Collective Embedding for Neural Context-Aware Recommender Systems

Felipe Soares da Costa
Aalborg University
Aalborg, Denmark
fcosta@cs.aau.dk

Peter Dolog
Aalborg University
Aalborg, Denmark
dolog@cs.aau.dk

ABSTRACT

Context-aware recommender systems consider contextual features as additional information to predict user's preferences. For example, the recommendations could be based on time, location, or the company of other people. Among the contextual information, time became an important feature because user preferences tend to change over time or be similar in the near future. Researchers have proposed different models to incorporate time into their recommender system, however, the current models are not able to capture specific temporal patterns. To address the limitation observed in previous works, we propose Collective embedding for Neural Context-Aware Recommender Systems (CoNCARS). The proposed solution jointly model the item, user and time embeddings to capture temporal patterns. Then, CoNCARS use the outer product to model the user-item-time correlations between dimensions of the embedding space. The hidden features feed our Convolutional Neural Networks (CNNs) to learn the non-linearities between the different features. Finally, we combine the output from our CNNs in the fusion layer and then predict the user's preference score. We conduct extensive experiments on real-world datasets, demonstrating CoNCARS improves the top-N item recommendation task and outperform the state-of-the-art recommendation methods.

CCS CONCEPTS

• **Information systems → Recommender systems**; **Collaborative filtering**; • **Computing methodologies → Neural networks**.

KEYWORDS

Context-aware Recommender Systems; Collective Embedding; Convolutional Neural Networks; Time-aware Recommendations

ACM Reference Format:
Felipe Soares da Costa and Peter Dolog. 2019. Collective Embedding for Neural Context-Aware Recommender Systems. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347028

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347028

1 INTRODUCTION

Recommender systems algorithms aim to assist the user in the decision-making process when facing an overwhelming amount of choices. Recommender systems play an important role in filtering out irrelevant information and selecting a personalized subset of items to maximize the users' satisfaction. Collaborative Filtering (CF) is often the recommendation technique applied to the modern systems, exploiting the previous user-items interactions and the interactions of other users for modeling the user's preference. Among the various CF methods, Matrix Factorization (MF) based methods are known to outperform other methods and have become the state-of-art solution of recommendation research.

The design of a CF model requires the understanding of how to represent a user and an item, and how to model the user-item interaction based on the representation. MF characterizes users and items as vectors of latent factors (also known as embeddings). Furthermore, MF represents a user-item interaction as the inner product of the user and item embeddings as the preference score. Several extensions have been developed for MF to improve both the modeling and learning aspect. For example, Adversarial Personalized Ranking (APR) [10] learns user and item embeddings through an adversarial training procedure, Neural Matrix Factorization (NeuMF) [12] extends MF by learning embeddings with a multi-layer perceptron network, and Bayesian Personalized Ranking (BPR) [23] optimizes MF for implicit feedback applying the pairwise ranking method.

Despite the accuracy of the aforementioned methods, the proposed techniques assume the dimensions of the embedding space are independent of each other. The hypothesis is impractical since the embedding dimensions could contain dependent features such as items' attributes. Furthermore, the hypothesis is sub-optimal for learning contextual information from real-world data as demonstrated by Context-aware Recommender Systems (CARS) [2, 5, 20]. CARS has outperformed other models by learning the contextual interaction function from the data.

Among the CARS for CF, time-aware recommender systems have gained some attention, since users preferences may remain the same in the near future [3] or change over time [21]. Tensor Factorization (TF) provides the state-of-art performance by projecting users and items embeddings into a latent space with time embeddings [2]. We argue that a potential limitation of this method is TF assumes two consecutive time slots independent of each other, which makes it infeasible to predict the recommendation of the next time slot. Moreover, TF does not capture the time-evolving patterns, for example, the user's impression of a movie may be dynamically affected according to the award received by the movie in the past hours. On the other hand, the user may keep similar preferences

over time. As an example, Figure 1 illustrates a scenario where a user watches a sequence of movies in different time slots, and after that, the recommender engine suggests a list of items according to the previous sequence.

