MMCF: Multimodal Collaborative Filtering for Automatic Playlist Continuation

Hojin Yang
Sungkyunkwan University, Republic of Korea
hojinyang7@skku.edu

Yoonki Jeong
Sungkyunkwan University, Republic of Korea
yoongi0428@skku.edu

Minjin Choi
Sungkyunkwan University, Republic of Korea
zxcvxd@skku.edu

Jongwuk Lee
Sungkyunkwan University, Republic of Korea
jongwuklee@skku.edu

ABSTRACT

Automatic playlist continuation (APC) is a common task of music recommender systems, enabling the automatic discovery of tracks that fit into a given playlist. To recommend a coherent list of tracks to users, it is important to capture the underlying characteristics of a playlist. Unfortunately, existing recommender models suffer from several problems: (1) They tend to misinterpret tracks that appear rarely in a playlist (*popularity bias*) (2) they cannot extend user's playlist that consists of very few tracks (*cold-start problem*), and (3) they neglect the context of a playlist such as the sequence of tracks or playlist title (*context-aware continuation*). This year's ACM RecSys Challenge'18 aimed to find new solutions to tackle these problems.

In this paper, we propose a *multimodal collaborative filtering* model to deal effectively with diverse data. This consists of two components: (1) an autoencoder using both the playlist and its categorical contents and (2) a character-level convolutional neural network using the playlist title only. By simultaneously analyzing the playlist and the categorical contents, our model successfully addresses the cold-start and popularity bias problems. In addition, we consider the context of a playlist by utilizing its title, thus enhancing the prediction of well-suited tracks. In the challenge, our team "hello world!" was ranked the 2nd place, scoring 0.224, 0.394, and 1.928 for the three evaluation metrics, respectively. Our implementation code is publicly available at https://github.com/hojinYang/spotify_recSys_challenge_2018.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**;

KEYWORDS

Music recommendation; multimodal data; hide-and-seek

RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-6586-4/18/10…$15.00
https://doi.org/10.1145/3267471.3267482

ACM Reference Format:
Hojin Yang, Yoonki Jeong, Minjin Choi, and Jongwuk Lee. 2018. MMCF: Multimodal Collaborative Filtering for Automatic Playlist Continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3267471.3267482

1 INTRODUCTION

With the growing popularity of online music streaming services such as Spotify, Pandora, Apple Music, and last.fm, *automatic playlist continuation* (*APC*) [2, 12] has received increasing attention. Given a playlist including a list of tracks, the goal of APC is to add one or more tracks that share the intrinsic characteristics of the target playlist sequentially. Therefore, it is important to reveal the underlying semantics of a given playlist by leveraging both a catalog of playlists and other playlist metadata.

Among the many possible solutions, *collaborative filtering* (*CF*) [11] is a common approach for developing recommender models for APC. The CF method analyzes a playlist-track dataset and predicts the unknown interests of users by inducing the latent patterns reflecting user preferences. While CF is effective for extracting the latent patterns of playlists for APC, its performance relies on the availability of a sufficient number of playlists.

Specifically, existing CF models [7, 8, 12] face the following challenges. (1) They tend to misinterpret tracks that appear very few times in a playlist (*popularity bias*). (2) Their accuracy is significantly degraded for a playlist with very few tracks (*cold-start problem*). In practice, an extreme case could be a playlist with no tracks, just a playlist title. (3) They do not fully utilize the context of a playlist such as the sequential order of tracks or the playlist title (*context-aware continuation*), even though the context is an evident clue for APC.

To address these issues, several hybrid approaches [5, 10, 15, 17] have been considered to fuse CF with content-based approaches. In early work, [5, 17] fused CF with content features such as acoustic features. Recently, Pichl et al. [10] attempted to understand the hidden meanings of playlist titles using contextual clustering. Vall et al. [15] proposed a hybrid model that combines playlists with multimodal features of tracks to infer playlist-track membership. Although the authors exploited playlists and their contents, they focused on the contents of either playlists or tracks, not both.

In this study, we propose a new *multimodal collaborative filtering* model , which considers the contents of both playlists and tracks. Our model consists of two components. First, we develop

an autoencoder that uses both playlists and categorical contents as input. Inspired by a *hide-and-seek* approach [13], we ignore either playlists or their contents during training. Furthermore, we devise two dropout strategies to adapt an order of playlists. In this manner, it is possible to identify not only complex patterns across playlists and contents but also to reveal the hidden context of an order of playlists. Next, we develop a character-level convolutional neural network (charCNN) [6, 18]. This model learns the latent relationships between playlists and their titles. For inference, we account for an ensemble method of two models by dynamically adjusting the importance of each model.

