

Consistency-Aware Recommendation for User-Generated Item List Continuation

Yun He
Texas A&M University
yunhe@tamu.edu

Weiwen Liu
The Chinese University of Hong Kong
wwliu@cse.cuhk.edu.hk

Yin Zhang
Texas A&M University
zhan13679@tamu.edu

James Caverlee
Texas A&M University
caverlee@tamu.edu

ABSTRACT

User-generated item lists are popular on many platforms. Examples include video-based playlists on YouTube, image-based lists (or “boards”) on Pinterest, book-based lists on Goodreads, and answer-based lists on question-answer forums like Zhihu. As users create these lists, a common challenge is in identifying what items to curate next. Some lists are organized around particular genres or topics, while others are seemingly incoherent, reflecting individual preferences for what items belong together. Furthermore, this heterogeneity in item consistency may vary from platform to platform, and from sub-community to sub-community. Hence, this paper proposes a generalizable approach for user-generated item list continuation. Complementary to methods that exploit specific content patterns (e.g., as in song-based playlists that rely on audio features), the proposed approach models the consistency of item lists based on human curation patterns, and so can be deployed across a wide range of varying item types (e.g., videos, images, books). A key contribution is in intelligently combining two preference models via a novel consistency-aware gating network – a general user preference model that captures a user’s overall interests, and a current preference priority model that captures a user’s current (as of the most recent item) interests. In this way, the proposed consistency-aware recommender can dynamically adapt as user preferences evolve. Evaluation over four datasets (of songs, books, and answers) confirms these observations and demonstrates the effectiveness of the proposed model versus state-of-the-art alternatives. Further, all code and data are available at https://github.com/heyunh2015/ListContinuation_WSDM2020.

CCS CONCEPTS

• Information systems → Recommender systems;

KEYWORDS

Consistency-aware Recommendation, User-Generated Item Lists, Automatic Lists Continuation, Attention Mechanism

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WSDM '20, February 3–7, 2020, Houston, TX, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-6822-3/20/02...\$15.00

<https://doi.org/10.1145/3336191.3371776>

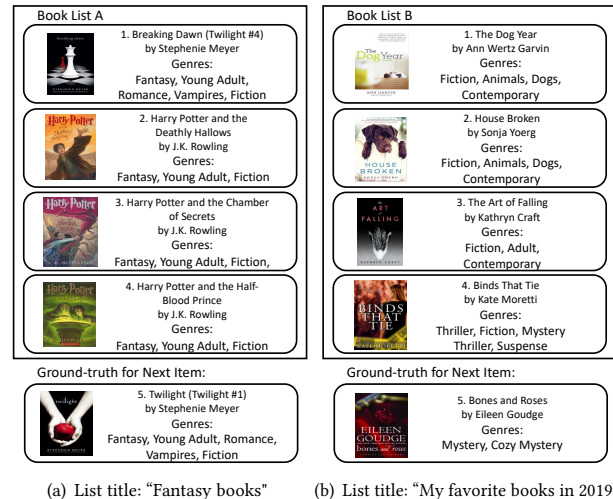


Figure 1: A motivating example of two lists with strong and weak consistency between recent items and previous items.

ACM Reference Format:

Yun He, Yin Zhang, Weiwen Liu, and James Caverlee. 2020. Consistency-Aware Recommendation for User-Generated Item List Continuation. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20)*, February 3–7, 2020, Houston, TX, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3336191.3371776>

1 INTRODUCTION

Human curation is a widely used feature in platforms like Spotify, Pinterest, YouTube, and Goodreads. Users can curate items like songs, images, videos and books to form lists that provide a unique perspective into how items can be grouped together. For example, Figure 1 shows two book lists on the book sharing platform Goodreads; one is organized around a genre (fantasy) while the other collects a personal list of favorites spanning genres. Since correlated items can be explored and consumed together, these item lists directly power user engagement. For example, more than 50% of Spotify users listen to playlists, accounting for more than 1 billion plays per week [38]; and Pinterest users have curated more than 3 billion pins to about 4 billion boards [10].

In these platforms, user-generated item lists are manually created, curated, and managed by users. Typically, users must first identify candidate items, determine if they are a good fit for a list, add them to a list, and then potentially provide ongoing updates

to the list (e.g., by adding or deleting items over time). To accelerate this process and assist users to explore more related items for their lists, we study the important yet challenging problem of *user-generated item list continuation*. That is, how can we recommend items that are related to the list and fit the user’s preferences? Compared with traditional item-based recommendation [34], list continuation faces complexities, since the length of lists varies from several items to even thousands of items and the user preference on items may dynamically evolve as a list develops. Moreover, the individual preference of how items are grouped together also varies for each user and across different platforms. Facing these challenges, we propose a consistency-aware recommender for user-generated item list continuation that is motivated by three key observations.

First, we observe that *some lists are strongly consistent while others are only weakly consistent*. Traditional song-based playlist continuation usually aims to generate strongly consistent lists, where each new song naturally fits with the previous ones [1, 4, 6, 12, 13, 21, 29–31, 33, 37]. A typical example is in [33], which generates a playlist with songs that are similar to the seed songs provided by the user. Indeed, some user-generated item lists may be organized around a specific theme, genre, or mood. However, some lists are organized by more personalized patterns and are seemingly inconsistent. To illustrate, List A in Figure 1(a) has strong consistency between recent books and previous books. The books are from either the *Harry Potter* series or the *Twilight* series, organized around genres like “Fantasy”. List B in Figure 1(b) is only weakly consistent with a mystery book *Binds That Tie* following two books about dogs. Indeed, we find in Figure 2 that list consistency varies greatly both within a single platform and across different platforms. As a result, we require new methods that can dynamically adapt to this heterogeneity of list consistency.

Second, we observe that *list consistency has a strong impact on the quality of predicting the next item*. The examples in Figure 1 show that the ground-truth for strongly consistent list A is the fantasy book in the *Twilight* series while the ground-truth for the weakly consistent list B is a mystery book that is similar with the most recent books in the list, but not the books earlier in the list. Our intuition is that if recent items are not similar with previous items, the user preference has probably changed recently. Therefore, the recent items (reflecting the current user preference) should be given more attention when we predict the next possible item. On the contrary, if recent items are still consistent with previous ones, the user preference is probably unchanged and so the general user preference should pay equal attention to items no matter if they have been curated early or late.

