax, and Carlos Bentes. 2018. Automatic Playlist Continuation through a Composition of Collaborative Filters. *arXiv preprint arXiv:1808.04288* (2018).

[47] Thanh Tran, Kyumin Lee, Yiming Liao, and Dongwon Lee. 2018. Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. In *CIKM*. 687–696.

[48] Thanh Tran, Xinyue Liu, Kyumin Lee, and Xiangnan Kong. 2019. Signed Distance-based Deep Memory Recommender. In *WWW*. 1841–1852.

[49] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music Listening and Playlists Dataset.

[50] Andreu Vall, Matthias Dorfer, Markus Schedl, and Gerhard Widmer. 2018. A Hybrid Approach to Music Playlist Continuation Based on Playlist-Song Membership. *arXiv preprint arXiv:1805.09557* (2018).

[51] Andreu Vall, Hamid Eghbal-Zadeh, Matthias Dorfer, Markus Schedl, and Gerhard Widmer. 2017. Music playlist continuation by learning from hand-curated examples and song features: Alleviating the cold-start problem for rare and out-of-set songs. In *DLRS*. 46–54.

[52] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *NIPS*. 2643–2651.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*. 5998–6008.

[54] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR*. 403–412.

[55] Xinxi Wang and Ye Wang. 2014. Improving content-based and hybrid music recommendation using deep learning. In *SIGMM*. 627–636.

[56] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. 2014. Exploration in interactive personalized music recommendation: a reinforcement learning approach. *TOMM* 11, 1 (2014), 7.

[57] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *NIPS*. 1473–1480.

[58] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*. 153–162.

[59] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. 2003. Distance metric learning with application to clustering with side-information. In *NIPS*. 521–528.

[60] Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *ICML*. 2397–2406.

[61] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *NAACL HLT*. 1480–1489.

[62] Ziwei Zhu, Jianling Wang, and James Caverlee. 2019. Improving Top-K Recommendation via Joint Collaborative Autoencoders. In *WWW*.