2 PRELIMINARIES

2.1 Challenge Framework

Problem statement. This year's ACM RecSys Challenge 2018 [3] was *automatic playlist continuation* (*APC*), which adds one or more tracks that fit into the underlying characteristics of a given playlist. For this challenge, the organizer (Spotify) released a dataset including three pillars: a collection of playlists, the title of each playlist, and some metadata of tracks. A playlist is represented by a list of tracks in the form of user-track interactions. A playlist title is in the form of a short text field, implying an abstract description of the playlist. The playlist title can provide sufficient evidence even if no tracks exist in the playlist. In addition, each track is associated with some categorical contents such as artists and albums. In Section 4.1, we will explain the statistics of the dataset in detail.

Evaluation task. Given a playlist of arbitrary length and its associated contents, the goal of the challenge is to predict the top-500 favorite tracks in a way that fits with the playlist. Specifically, the evaluation task consists of several subtasks according to the number of seed tracks (0, 1, 5, 10, 25, and 100), the availability of a playlist title, and the order of a playlist (*sequential* or *shuffled*). The evaluation tasks aim to manage the following issues faced by existing models:

- **Utilizing diverse contents**: To handle tracks that occur infrequently, some useful contents are provided, such as playlist titles and metadata for tracks. It is crucial to utilize both playlists and their contents.
- **Addressing the cold-start problem**: When no tracks are available, only a playlist title is provided. Therefore, it is necessary to develop a model that takes a short text title as input.
- **Understanding the order of tracks**: Users listen to tracks in either *sequential* or *shuffled* order, implying the context of the playlist. It is necessary to reveal the hidden context in the order of the playlist.

Evaluation metrics. In the challenge, three evaluation measures are used[1]. Let G be the ground truth set of tracks and R be the ordered list of recommended tracks. The size of a set or a list is denoted by $|\cdot|$ and from-to-subscript is used to index a list.

- **R-precision**: This is calculated as the number of retrieved relevant tracks divided by the number of known relevant tracks, regardless of the order of R.

$$\text{R-precision} = \frac{\left|G \cap R_{1:|G|}\right|}{|G|} \tag{1}$$

- **Normalized discounted cumulative gain (NDCG)**: The DCG is calculated and divided by the ideal DCG.

$$NDCG = \frac{DCG}{IDCG}, \text{ where } DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)}, \tag{2}$$

- **Recommended songs clicks**: Suppose that 10 tracks are added to a given playlist in sequence. This is calculated as the number of refreshes until a relevant track is found.

$$\text{clicks} = \left\lfloor \frac{\arg\min_i \{R_i : R_i \in G|\} - 1}{10} \right\rfloor \tag{3}$$

2.2 Mathematical Notations

Given a set of m playlists and a set of n tracks (or songs), a collection of playlists is represented by an $m \times n$ matrix $\mathbf{P} \in [0,1]^{m \times n}$. A playlist $\mathbf{p} \in \mathbf{P}$ is a list of tracks, and it is also represented by a row vector $\mathbf{p}$ of $\mathbf{P}$. That is, an entry $p_i \in \mathbf{p}$ represents the occurrence of track i in the playlist. If $\mathbf{p}$ includes track i, then p_i is set as 1. Otherwise, p_i is set as 0. Note that this does not necessarily imply that $\mathbf{p}$ is not relevant to track i, because the user who created $\mathbf{p}$ may be unaware of i.

Similar to the matrix $\mathbf{P}$ for playlists, the contents of playlists can be represented by an $m \times k$ matrix $\mathbf{A} \in [0,1]^{m \times k}$. The content $\mathbf{a_p} \in \mathbf{A}$ corresponds to $\mathbf{p}$, and is also represented by a row vector of $\mathbf{A}$. In this paper, we only consider artists as the content although other content can be available. An entry $a_i \in \mathbf{a_p}$ indicates the relevance of artist i for playlist $\mathbf{p}$. If playlist $\mathbf{p}$ includes any track by artist i, then a_i is set as 1. Otherwise, a_i is set as 0, meaning that there are no tracks by artist i.

Lastly, a playlist title is associated with the playlist $\mathbf{p}$. Let $c \in \mathbb{R}^d$ be a character-level embedding vector. Some text $\mathbf{T} \in \mathbb{R}^{t \times d}$ of length t is represented by an embedding matrix by concatenating a sequence of t character vectors, *i.e.*, $\mathbf{T} = \langle c_1; \cdots; c_t \rangle$. Let $\mathbf{T_p}$ be the playlist title corresponding to playlist $\mathbf{p}$.