Third, we observe that *the definition of what makes a list consistent should arise from community norms rather than be imposed top-down*. Prior work in song-based playlists has found that neither audio signal-based similarity nor social tag-based similarity accurately reflect the consistency of manually constructed playlists [6, 30]. Our example so far in Figure 1 has highlighted how books in a single list may be drawn from different (seemingly inconsistent) genres. However, the consistency of items should depend on how users in the community perceive those items; so, if many users curate two items together, then those items should be considered consistent regardless of their superficial similarity. Hence, item list

continuation methods should seek to model consistency based on the curation patterns arising from the community itself.

These observations motivate us to attack the user-generated item list continuation problem with a **C**onsistency-aware and **A**ttention-based **R**ecommender (CAR). First, we propose both a general user preference model that aims to capture a user’s overall interests for the case of strongly consistent lists, and a current preference priority model that captures a user’s current interests for cases where consistency adapts over time. Second, to provide a generalizable approach that can handle the heterogeneity in list consistency across lists in different platforms, we design a novel consistency-aware gating network to adaptively balance and fuse the current and general user preferences, where the list consistency is modeled by computing the variation of the items in a list.

We evaluate CAR over four datasets reflecting different kinds of item lists and different curation patterns: the song-based Art of the Mix (AOTM) and Spotify, book-based Goodreads, and the answer-based Zhihu platform. Through experiments versus a suite of state-of-the-art baselines, we find that CAR significantly outperforms them and our general user preference model and current preference priority model can really complement each other guided by list consistency. Further, all code and data will be released to the research community for further exploration.

2 RELATED WORK

Automatic Playlist Generation and Continuation. Considerable prior research has focused on automatic song-based playlist generation and continuation, aligned along three branches. The first branch focuses on generative models for estimating the likelihood of a new playlist by treating training playlists as a set of song sequences. For example, McFee et al. [30, 31] apply a first-order Markov Chain for modeling playlists, which is improved by Chen et al. [6] via introducing metric embeddings. The second branch mainly relies on song-based audio features (e.g., Mel-frequency cepstral coefficients) to generate new playlists. Examples include [12] and [29], where Maillet et al. [29] train classifiers using audio-based features to determine if a sequence of songs can form a playlist. The third branch is most similar to our work, often applying information retrieval or recommendation methods to predict (or continue) the next N items. These approaches often are enhanced by content-based methods [13, 21, 33, 40–42]. A typical example by Jannach et al. [21] uses collaborative filtering and incorporates content features like social tags from Last.fm and popularity of songs.

Differences: There are two key differences between our work and traditional song-based playlist generation. First, we focus on not only song-based lists but propose a general framework that can be applied to other user-generated item lists, like book-based (Goodreads book lists), video-based (YouTube playlists), image-based (Pinterest boards) and answer-based (Zhihu collections) lists. Like song-based playlists, each collection of these item lists also provides a unique resource where correlated items can be easily explored and consumed together, directly empowering user engagement. Therefore, automatic generation and continuation for these item lists should be equally important as song-based playlists. Second, most traditional playlist continuation work assumes lists are always consistent [6, 12, 13, 21, 29–31, 33]. Except for [33]

introduced in the introduction, examples include [6, 30, 31] that apply language models to generate coherent playlists and [21] that recommends a set of songs whose tempo distribution is as similar as possible to the current playlist. However, we observe that the consistency varies greatly for lists both within and across different platforms, meaning that list continuation methods should dynamically adapt to these scenarios.

User-Generated Item Lists. Recently, user-generated item lists have received more and more interest. Zhong et al. [43] study the motivations of human curation to reveal the social values of user-generated item lists. Feinberg et al. [11] study the characteristics of item lists and claim there are two kinds of lists: for personal information management and for public expression. Lo et al. [27] analyze the growth of image collections on Pinterest. Lu et al. [25] and Eksombatchai et al. [10] distill user preference from Pinterest image-based lists to enhance individual image recommendation. Another interesting problem is to recommend existing item lists to users. He et al. [17] propose a hierarchical self-attentive model for recommending user-generated item lists (e.g., book lists and playlists) to right users. Besides, the List Recommending Model in [26] is proposed for recommending book lists and the Embedding Factorization Model in [3] is for recommending song playlists.

Neural Networks for Recommendation. Recently, neural networks have been widely applied in recommendation. He et al. [16] apply multilayer perceptron and generalized matrix factorization for implicit top-k recommendation. Ebesu et al. [9] apply memory networks for recommendation. Hidasi et al. [18] propose a GRU-based model for session-based recommendation and Tang et al. [36] propose a CNN-based model for sequential recommendation. Attention networks are often used for weighted-summing elements in a model. [5] propose an attentive collaborative filtering framework, where each item is segmented into component-level elements, and attention scores are learned for these components for obtaining a better representation of items. Attention networks are also applied in group recommendation [2] and sequential recommendation [19, 22].

3 PRELIMINARIES

In this section, we formulate the problem of user-generated item list continuation and present data-driven evidence to motivate our approach.

3.1 User-generated Item List Continuation

Let $U = \{u_1, u_2, \dots, u_{|U|}\}$ denote a set of users and a set of unique items is denoted by $V = \{v_1, v_2, \dots, v_{|V|}\}$. Each list generated by a user u , denoted by $S^u = [s_1, s_2, \dots, s_N]$, consists of a sequence of N items from set V . Moreover, $S_t^u = [s_1, s_2, \dots, s_t]$, $1 \leq t \leq N$ denotes a truncated item list at item-step t with regard to list S^u , where s_i is an item curated at time-step i . The goal of user-generated item list continuation is to predict the next possible curated item (i.e., s_{t+1}) based on the given S_t^u (for simplicity, we use S to denote S_t^u in the rest of the paper). Specifically, the recommender outputs a score for each item in the candidate set D ($D \in V$). After that, items in D can be ranked by their scores in descending order and top items can be returned for the item list continuation.

3.2 Evidence of Strong and Weak Consistency

The two examples in Figure 1 highlight a strongly consistent list and a weakly consistent list. Here, we present a brief data-driven examination to further explore such consistency across four different platforms: AOTM, Spotify, Goodreads and Zhihu (details of these datasets can be found in Section 5.1).