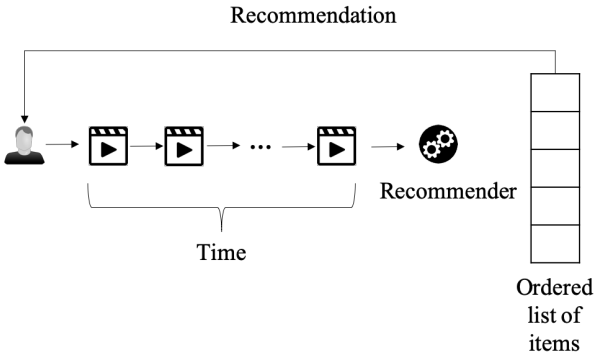

Figure 1: Example for user behaviour over time.

To address the aforementioned limitations and support the hypothesis given by Figure 1, we propose a new architecture for Collective embedding for Neural Context-aware Recommender Systems (CoNCARS). CoNCARS models a user by embedding all items interacted by the user in a specific time slot. Likewise, CoNCARS models the item by embedding all users interacting with the item in a specific time slot.

We incorporate the time as a feature to identify the importance of different interacted items. To do so, we compute the *time-based user embeddings* by aggregating the user and time embeddings. Similarly, we compute the *time-based item embeddings* by aggregating the item and time embeddings. The collective embeddings related to each user and item jointly project their latent representations more accurately as shown in the empirical evaluation.

Furthermore, we combine the *time-based item embeddings* and *time-based user embeddings* with the user and item embeddings to model the user-item correlations. Moreover, CoNCARS learns the non-linear correlations between different dimensions with the Convolutional Neural Network (CNN).

To summarize, the paper presents the following contributions:

- A context-aware model named CoNCARS to improve the prediction based on user's preferences over time;
- A training model for a better learning correlation between users, items and time embedding dimensions;
- Empirical evaluation in three real-world datasets demonstrating the effectiveness of the CoNCARS model.

2 RELATED WORKS

Context-aware recommender systems improved the recommendation task, especially by minimizing the cold-start problem and by retrieving items based on the contextual information. Time has shown to be strong contextual information because users tend to change or keep their behavior over time [3, 27]. For a better overview, we divide the time-based recommendations approaches into two categories and exploit the techniques closely related to ours:

- *Time-dependent recommender systems* model user's preference as a chronological sequence. It considers the order of the events to have a strong influence on the estimated predictions. This approach requires that the input data is in chronological order and does not take the exact timestamps into account. Therefore, the time-dependent approach aims to adapt the recommendations according to changes over time.
- *Time-aware recommender systems* consider the time as a contextual feature during the training step, aiming to model the time as being cyclic. Timestamps are used as additional information to enrich the model. Time-aware methods consider that a user's behavior repeats at regular time intervals. Based on this assumption, time-aware recommender systems tend to have more accurate predictions of similar patterns in the future.

In this paper, we propose a time-aware recommender system. Usually, the time-aware recommender system is modeled in three different ways: pre-filtering, post-filtering, and contextual modeling. Pre- and post-filtering have shown slightly lower performance when compared to contextual modeling. Therefore, we explore the contextual modeling technique.

Baltrunas *et al.* [1] propose the Context-aware Matrix Factorization (CAMF) by extending matrix factorization using context as a baseline predictor to represent the interaction of contextual information with users or items. Another variation of MF is called TF, considering time as a third-dimensional factor in the user-item interaction [2, 16]. Collective Matrix Factorization (CMF) is an extension of MF proposing to collectively factorize multiple representations of user-item relations, identifying a common latent space to both item features and user-item interactions vector [5, 11, 19, 25].