2.3 Baseline Models

Autoencoders. An autoencoder [1] provides an effective method for learning the non-linear compact representation of data, and has also been employed for generative models. Recently, the autoencoder has been utilized for CF [4, 16]. Architecturally, it consists of two phases: *encoding* and *decoding*. Input data is first projected into a hidden representation using a mapping function, and the output connecting to the hidden representation is enforced to reconstruct its input. Because the hidden representation is usually represented by a low-dimensional vector, the latent representation of data can be accomplished, analogous to the dimensionality reduction approach.

We briefly explain how the autoencoder is used for CF. For simplicity, we formulate the autoencoder with one fully-connected-layer, although the number of hidden layers can be two or more.

- **Encoder**: The encoding function f maps an input vector $\mathbf{x} \in [0,1]^n$ to $\mathbf{y} \in \mathbb{R}^d$.

$$\mathbf{y} = f(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{4}$$

[1] http://www.recsyschallenge.com/2018/

where $\mathbf{W} \in \mathbb{R}^{d \times n}$, $\mathbf{b} \in \mathbb{R}^d$, and σ is a non-linear function. In this paper, the sigmoid function is used.

- **Decoder**: The decoding function g maps the hidden vector $\mathbf{y}$ to the output $\hat{\mathbf{x}} \in \mathbb{R}^n$ to reconstruct the input vector.

$$\hat{\mathbf{x}} = g(\mathbf{y}) = \sigma(\mathbf{W}'\mathbf{y} + \mathbf{b}'), \tag{5}$$

where $\mathbf{W}' \in \mathbb{R}^{n \times d}$ and $\mathbf{b}' \in \mathbb{R}^n$. Hence, the overall process can be written as $g(f(x))$ with respect to $\mathbf{x}$.

In this paper, because playlist $\mathbf{p}$ is taken as input, the objective function of the autoencoder is defined as follows:

$$\underset{\Theta}{\operatorname{argmin}} \sum_{\mathbf{p} \in \mathbf{P}} \mathcal{L}(\mathbf{p}, \hat{\mathbf{p}}), \tag{6}$$

where $\mathcal{L}$ is the loss function and $\hat{\mathbf{p}} = g(f(\mathbf{p}))$. For training, we learn parameters $\Theta = \{\mathbf{W}, \mathbf{W}', \mathbf{b}, \mathbf{b}'\}$ to minimize the reconstruction error between input vector $\mathbf{p}$ and and its reconstructed vector $\hat{\mathbf{p}}$. In this process, we adopt a cross-entropy loss function.

$$\mathcal{L}(\mathbf{p}, \hat{\mathbf{p}}) = -\frac{1}{n} \sum_{i=1}^{n} p_i \log \hat{p}_i + (1 - p_i) \log (1 - \hat{p}_i) \tag{7}$$

Although this is effective for playlists, it is limited for incorporating other contents. In Section 3.1, we will present a content-aware autoencoder using both the playlist and its contents.

Character-level convolutional neural network. The convolutional neural network (CNN) was originally proposed for image and video recognition. Because the CNN can capture the *spatial locality* of texts, it has also been utilized for text classification. In this paper, we adopt a character-level convolutional neural network (charCNN) [6, 18] to represent a latent embedding vector of a text.

Architecturally, this consists two components: *convolution layers* for generating local features and *pooling layers* for subsampling features to yield a concise representation. Using a filter $\mathbf{F} \in \mathbb{R}^{s \times d}$ of window size s, this model performs convolution on a sequence of text $\mathbf{T}$. (By default, the stride is 1.)

$$f_i = h(\mathbf{F} * \mathbf{T}_{i:(i+s-1)} + b), \tag{8}$$

where $*$ is the convolution operator, $\mathbf{T}_{i:(i+s-1)}$ is the subsequence of $\mathbf{T}$ from i to $i+s-1$, b is a bias for $\mathbf{F}$, and h represents the activation function. In this paper, h is the rectified linear unit (ReLU).