Measuring Consistency. To measure consistency of a user-generated item list S , we propose to measure the average of similarities between recent items and each of the previous items. So a strongly consistent list will have a high average similarity of new items that are added compared to the previous items. A weakly consistent list will have a low average similarity as new items begin to deviate from previous items. In contrast to related approaches like Intra-List Similarity proposed in [46] that measure the average of pairwise similarities between all items in a list, this consistency measure is designed to capture how a human curator may incrementally add items to the list. For simplicity, if we use the last item to represent recent items, then the consistency is:

$$\text{consistency-scores} = \sum_{s_i \in S, i \neq t} \frac{\text{sim}(s_i, s_t)}{N} \quad (1)$$

where N is the length of a list, s_t is the last item of list S and $\text{sim}(s_i, s_t)$ is the similarity between s_i and s_t . Of course there are many ways to measure this item-based similarity including collaborative filtering [35], co-occurrence of items [32] and content-based similarity. For this analysis we adopt a variation of word2vec [32] where items are “words” and lists are “sentences”. This approach captures the co-occurrence of items to generate item embeddings, which can then be used to calculate the cosine similarity between the embeddings: $\text{sim}(s_i, s_t) = \text{cosine}(e_i, e_t)$, where e_i, e_t are the embeddings of s_i, s_t derived from word2vec. Note that our proposed approach in Section 4 adopts a recommendation model-driven representation for items and lists instead of this word2vec-style approach.

Observations: For each of our four datasets – AOTM, Spotify, Goodreads and Zhihu – the histogram of *consistency-scores* is presented in Figure 2. We observe that *the list consistency varies greatly both within a single platform and across different platforms*. For example, Spotify lists cover a wide range of consistency values, with some lists having a *consistency-scores* more than 0.8 while others are lower than 0.4. Both AOTM and Goodreads have scores that tend to skew high, while the answer-based lists on Zhihu tend to be lower. Furthermore, this heterogeneity of list consistency could negatively impact the quality of predicting the next item. For example, a traditional playlist recommender that expects all items to maintain high consistency will suggest items that do not match the current user preference as suggested by the examples in Figure 1 and the data-driven evidence here. Therefore, we are motivated to propose a consistency-aware recommendation in the next section.

4 PROPOSED APPROACH: CAR

Motivated by this heterogeneity of list consistency, we propose a general user preference model to capture the user’s overall interests for strongly consistent lists, and a current preference priority model to emphasize more on the user’s current interests for lists with

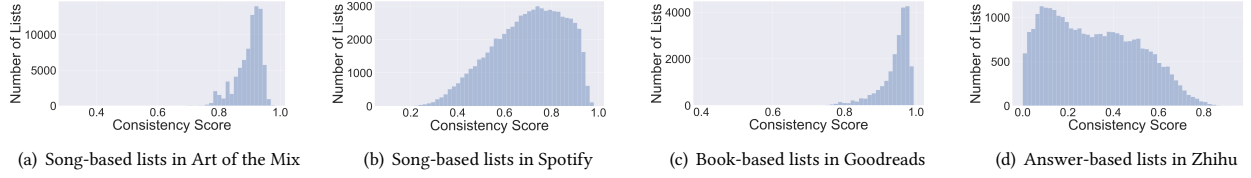


Figure 2: The consistency between recent items and previous items (by Eq.(1)). We observe that the list consistency varies greatly both within a single platform and across different platforms.

weak consistency. Moreover, as list consistency varies across lists in different platforms, we design a novel consistency-aware gating network to adaptively combine the two models. As discussed in Section 3.2, we propose to measure the consistency by computing the variation of the items in a list to aware if the current preference has deviated from the general user preference.

4.1 Attention-based User Preference Model

To predict the next possible item for each list, the key problem is to model the user preference from the items already curated in the list. A natural way is to first aggregate the item embeddings to represent the list embedding, and then match the obtained list embedding with other item embeddings to predict the next possible item.

To adaptively aggregate the item embeddings in the list, we design two aggregation schemes with attention networks – the general user preference model and the current preference priority model. Attention networks have been widely applied in recommendation to weighted-sum a variety of components (e.g., items [5, 15] and user behaviors [44, 45]), achieving promising performance. Following the terminologies in [39], attention mechanism can be formalized as:

$$\alpha_i = \text{softmax}(k_i^T q_i),$$

$$\text{Attention}(\{v_i, k_i, q_i | i = 1, \dots, t\}) = \sum_{i=1}^t \alpha_i v_i \quad (2)$$

where we first compute the inner product ($k_i^T q_i$) of (k_i) and query (q_i), then the weight (α_i), which reflects the importance of values (v_i), is obtained by normalizing over all key-query pairs with a softmax function ($\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^t e^{z_j}}$). The result is a weighted-sum by aggregating values (v_i). More clearly:

- Values (v_i) are components await to be aggregated together.
- Keys (k_i) correspond to values that are normally linear or non-linear transformed from values.
- Queries (q_i) play a role of benchmarking to measure the importance of values by matching against keys.

Embedding Layer: Let $Y = \{y_1, y_2, \dots, y_{|U|}\}$ denote the embedding vectors of users in U , where $y_u \in \mathbb{R}^d$ is a learnable d-dimensional real-valued vector for user u in U . Likewise, let $X = \{x_1, x_2, \dots, x_{|V|}\}$ denote the embedding vectors of items in set V , where $x_i \in \mathbb{R}^d$ is the i-th item embedding in list S . Note that $x_t \in \mathbb{R}^d$ represents the embedding of the last (most recent) item curated in the list at time-step t (i.e., s_t).

Design of Values and Keys: For both preference models, we first let $v_i = x_i$ in Eq.(2), since the goal of a user preference model is to aggregate item embeddings to represent a list embedding. After that,

we let $k_i = W^K x_i$, where $W^K \in \mathbb{R}^{d \times d}$ is a linear projection to map x_i into the “keys” space for matching against queries. The critical difference between the general and current preference priority model is how to choose queries (q_i), elaborated as follows.

General User Preference Model (GUPM): In strongly consistent lists, the user preference is probably unchanged. Hence, we design GUPM to model the overall user preferences in the list by treating items equally important, no matter at which time step the items have been curated. A natural way is to incorporate another learnable vector $h \in \mathbb{R}^d$ as the query, normally referred to as global or context vector in the literature [2, 5]. We have:

$$\alpha_i = \text{softmax}((W^K x_i)^T h),$$

$$l_S^G = \sum_{i=1}^t \alpha_i x_i \quad (3)$$

where $\forall i \in \{1, \dots, t\}$, $q_i = h$ denotes the query and l_S^G is the list embedding generated by the GUPM.

Current Preference Priority Model (CPPM): In weakly consistent lists, recent items begin to deviate from previous items and should be given more attention for predicting the next item. CPPM captures user’s current interests by assigning higher priority on recent item over other items. A natural way is to let the last item as the benchmark to measure the importance of other items: the more similar to the last item the higher weight is assigned. Hence, we let the last item embedding be the query:

$$\alpha_i = \text{softmax}((W^K x_i)^T W^Q x_t),$$

$$l_S^C = \sum_{i=1}^t \alpha_i x_i \quad (4)$$

where x_t is the embedding of the last item, $\forall i \in \{1, \dots, t\}$, $q_i = W^Q x_t$ denotes the query, $W^Q \in \mathbb{R}^{d \times d}$ is a linear projection to map x_t into “queries” space for matching against keys (i.e., $W^K x_i$) and l_S^C is the list embedding generated by CPPM.

4.2 Consistency-aware Gating Network

As observed in Section 3.2, the list consistency varies greatly both within a single platform and across different platforms. Hence, the mixture of GUPM and CPPM may be better to handle this heterogeneity of list consistency. Since GUPM is designed for strongly consistent lists while CPPM is designed for weakly consistent lists, a natural way to intelligently fuse them is to consider the degree of the list consistency. Here, we propose a novel consistency-aware gating network for assigning different weights for the two models.

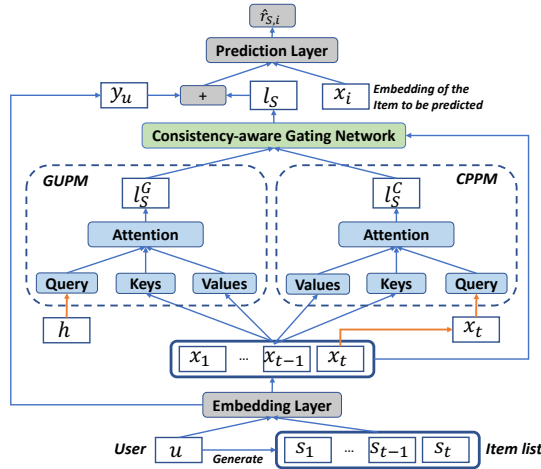


Figure 3: Framework of CAR for user-generated item list continuation.

The key question is how to feed the consistency into the gating network:

Consistency-based Input: We have applied the average of similarities between the last item and previous items for measuring the consistency in Eq.(1), which can also be viewed as the similarity between the last item and the centroid of a list in Eq.(5). The intuition here is that if the last item is different from the centroid of a list then the user preference has deviated from previous items.

$$\sum_{i=1}^t \frac{\text{sim}(x_i, x_t)}{t} = \text{sim}\left(\sum_{i=1}^t \frac{x_i}{t}, x_t\right) \quad (5)$$

where x_t is the embedding of the last item and $(\sum_{i=1}^t \frac{x_i}{t}) \in \mathbb{R}^d$ is the centroid embedding of a list. There are many ways to calculate the distance between two vectors like Manhattan distance or Euclidean distance. However, our goal here is not only to obtain the distance but also keeping the dimension of the vectors, which could be fed into the gating network for learning the weights. Hence, we use the difference between the two vectors to represent the consistency:

$$z^C = \sum_{i=1}^t \frac{x_i}{t} - x_t \quad (6)$$

where $z^C \in \mathbb{R}^d$ is the consistency-based input.

List-embedding-based Input: Items within a list also contain rich information to guide a recommender in fusing general and current preferences and should be fed into the gating network. The gating network receives a vector ($z \in \mathbb{R}^d$) as input so that it is inappropriate to pack item embeddings into a matrix as the input. Therefore, another attention network is firstly used to compress item embeddings into a list embedding:

$$z^L = \sum_{i=1}^t \text{softmax}((W_G^K x_i)^T h^G) x_i \quad (7)$$

where $z^L \in \mathbb{R}^d$ is the list embedding-based input, $W_G^K \in \mathbb{R}^{d \times d}$ is the projection to map x_i into “keys” space in this gating network and $h^G \in \mathbb{R}^d$ is the learnable vector to measure the weights of x_i . The final input is formed by concatenating z^C and z^L together:

$z = z^C \oplus z^L$. Then, following [20, 28], the gating networks are linear transformations of the input with a softmax layer:

$$g(z) = \text{softmax}(W^G z) \quad (8)$$

where $g(z) = [g(z)_C, g(z)_G] \in \mathbb{R}^2$ is the vector of gating values, $W^G \in \mathbb{R}^{2 \times d}$ is the transformation matrix.

4.3 CAR: Consistency-aware Attention-based Recommender

In Section 4.1, we propose two attention-based user preference models: the general preference model l_S^G and the current preference priority model l_S^C . In this section, we adaptively fuse these two by weighted-summing together via the consistency-aware gating network in Section 4.2:

$$l_S = g(z)_C \times l_S^C + g(z)_G \times l_S^G \quad (9)$$

where $l_S \in \mathbb{R}^d$ is the final list embedding encoded with both general and current user preferences and $g(z) = [g(z)_C, g(z)_G]$ are gating values.

User Embedding: So far, we only use item embeddings and list embeddings (aggregated from the item embeddings). Here, we add the user embedding of the list creator and list embedding together:

$$l_S^u = l_S + y_u \quad (10)$$

where y_u is the embedding of the user who generates list S . The intuition is that adding the user embedding may introduce more personalization information and provide better recommendation performance. Whether using this user embedding depends on the performance at the validation set and the impact of it will be discussed in Section 6.2.