By using the filter in sequence, a feature vector $\mathbf{f} \in \mathbb{R}^{t-s+1}$ is obtained from $\mathbf{T}$.

$$\mathbf{f} = [f_1; \ldots; f_{t-s+1}] \tag{9}$$

Next, the pooling layer is used to subsample a given feature vector. In the case of max-pooling, this returns the maximum value of the feature vector $\mathbf{f}$, *i.e.*, $f' = \max([f_1; \ldots; f_{t-s+1}])$. Although we solely describe the convolution and pooling operator for one filter, it is possible to apply multiple filters and feature maps for $\mathbf{T}$. Using the charCNN, we can finally obtain a latent embedding vector of $\mathbf{T}$. In Section 3.2, we will explain how to utilize the charCNN model for a playlist title.

3 MULTIMODAL COLLABORATIVE FILTERING (MMCF)

In this section, we propose a *multimodal collaborative filtering* model using both a playlist and its contents. Figure 1 depicts the conceptual procedure of the proposed model with two components.

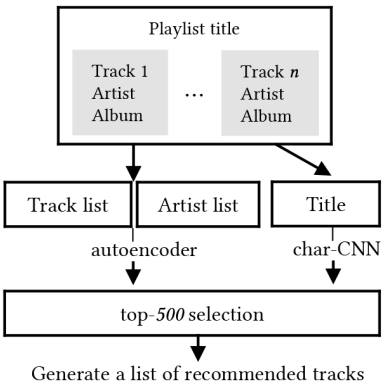

Figure 1: Overall procedure of our model handling multimodal data.

One is the autoencoder, which takes a playlist and artists as input. The other is the charCNN, which takes a playlist title as input. These are learned independently. After training, we finally discuss how to combine two models to recommend the top-N tracks for a given playlist.

3.1 Content-Aware Autoencoders

Although existing autoencoders can capture the characteristics of playlists, it is non-trivial to incorporate other categorical contents. Analogous to existing solutions such as CDAE [16] and Deep-Wide [4], one possible solution is to adapt the input layer by concatenating both a playlist and its contents. This solution has limitations for training, because it mainly focuses on reconstructing the playlist and it neglects to learn the latent pattern of playlists from contents. As a result, it is difficult to overcome popularity bias and the cold-start problem.

Model architecture. To address this problem, we propose a *content-aware autoencoder*, as depicted in Figure 2. A playlist $\mathbf{p}$ and its content $\mathbf{a_p}$ are concatenated, *i.e.*, $[\mathbf{p}; \mathbf{a_p}]$. Next, our model takes a *joint* playlist-content vector as input.

$$\underset{\Theta}{\operatorname{argmin}} \sum_{\mathbf{p} \in \mathbf{P}} \mathcal{L}([\mathbf{p}; \mathbf{a_p}], [\hat{\mathbf{p}}; \hat{\mathbf{a}}_\mathbf{p}]), \tag{10}$$

where $\Theta = \{\mathbf{W} \in \mathbb{R}^{d \times (n+k)}, \mathbf{W}' \in \mathbb{R}^{(n+k) \times d}, \mathbf{b} \in \mathbb{R}^d, \mathbf{b}' \in \mathbb{R}^{n+k}\}$.

Training. Inspired by a *hide-and-seek* idea, we propose a novel dropout method. First, we randomly select either a playlist or its artists. Next, we deactivate its feature vector by setting all values to 0. Either of the playlist or the artist are fully ignored during training, thereby enforcing that we learn both the marginals and joint information across playlists and contents. Although this training process is very simple, it is able to capture complex and diverse patterns across tracks and contents. Next, we perform a dropout scheme. For a non-zero vector, we randomly choose some values and set them to 0. This training process enables our model to be more robust.

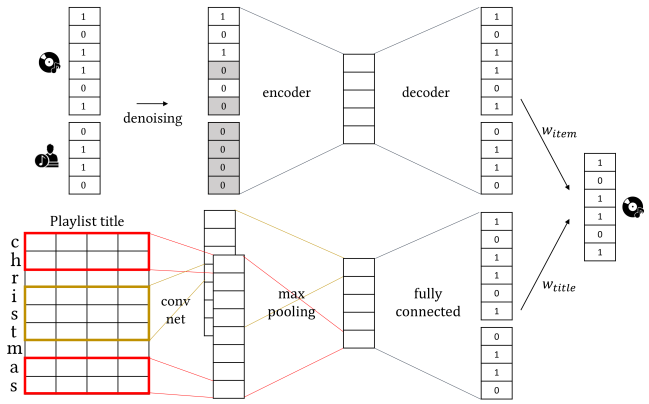

Figure 2: An architecture of the proposed model consisting of (1) the content-aware autoencoder using both the playlist and the artist and (2) the charCNN using the playlist title.

Specifically, we explain two possible training cases using the hide-and-seek idea.