Prediction Layer and Loss Function: For CUPM, CPPM and CAR, we apply the same prediction layer and loss function for fair comparison. List embedding l_S is firstly fed into a two-layer feed-forward network to introduce nonlinearity into the model, following [22, 39]. We have:

$$f_S^u = \text{ReLU}(l_S^u W^1 + b^1) W^2 + b^2 \quad (11)$$

where $W^1, W^2 \in \mathbb{R}^{d \times d}$ and $b^1, b^2 \in \mathbb{R}^d$ are parameters of the feed-forward network. After that, prediction score for item i is calculated by a matrix factorization layer: $r_{S,i} = x_i^T f_S^u$. Finally, the classic pairwise ranking loss used in BPR [34] is applied as our objective function:

$$- \sum_{S \in S^{all}} \sum_{t=1}^N \log(\sigma(r_{S,i} - r_{S,j})) \quad (12)$$

where S^{all} denotes the set of all lists, $r_{S,j} = x_j^T f_S^u$ and x_j is the embedding of randomly sampled negative item j and σ is the sigmoid function.

Model Training Adam [23] is applied as the optimizer. We perform mini-batch training, where each batch contains a group of lists. To generate more training samples, for each list, starting from the second item, each item is treated as the test item (s_{t+1}) and items before this item are treated as the training data. To control the length of lists in mini-batch, the most recent n items are considered. If the list length is less than n , a ‘padding’ item is repeatedly added until the length is n .

5 EXPERIMENTAL SETUP

In this section, we introduce the experimental setup, including datasets, metrics, baselines, and reproducibility details.

5.1 Datasets

We first select one relatively new dataset from the literature of automatic playlist generation and continuation:

- **Art of the Mix**¹ is a website where users can upload their song playlists. **AOTM**² dataset is proposed by [31] in 2011, collecting playlists from Jan 1998 to June 2011, resulting in totally 101,343 unique playlists. The main reason for choosing this dataset is that each list was generated by a user not a commercial radio DJ, which is an accurate sample of real playlists that occur in our daily life [30]. (Note that datasets used in the RecSys Challenge 2018 [4] have not been released.)

We also crawl and build three datasets, which are available at here³. These datasets are drawn from three popular platforms with different kinds of lists (book-based, song-based, and answer-based), reflecting a variety of scenarios to evaluate our approach.

- **Goodreads** is a popular site for book reviews and recommendation where users can curate correlated books into booklists.
- **Spotify** is a music streaming service where users can create playlists and curate songs into them.
- **Zhihu** is a question and answer community where users can select correlated answers to form an answer list. For example, a list named "Food" may contain answers as the response to two questions correspondingly: "How to cook steak?" and "Who is the barbecue king of Texas?".

Crawling all data generated in a short time window (e.g., one or two weeks) is widely used to sample a moderate-size dataset for research from millions or even billions of data [43]. However, since the generation of a list could take its owner months or years to occasionally curate hundreds of items, crawling all lists created in a short time window would obtain relatively short "immature" lists. We first randomly sample a batch of users and then crawl lists either created or followed by them.

For AOTM-2011, Goodreads and Zhihu datasets, following [22], we filter out items appearing fewer than 5 times and lists shorter than 5 items. For Spotify, the largest dataset, we filter out items appearing fewer than 20 times and lists shorter than 20 items. For extreme long lists, we truncate them by keeping the first 1,000 items. Table 1 summarizes these datasets, where any one of them has a million-size list-item interactions.

Table 1: Statistics of datasets

	AOTM	Goodreads	Spotify	Zhihu
#Users	14,115	12,525	14,911	9,561
#Lists	81,798	21,877	70,485	31,040
#Items	64,181	148,576	104,910	119,147
#List-item interactions	1,030,596	2,441,308	7,287,212	1,593,865
Avg. # of lists per user	5.8	1.7	4.7	3.2
Avg. # items per list	12.6	111.6	103.4	51.3
Density	0.020%	0.075%	0.099%	0.043%

¹<http://www.artofthemix.org/>

²<https://bmcfec.github.io/data/aotm2011.html>

³https://github.com/heyunh2015/ListContinuation_WSDM2020

Data Partition: We split each list into three parts: (1) the last item for testing, (2) the item right before the last one for validation, and (3) the rest of items for training. Note that during testing, the input sequences contain the training items and the validation item.

5.2 Evaluation Protocol

In real world like YouTube recommendation [8], it is common to only calculate prediction scores for items from a candidate set and rank them for the final recommendation, since it will be impractical to rank the whole millions of items in real world. The candidate generation model is not the emphasis of this work, hence we follow [16, 22] to randomly sample 100 negative items and rank these items with the ground-truth item.

Metrics: For each list, each tested algorithm selects K items ordered by prediction scores from the candidate items and then match the top K items against the ground-truth item. We adopt two evaluation metrics: Normalized Discounted Cumulative Gain (NDCG) at 5 (N@5) and 10 (N@10), and Hit Rate at 5 (HR@5) and 10 (HR@10).

5.3 Baseline

In this section, we introduce a suite of baselines. Except to traditional item-based recommendation methods include **ItemPop** (most popularity), **MF** [24] and **BPR** [34], we also have:

NeuMF. Neural collaborative filtering (NeuMF) [16] concatenates latent factors learned from a generalized matrix factorization model and a multi-layered perceptron model.

NMN. Neural Memory Network (NMN) [9] is a state-of-the-art memory-based neural network to unify the global latent factor model and local neighborhood structure.

TransRec. Translation-based Recommendation (TransRec) [14] models the third order interactions between a list, the item previously curated and the next item to curate.

GRU4Rec. GRU4Rec [18] is a GRU-based [7] RNN model for session-based recommendation (predict the last item for testing sessions). We treat each list as a session.

Caser. Convolutional Sequence Embeddings (Caser) [36] is a CNN-based model for sequential recommendation, which captures high-order markov chains by convolutional operations applied on the most recent items.

SASRec. Self-Attention based Sequential Recommendation model (SASRec) apply self-attention mechanism [39] for next item recommendation and achieves state-of-the-art performance, with large improvements over GRU4Rec and Caser observed in their paper.