- **Setting the artist as 0**: All values in $\mathbf{a_p}$ are set to 0, and $\mathbf{p}$ is only involved for learning, *i.e.*, $[\mathbf{p}; \mathbf{0}]$. Then, $\mathbf{p}$ is partially corrupted. Let $\tilde{\mathbf{p}}$ denote the corrupted vector of $\mathbf{p}$. In this manner, our model can learn *intra-relationships* between tracks (or contents) as well as the *inter-relationships* across tracks and artists. That is, the prediction for $\mathbf{a_p}$ is derived from $\mathbf{p}$.

$$\underset{\Theta}{\operatorname{argmin}} \sum_{\mathbf{p} \in \mathbf{P}} \mathcal{L}([\mathbf{p}; \mathbf{a_p}], g(f([\tilde{\mathbf{p}}; \mathbf{0}]))) \tag{11}$$

- **Setting the playlist as 0**: Conversely, all values in $\mathbf{p}$ are set to 0, and $\mathbf{a_p}$ is only involved for learning, *i.e.*, $[\mathbf{0}; \mathbf{a_p}]$, and $\mathbf{a_p}$ is partially corrupted. In this manner, we can learn other intra- and inter-relationships across tracks and artists.

$$\underset{\Theta}{\operatorname{argmin}} \sum_{\mathbf{p} \in \mathbf{P}} \mathcal{L}([\mathbf{p}; \mathbf{a_p}], g(f([\mathbf{0}; \tilde{\mathbf{a}}_\mathbf{p}]))) \tag{12}$$

Training using this the hide-and-seek idea has a two-fold effect. First, while a playlist-artist pair without dropout can ignore the artists to effectively reconstruct the playlist vector, the playlist-artist pair with 0 encourages our model to learn in an interleaved manner. Therefore, our model is learned by balancing two extreme cases. Next, the dropout method is also effective for regularization. As a result, additional regularization is rarely required, even if we treat sparse inputs.

Next, we need to capture the context by considering the order of a playlist. That is, users can listen to the tracks in either a *sequential* or a *shuffled* manner. It is observed that an optimal strategy for dropout should vary with the order of the playlist. Specifically, we account for a dropout strategy for each scenario.

- **Sequential list**: This suggests a *sequential dropout* strategy, which only chooses the forepart of the playlist for training, *i.e.*, the first K tracks, and predicts the remaining tracks of the playlist. The form of sequential dropout is similar for continuing tracks in the playlist.
- **Shuffled list**: As the typical dropout approach, this suggests a *random droput strategy*, by learning the model from randomly chosen tracks of the playlist.

Finally, we discuss how to distinguish the observed and missing values for input vectors. This problem is well-known as *one-class collaborative filtering* [9]. To address this problem, we adopt a uniform weight scheme. If a value is observed, then the weight is set to 1. Otherwise, the weight is set to α. (In our empirical study, α is set to 0.5). The loss function with the weighting scheme is formulated as follows.

$$\mathcal{L}(\mathbf{p}, \hat{\mathbf{p}}) = -\frac{1}{n} \sum_{\mathbf{p} \in \mathbf{P}} p_i \log \hat{p}_i + \alpha(1 - p_i) \log(1 - \hat{p}_i) \tag{13}$$

Inference. For inference, both $\mathbf{p}$ and $\mathbf{a_p}$ can be involved. However, because we are only interested in recommending tracks for a given playlist, the relative importance of $\mathbf{p}$ and $\mathbf{a_p}$ can vary. That is, the observed values of $\mathbf{a_p}$ are set to be smaller than those of $\mathbf{p}$, thereby enabling $\mathbf{p}$ to be regarded as more important. In our empirical study, $\mathbf{p}$ and $\mathbf{a_p}$ are set to have importance of 1.0 and 0.5, respectively.

3.2 CharCNN for Playlist Titles

We explain how to adapt the CharCNN model to infer the latent patterns between playlists and their titles. CharCNN is appropriate for incorporating playlist titles in two aspects. (1) Because playlist titles may contain *out-of-dictionary* words or typos, character-level embedding is more effective for playlist titles than word-level embedding. That is, character-level embedding can treat any words by decomposing the word into a sequence of characters. (2) For training, charCNN is faster than a recurrent neural network (RNN), while capturing the spatial locality of texts.