Other Related Work but with Different Settings: Note that we are aware of the recent progress on automatic playlists continuation in [37]. However, their goal is to recommend a song to a list where order information is absent (i.e., a collection of songs). Since this goal is quite different from our continuing ordered items, we do not compare with it. Besides, many traditional playlists generation work (e.g., [6, 30, 31]) focus on estimating the likelihood of a new playlist, which is also different and not compared.

5.4 Reproducibility

All datasets and code of our approach can be found at here⁴. We implement CAR with Tensorflow. We implement ItemPop, BPR,

⁴https://github.com/heyunh2015/ListContinuation_WSDM2020

Table 2: Performance of CAR and the baselines

	AOTM				Goodreads				Spotify				Zhihu			
Metrics	HR@5	N@5	HR@10	N@10	HR@5	N@5	HR@10	N@10	HR@5	N@5	HR@10	N@10	HR@5	N@5	HR@10	N@10
ItemPop	0.2632	0.1817	0.3806	0.2194	0.2379	0.1666	0.3308	0.1964	0.2884	0.1983	0.4060	0.2362	0.4061	0.3003	0.5214	0.3375
MF	0.3821	0.2767	0.5002	0.3149	0.5900	0.4506	0.7105	0.4897	0.6432	0.4885	0.7686	0.5292	0.6135	0.5008	0.7059	0.5307
BPR	0.4711	0.3528	0.5965	0.3934	0.7229	0.5860	0.8207	0.6178	0.7367	0.5788	0.8495	0.6155	0.6209	0.5014	0.7255	0.5352
NeuMF	0.4519	0.3338	0.5733	0.3730	0.6779	0.5235	0.7906	0.5602	0.5950	0.4375	0.7362	0.4833	0.6363	0.5078	0.7434	0.5425
CMN	0.3885	0.2852	0.5036	0.3223	0.7262	0.5738	0.8277	0.6069	0.7230	0.5629	0.8385	0.6005	0.6633	0.5323	0.7653	0.5655
transRec	0.3704	0.2664	0.4944	0.3064	0.3287	0.2374	0.4398	0.2731	0.4012	0.2922	0.5224	0.3314	0.5033	0.3856	0.6200	0.4234
GRU4Rec	0.3013	0.2162	0.4091	0.2509	0.5131	0.4187	0.6140	0.4513	0.7584	0.6458	0.8403	0.6724	0.5603	0.4498	0.6647	0.4836
Caser	0.3678	0.2615	0.4980	0.3035	0.6118	0.4659	0.7442	0.5090	0.7094	0.5598	0.8193	0.5955	0.6461	0.5042	0.7709	0.5447
SASRec	0.4977	0.3650	0.6298	0.4077	0.7808	0.6449	0.8599	0.6707	0.8019	0.6577	0.8904	0.6865	0.7355	0.5899	0.8331	0.6217
CAR	0.5505*	0.4140*	0.6807*	0.4562*	0.7991*	0.6778*	0.8639	0.6990*	0.8113*	0.6758*	0.8945°	0.7029*	0.7468°	0.6096*	0.8410	0.6402*

[^] $p < 0.05$, [°] $p < 0.01$, * $p < 0.001$

and MF. The code of NeuMF⁵, MNM⁶, Caser⁷ and SASRec⁸ are from the authors. For TransRec⁹ and GRU4Rec¹⁰, we also use code from public resources. All neural network models were trained using Nvidia GeForce GTX Titan X GPU with 12 GB memory.

Parameter Settings. For CAR and baseline methods, all hyper-parameters are tuned on the validation dataset, where early stopping strategy is applied such that we terminate training if validation performance does not improve over 10 iterations. For CAR: the batch size is tested from {16, 32, 64, 128, 256} and 128 is selected for all datasets according to the results on the validation dataset. The learning rate is 0.001 for all datasets. Following [22], the candidates for the latent dimensionality d is from {10, 20, 30, 40, 50} and we select 50 for all datasets. The maximum sequence length n for all datasets is set to 500. For NeuMF, the number of MLP layers is set to 3. For CMN, the number of hops is set to 3. The Markov order of GRU4Rec and Caser is selected from {1, ..., 9}. For SASRec, the maximum length of handled sequence is set to 500 and the number of blocks is set to 2.

6 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we present our experimental results and discussion toward answering the following research questions (RQs):

- **RQ1:** How well does CAR perform for continuing user-generated item lists compared to baseline methods?
- **RQ2:** What is the impact of the design choices of CAR on the quality of list continuation? For example, is the consistency-aware gating network effective for fusing the two models?
- **RQ3:** What is the impact of list consistency? Specifically, for lists with the weak consistency, is the current preference priority model superior to the general preference model (and vice versa)?
- **RQ4:** What is the impact of the hyper-parameters of CAR on the quality of list continuation.

6.1 RQ1: List Continuation Performance

Experimental results of CAR and baseline methods are presented in Table 2. The results show that CAR outperforms all baseline. For example, CAR achieves an NDCG@5 of 0.4140 compared to

the second best performance 0.3650 from SASRec. The results of a two-sided significant test between CAR and the strongest baseline SASRec is also presented, where $\wedge$ means the p-value is smaller than 0.05, $\circ$ means the p-value is smaller than 0.01 and * means the p-value is smaller than 0.001. We see that the CAR approach significantly outperforms the strongest baselines in nearly all metrics. We observe that SASRec is the most competitive method with CAR, which is not surprising because SASRec is also an attention-based model that, to a certain extent, overcomes the long-term dependency problem suffered by CNN and RNN based models [39]. However, SASRec is proposed for sequential user action (e.g., purchase) prediction where current user preference (last purchased item) is often more important than general preference. Hence, for lists with strong consistency among the items (e.g., playlist “The lion king music”), SASRec may not fully utilize the user preference information from previous items. To sum up, the main improvement of CAR compared with SASRec is that we intelligently fuse current and general user preference together and further enhanced by the consistency between recent items and previous items.

6.2 RQ2: Ablation Study

Table 3: Ablation Study

	AOTM	Goodreads	Spotify	Zhihu
CAR	0.4140	0.6778	0.6758	0.6096
No Gating Network	0.3960	0.6734	0.6688	0.6035
CPPM	0.3935	0.6578	0.6448	0.5896
GUPM	0.3921	0.6524	0.6446	0.5812

Table 3 presents the results in terms of NDCG@5 of the ablation study at CAR. **Mixture of the Models:** We firstly compare GUPM (general preference model) and CPPM (current preference priority model) against “No Gating Network” model, which simply adds CPPM and GUPM together without using the consistency-aware gating network (i.e., $I_S = I_S^C + I_S^G$). We observe that the simple mixture of the two models outperforms any one of them. This shows that CPPM and GUPM do complement each other to provide a better overall performance.