Model architecture. As depicted in Figure 2, this model takes a sequence of characters as input and returns the probabilities for tracks. Given a playlist title $\mathbf{T_p}$ of playlist $\mathbf{p}$, it predicts whether a track in $\mathbf{p}$ and its content $\mathbf{a_p}$ fit. That is, the prediction of charCNN is defined as follows:

$$[\hat{\mathbf{p}}; \hat{\mathbf{a}}_\mathbf{p}] = charCNN(\Phi, \mathbf{T_p}), \tag{14}$$

where $charCNN$ denotes the charCNN model and Φ is the set of learning parameters

Training. This model is trained by minimizing the average loss between the actual and predicted values.

$$\underset{\Phi}{\operatorname{argmin}} \sum_{\mathbf{p} \in \mathbf{P}} \mathcal{L}([\mathbf{p}; \mathbf{a_p}], [\hat{\mathbf{p}}; \hat{\mathbf{a}}_\mathbf{p}]), \tag{15}$$

where $[\hat{\mathbf{p}}; \hat{\mathbf{a}}_\mathbf{p}] = charCNN(\Phi, \mathbf{T_p})$ as in Eq (14). Given a playlist title, predicting the tracks in the playlist can be viewed as a classification problem. As in the proposed autoencoder, we also adopt the loss function as the cross entropy loss in Equation (13). Note that the loss function also distinguishes the difference between positive and negative feedback.

Inference. This is almost the same as in the training phase. Given a playlist title, we obtain the prediction $\hat{p}_i$ for each track, and choose the top-N tracks with the highest probabilities.

3.3 Combining Two Models

To account for multimodal data, we discuss how to combine two models for inference. In the case of predicting tracks for a playlist with a list of tracks and a title, we can boost the accuracy by combining the results of two models. In this paper, we introduce a simple linear combination of two output vectors. That is, w_{item} and w_{title} denote the weights of two models, respectively.

In the simplest way, the two weights can be set as constants. For example, in case with $w_{item} = 0.5$ and $w_{title} = 0.5$, the outputs of the two models contribute equally to the final inference. However, this ignores the fact that the accuracy of the autoencoder relies on the number of input items (tracks and contents). That is, the autoencoder can capture the characteristics of a given playlist more precisely as the number of input items increases. Based on this observation, we dynamically set the weights according to the number of items. Let $N([\mathbf{p}; \mathbf{a_p}])$ denote the number of items, and let $I(\mathbf{T_p})$ be the importance of the playlist title. In such a case, the weight for each model is set as follows:

$$w_{item} = \frac{N([\mathbf{p}; \mathbf{a_p}])}{N([\mathbf{p}; \mathbf{a_p}]) + I(\mathbf{T_p})}, w_{title} = \frac{I(\mathbf{T_p})}{N([\mathbf{p}; \mathbf{a_p}]) + I(\mathbf{T_p})} \quad (16)$$

It is observed that the influence of charCNN increases when the number of items is small. Conversely, the recommendation results are highly affected by the autoencoder as the number of items increases.

4 EXPERIMENTS

4.1 Experimental Setup

Dataset. The Million Playlist Dataset [14] comprises a set of 1M playlists including track lists, playlist titles, and other metadata for tracks. Each playlist contains 5–250 tracks. To process a playlist with few tracks, we utilize playlist titles to learn the context associated with a playlist, because this text-type title describes the intention of creating the playlist. The number of unique tracks is 2,262,292, where 1,073,419 tracks (47.45%) only appear once in the entire collection of playlists. The data sparsity is 99.9971%. Each track is associated with an artist and an album as contents, and in this paper, we only consider the artist. The number of unique albums and artists is 734,684 and 295,860, respectively.

Because the challenge does not provide a public test dataset, we randomly split a given dataset into training and validation sets. The training set consists of 997,000 playlists while the validation set consists of 3,000 playlists. As in the challenge evaluation setting, the number of seed tracks in the validation set varies as 0, 1, 5, 10, 25, and 100. In this paper, all the experimental results are reported in terms of the R-precision on the validation dataset. Due to space limitation, we only report R-precision results, showing a similar tendency for other metrics such as NDCG and song clicks.

Preprocessing. Because analyzing rare samples does not provide a statistically meaningful pattern, we decided to remove samples for tracks and artists that were too rare. To this end, we only consider tracks that occur more than 5, and artists who appears more than 3. After pruning, the number of tracks is 598,293 (26%) and the number of artists is 155,942 (53%). The data sparsity is reduced to 99.9893%. Although some information on playlists and their contents is lost, the accuracy of our model is not significantly affected. For playlist titles, we manage 41 characters including 26 letters (a-z), 10 numbers (0-9), and 5 special characters (/<>+-). The maximum length of a title is set to 25, and titles with more than 25 characters are truncated.

Table 1: Comparison between the existing autoencoder and the proposed autoencoder using artists. (R-precision)

	Sequential					*Shuffled*	
Input	1	5	10	25	100	25	100
Track	0.111	0.153	0.183	0.215	0.159	0.306	0.301
Track+Artist	**0.121**	**0.156**	**0.184**	**0.216**	**0.172**	**0.317**	**0.303**
Gain (%)	+9.00	+1.96	+0.55	+0.47	+8.18	+3.60	+0.66

Table 2: Effect of an ensemble method over varying the importance of the playlist title. (R-precision)

	Sequential						*Shuffled*	
Title leverage	0	1	5	10	25	100	25	100
Item only	-	0.121	0.157	0.184	0.216	0.172	0.317	0.303
Title only	0.078	-	-	-	-	-	-	-
Constant(0.5)	0.078	**0.131**	**0.158**	0.181	0.211	0.161	0.31	0.292
$I(\mathbf{T_p})$=20	0.078	0.102	0.142	0.175	0.213	0.171	0.313	0.301
$I(\mathbf{T_p})$=10	0.078	0.115	0.153	0.181	0.215	0.171	0.316	0.302
$I(\mathbf{T_p})$=5	0.078	0.128	**0.158**	**0.184**	**0.216**	**0.172**	**0.317**	0.302
$I(\mathbf{T_p})$=1	0.078	**0.131**	**0.158**	**0.184**	**0.216**	**0.172**	**0.317**	**0.303**
Gain (%)	-	+8.26	+0.64	-	-	-	-	-

Table 3: Comparison of different dropout strategies. The best result for each task corresponds to the final challenge submission. (R-precision)

	Sequential					*Shuffled*	
Input scheme	1	5	10	25	100	25	100
First[0.7,1.0)	**0.131**	0.147	0.168	0.195	0.149	0.217	0.184
First[1,50]	0.130	**0.158**	0.183	0.214	0.169	0.264	0.230
First[0.4,0.7]	0.114	0.152	**0.184**	**0.221**	**0.180**	0.291	0.254
Random[0.2,0.5]	0.102	0.118	0.151	0.191	0.162	**0.317**	**0.303**
Gain (%)	+28.43	+33.90	+21.85	+15.71	+11.11	+8.94	+19.29

Hyper-parameter setting. We adopt the Adam optimizer for both the autoencoder and the charCNN model with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and learning rate $\eta = 0.005$. For the autoencoder, the number of hidden dimensions is 256 and the mini-batch size is 250. For the charCNN model, the sizes of the filters are 3, 5, 7, and 9 with 150 feature maps. The mini-batch size is 150.

4.2 Experimental Results

In our empirical study, we evaluate the proposed model in three aspects: the effectiveness of the proposed autoencoder as opposed to existing ones, the effects of the ensemble methods on prediction accuracy, and the effects of various dropout strategies on the prediction accuracy.

Advantage of the proposed autoencoder using artists. We compared the proposed autoencoder against the classical autoencoder. To focus on the effect of artist information, the same dropout strategies were applied for both models. For the tasks with sequential

tracks, the dropout ratio was randomly chosen within [0.4,1.0) for each mini-batch. Subsequently, the forepart of the playlist remained, denoted as First[0.4,1.0). For shuffled tracks, the dropout ratio is chosen in [0.2, 0.5] for each mini-bath and tracks were randomly selected in the playlist, denoted as Random[0.2,0.5]. As reported in Table 1, it is observed that the proposed model using artists always outperforms the baseline using the playlists only, over all tasks. Specifically, when the number of input tracks is very small (*e.g.*, 1), our model exhibits an improvement of approximately 9% over the baseline model. As expected, additional information is useful for predicting playlists with sparse tracks. In addition, our model is effective for both sequential and shuffled orders. Regardless of the order of tracks in the playlist, artists can be helpful for addressing the popularity bias and cold-start problems.

Combination of two models. We evaluate how two models can be effectively combined with different weights. We vary the importance $I(\mathbf{T_p})$ of the title. As observed in Table 2, the best accuracy is obtained when $I(\mathbf{T_p}) = 1$. The accuracy decreases as the importance of the title information increases. Such a tendency indicates that the role of the charCNN model should be limited, ensuring it will not interfere with the prediction accuracy achieved by the autoencoder. Furthermore, when the number of input tracks is very small (*e.g.*, 1), combining two models yields an improvement of approximately 9% over the autoencoder only. Again, this means that the playlist title is particularly useful for improving the prediction accuracy for playlists with very few tracks.