Consistency-aware Gating Network: Then, we observe that CAR is superior to the version without the gating network. For example, CAR obtains 0.4140 in terms of NDCG@5 which is better than 0.3960 obtained by “No Gating Network”. This comparison shows that our proposed consistency-aware gating network is effective to intelligently fuse the current and general user preferences.

⁵https://github.com/hexiangnan/neural_collaborative_filtering

⁶<https://github.com/tebesu/CollaborativeMemoryNetwork>

⁷https://github.com/graytowne/caser_pytorch

⁸<https://github.com/kang205/SASRec>

⁹<https://github.com/YifanZhou95/Translation-based-Recommendation>

¹⁰https://github.com/Songweiping/GRU4Rec_TensorFlow

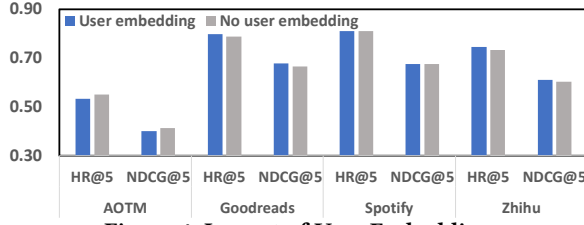


Figure 4: Impact of User Embedding

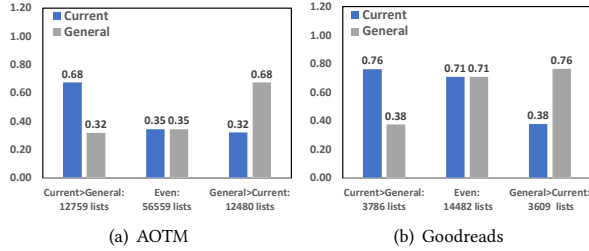


Figure 5: Performance in terms of NDCG@5 at cases that GUPM wins (General>Current), even and CPPM wins (Current>General). Each model has a large advantage over the other one on a bunch of lists. Same observation is obtained at Spotify and Zhihu.

User Embedding: We study the impact of user embedding introduced in Eq.(10). Users have different preferences for generating item lists. Hence, introducing user embeddings into the model of user preferences at each list is supposed to improve the personalization and provide better performance. However, in Figure 4, we observe that the user embedding harms the performance for Goodreads and Zhihu, achieves the same performance for Spotify and only improves the performance for AOTM. Table 1 shows that each user create 5.8 lists at AOTM, 4.7 lists at Spotify, 3.2 lists at Zhihu, and 1.7 lists at Goodreads. Hence, we observe that the more lists created by each user, the better performance can be obtained by adding the user embeddings. This is reasonable because the more lists a user creates the more personalization information of the user can be learned.

6.3 RQ3: Impact of List Consistency

In this section, we design an experiment to evaluate our intuition about the impact of list consistency on the quality of predicting the next item, presented in the introduction, that for lists with weak consistency between recent items and previous items, the user preference has changed and thus the recent items should be paid more attention. For these weakly consistent lists, we propose a current preference priority model (CPPM). And for lists with strong consistency, all of the curated items could be treated equally no matter curated early or late. Correspondingly, we propose general preference model (GUPM) for these strongly consistent lists. Hence, a natural way to evaluate the intuition is to compare the consistency between the lists where GUPM wins and the lists where CPPM wins.

Case by Case Comparison: We firstly compare the performance (taking NDCG@5 as an example) of GUPM and CPPM at each list to find out the lists where GUPM wins and the lists where CPPM

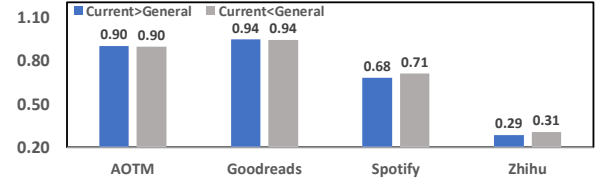


Figure 6: Comparison of average consistency between the cases that CPPM wins (Current>General) and the cases that GUPM wins (General>Current).

wins. In Figure 5, we observe that the two models both have a large advantage over the other one on a section of lists. Taking AOTM dataset as an example, GUPM is superior to CPPM with a large margin of 0.68 against 0.32 on a different set of 12,480 lists, while CPPM achieves 0.68, largely outperforming 0.32 obtained by GUPM on 12,759 lists. Such a huge divergence of the performance suggests that the two sets of lists are different from each other.

Consistency Comparison: Hence, we calculate the consistency between the last item and previous items according to Eq.(1) and compare the average consistency of the two bunches of lists in Figure 6. Note that we use the item embeddings from trained word2vec model in Section 3.2, rather than embeddings from our approach, in order to provide a more objective perspective for measuring the consistency. Interestingly, in AOTM and Goodreads with commonly strong consistency, we observe that the lists where GUPM wins have a similar consistency with the lists where CPPM wins. This is understandable that user general and current preference should generate similar results if a list is always consistent. However, in Spotify and Zhihu with commonly weak consistency, we observe that the lists where GUPM wins have a stronger consistency than the lists where CPPM wins. Taking Spotify as an example, the lists where GUPM wins have an average consistency of 0.71 while the lists where CPPM wins have a weaker consistency of 0.68. Hence, the comparison between the average consistency of GUPM and CPPM in these two datasets supports our intuition of the impact of list consistency on the quality of predicting the next item.

6.4 RQ4: Impact of Hyper-parameters

In this section, we analyze the impact of hyper-parameters. Due to the limited space, two important hyper-parameters are selected to discuss. In Figure 7(a), we observe that our approach benefits from larger numbers of the latent dimensionality d . The best results are obtained with $d = 50$ in Goodreads and Zhihu. In Figure 7(b), we observe that the longer the item sequence that is fed into our model, the better performance can be obtained for predicting the next item. The best results are achieved with $n = 500$ in Goodreads and Zhihu and the performance decreases largely when $n < 100$. This demonstrates that user's general preference always maintains influence on curating items in a list and items that are curated even a long time ago should also be paid attention for list continuation.