Effects of dropout strategies. We evaluate how dropout strategies affect the accuracy by adjusting the dropout ratios. Table 3 reports the two dropout strategies with different ratios over all tasks. One interesting observation is that an effective dropout strategy depends on the order of input tracks. If the tracks are in order, then the sequential strategy exhibits a better accuracy than the random strategy. (Exceptionally, First[1,50] denotes a dropout strategy where the forepart of the playlist is left, and the number of input items is randomly selected between 1 and 50.) In particular, when the number of tracks is 25, a 16% gain is observed. Meanwhile, for the shuffled order, the random strategy performs better with a 19% gain in the case of shuffle-100. This observation implies that the order of continuing tracks significantly influences the characteristics of the playlist. Next, we observe that, models with a lower dropout ratio tend to perform better as the number of input tracks decreases. When the number of tracks is 1, the dropout method with a range of [0.7, 1.0) performs the best. In contrast, the dropout method with [0.4,0.7] shows the best accuracy over the other tasks. This indicates that the dropout ratio should be carefully chosen depending on the number of input tracks. Therefore, it is necessary to develop an adaptive dropout method that considers the number of tracks in the playlist.

5 CONCLUSION

In this paper, we presented a multimodal collaborative filtering model to account for both playlists and their diverse contents. Specifically, the proposed model consists of two collaborative filtering models: (1) an autoencoder to interpret the characteristics of a playlist and its artists and (2) a character-level convolutional neural network to capture the hidden relationships between playlists and their titles. After learning the two models independently, we devised an ensemble method that combines the two models for inference. Experimental results have demonstrated that our model is effective for alleviating the popularity bias and cold-start problems using multimodal data.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2018R1A2B6009135). This research was supported by the MIST (Ministry of Science and ICT), under the National Program for Excellence in SW (2015-0-00914), supervised by the IITP (Institute for Information & communications Technology Promotion).

REFERENCES

[1] Yoshua Bengio. 2009. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* (2009), 1–127.
[2] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *Comput. Surveys* (2014), 26:1–26:35.
[3] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. RecSys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA.
[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *ACM Conference on Recommender Systems (RecSys)*. 191–198.
[5] Justin Donaldson. 2007. A hybrid social-acoustic recommendation system for popular music. In *ACM Conference on Recommender Systems (RecSys)*. 187–190.
[6] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1746–1751.
[7] Seok Kee Lee, Yoon Ho Cho, and Soung Hie Kim. 2010. Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. *Information Sciences* (2010), 2142–2155.
[8] Ning-Han Liu, Szu-Wei Lai, Chien-Yi Chen, and Shu-Ju Hsieh. 2009. Adaptive Music Recommendation Based on User Behavior in Time Slot. *International Journal of Computer Science and Network Security (IJCSNS)* (2009), 219–227.
[9] Rong Pan, Yunhong Zhou, Bin Cao, Nathan Nan Liu, Rajan M. Lukose, Martin Scholz, and Qiang Yang. 2008. One-Class Collaborative Filtering. In *IEEE International Conference on Data Mining (ICDM)*. 502–511.
[10] Martin Pichl, Eva Zangerle, and Gunther Specht. 2015. Towards a Context-Aware Music Recommendation Approach: What is Hidden in the Playlist Name?. In *IEEE International Conference on Data Mining Workshop (ICDMW)*. 1360–1365.
[11] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International World Wide Web Conference (WWW)*. 285–295.
[12] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval (IJMIR)* (2018), 95–116.
[13] Krishna Kumar Singh and Yong Jae Lee. 2017. Hide-and-Seek: Forcing a Network to be Meticulous for Weakly-Supervised Object and Action Localization. In *IEEE International Conference on Computer Vision (ICCV) 2017*. 3544–3553.
[14] Spotify. 2018. Million Playlist Dataset. https://recsys-challenge.spotify.com/
[15] Andreu Vall, Matthias Dorfer, Markus Schedl, and Gerhard Widmer. 2018. A hybrid approach to music playlist continuation based on playlist-song membership. In *ACM Symposium on Applied Computing (SAC)*. 1374–1382.
[16] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *ACM International Conference on Web Search and Data Mining (WSDM)*. 153–162.
[17] Kazuyoshi Yoshii, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno. 2006. Hybrid Collaborative and Content-based Music Recommendation Using Probabilistic Model with Latent User Preferences. In *International Conference on Music Information Retrieval (ISMIR)*. 296–301.
[18] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems (NIPS)*. 649–657.