7 CONCLUSION AND FUTURE WORK

In this paper, we focus on automatically continuing user-generated item lists, i.e., predicting the next possible item for each list. We observe that the consistency between recent and previous items

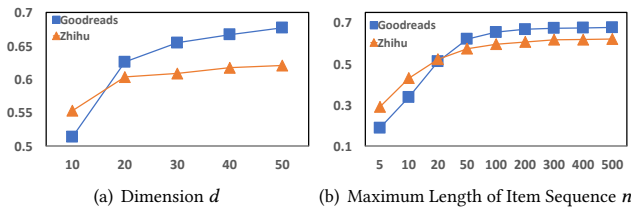


Figure 7: Impact of hyper-parameters in terms of NDCG@5 taking Goodreads and Zhihu as examples.

reflects the dynamic change of user preferences. Specifically, if recent items are not similar with previous items, the user preference has probably changed and hence we should pay more attention on the recent items for predicting the next item. Therefore, we are motivated to propose a consistency-aware attention-based recommender (CAR), where a novel consistency-aware gating network is designed for capture the discontinuity between the current user preference and the general user preference. The evaluation on four datasets demonstrate the effectiveness of CAR versus state-of-the-art alternatives. In the future, we will utilize some side information to enhance CAR, like friend relationships among users and text-based titles of user-generated item lists.

ACKNOWLEDGMENTS

This work is supported in part by NSF (#IIS-1841138).

REFERENCES

- [1] Geoffroy Bonnin and Dietmar Jannach. 2015. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)* (2015).
- [2] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. 2018. Attentive group recommendation. In *SIGIR*.
- [3] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. 2017. Embedding factorization models for jointly recommending items and user generated lists. In *SIGIR*.
- [4] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. Recsys challenge 2018: Automatic music playlist continuation. In *Recsys*.
- [5] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *SIGIR*.
- [6] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *KDD*.
- [7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
- [8] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Recsys*.
- [9] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *SIGIR*.
- [10] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *WWW*.
- [11] Melanie Feinberg, Gary Geisler, Eryn Whitworth, and Emily Clark. 2012. Understanding personal digital collections: an interdisciplinary exploration. In *Proceedings of the Designing Interactive Systems Conference*. ACM, 200–209.
- [12] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. 2008. Playlist Generation using Start and End Songs.. In *ISMIR*.
- [13] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latent topic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*. ACM, 131–138.
- [14] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *Recsys*.
- [15] Xiangnan He, Zhenkui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural Attentive Item Similarity Model for Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2018).
- [16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.
- [17] Yun He, Jianling Wang, Wei Niu, and James Caverlee. 2019. A Hierarchical Self-Attentive Model for Recommending User-Generated Item Lists. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, 1481–1490.
- [18] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
- [19] Xiaowen Huang, Shengsheng Qian, Quan Fang, Jitao Sang, and Changsheng Xu. 2018. CSAN: Contextual Self-Attention Network for User Sequential Recommendation. In *MM*.
- [20] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.
- [21] Dietmar Jannach, Lukas Lerche, and Iman Kamehkhosh. 2015. Beyond hitting the hits: Generating coherent music playlist continuations with the right tracks. In *Recsys*.
- [22] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*.
- [23] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. (2009).
- [25] Yuchen Liu, Dmitry Chechik, and Junghoo Cho. 2016. Power of Human Curation in Recommendation System.. In *WWW (Companion Volume)*.
- [26] Yidan Liu, Min Xie, and Laks VS Lakshmanan. 2014. Recommending user generated item lists. In *Recsys*.
- [27] Caroline Lo, Justin Cheng, and Jure Leskovec. 2017. Understanding online collection growth over time: A case study of Pinterest. In *WWW (Companion Volume)*.
- [28] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *KDD*.
- [29] François Maillet, Douglas Eck, Guillaume Desjardins, Paul Lamere, et al. 2009. Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists.. In *ISMIR*. 345–350.
- [30] Brian McFee and Gert RG Lanckriet. 2011. The Natural Language of Playlists.. In *ISMIR*, Vol. 11. 537–541.
- [31] Brian McFee and Gert RG Lanckriet. 2012. Hypergraph Models of Playlist Diagnostics.. In *ISMIR*, Vol. 12. Citeseer, 343–348.
- [32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.
- [33] John C Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. 2002. Learning a gaussian process prior for automatically generating music playlists. In *NIPS*.
- [34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
- [35] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. 2001. Item-based collaborative filtering recommendation algorithms. *WWW* (2001).
- [36] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*.
- [37] Thanh Tran, Renee Sweeney, and Kyumin Lee. 2019. Adversarial Mahalanobis Distance-based Attentive Song Recommender for Automatic Playlist Continuation. *arXiv preprint arXiv:1906.03450* (2019).
- [38] Reggie Ugwu. 2016. <https://www.buzzfeed.com/reggieugwu/the-unsung-heroes-of-the-music-streaming-boom>. In <https://www.buzzfeed.com/reggieugwu/>.
- [39] 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*.
- [40] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. 2018. Two-stage model for automatic playlist continuation at scale. In *Proceedings of the ACM Recommender Systems Challenge 2018*. ACM, 9.
- [41] Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. 2018. An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. *arXiv preprint arXiv:1810.01520* (2018).
- [42] Xing Zhao, Qingquan Song, James Caverlee, and Xia Hu. 2018. TrailMix: An Ensemble Recommender System for Playlist Curation and Continuation. In *Recsys*.
- [43] Changtao Zhong, Sunil Shah, Karthik Sundaravadivelan, and Nishanth Sastry. 2013. Sharing the loves: Understanding the how and why of online content curation. In *Seventh International AAAI Conference on Weblogs and Social Media*.
- [44] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiuxi Chen, and Jun Gao. 2017. ATRank: An Attention-Based User Behavior Modeling Framework for Recommendation. *arXiv preprint arXiv:1711.06632* (2017).
- [45] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. 2018. Micro behaviors: A new perspective in e-commerce recommender systems. In *WSDM*.
- [46] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*. ACM, 22–32.