Advances in deep learning research field allowed to capture non-linear correlations between users and items, improving the recommendation task. He *et al.* [12] provide a neural collaborative filtering model, named NeuMF, which adds an adaptable multilayer perceptron (MLP) to learn the interaction function. Wu *et al.* [26] propose an autoencoder based CF method to improve the effectiveness for top-N recommendation task. Kim *et al.* [15] integrate Convolutional Neural Network (CNN) into a Probabilistic Matrix Factorization (PMF) model to address the sparsity problem in recommendation techniques. Nonetheless, the deep collaborative filtering approaches do not exploit the advantages of time as an important dimension for the recommendations task.

Our recommendation method differentiates from the methods described above by capturing the temporal information provided by user-item interactions. Furthermore, our model represents a user-based embedding for each item considering the time as a contextual feature, which identifies the importance of different users and items. Likewise, our model denotes an item-based embedding for each user considering the time as a contextual feature, identifying the relevance of different users and items.

3 PRELIMINARIES

Consider a scenario of a user **u** interacting with an item **i** in time **t** in online services, for example, Yelp, Pinterest, or Netflix. To model

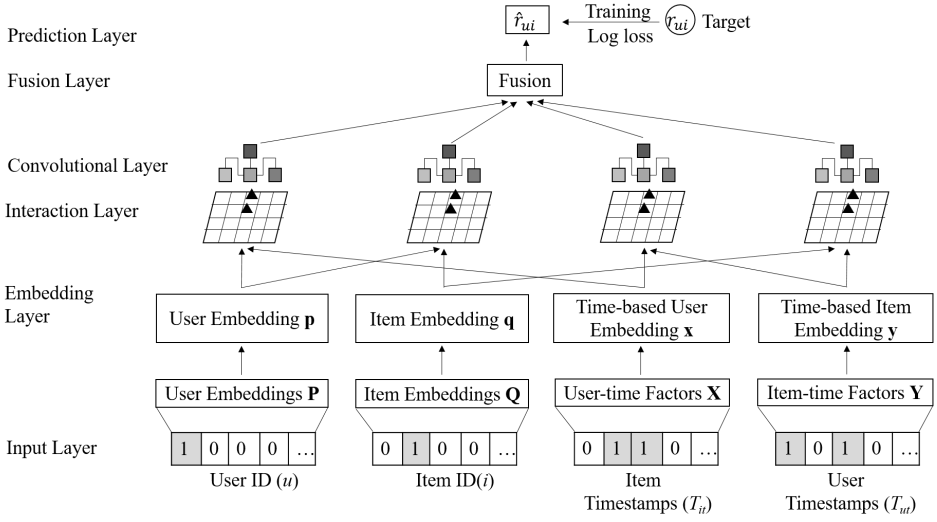

Figure 2: Collective Embedding for Neural Context-aware Recommender Systems.

the relationships among users and items over time, we apply joint embedding to represent their interactions.

The set of users is denoted as $U = \{\mathbf{u}_1, ..., \mathbf{u}_M\}$ and the set of items as $I = \{\mathbf{i}_1, ..., \mathbf{i}_N\}$. $R = \{r_{ui} \in \mathbb{R}^{M \times N} | 1 \leq r_{ui} \leq 5\}$ denotes the rating matrix, where r_{ui} is the rating of user $\mathbf{u}$ on item $\mathbf{i}$. M and N denotes the number of users and items, respectively. r_{ui} is labeled as *unk* if user $\mathbf{u}$ did not interact with item $\mathbf{i}$. The matrix R with implicit feedback is modeled as:

$$R_{ui} = \begin{cases} 0, & \text{if } R_{ui} = unk \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

Recommender systems solve the problem of predicting the preference score of each unobserved entry in R. The preference scores determine the order of items displayed in the recommended list of items. We generate the scores according to Equation 2.

$$\hat{r}_{uit} = F(u, i, t|\mathbf{\Theta}) \tag{2}$$

where $\hat{r}_{uit}$ is the predicted score of interaction r_{uit} between user $\mathbf{u}$ and item $\mathbf{i}$ in time $\mathbf{t}$, $\mathbf{\Theta}$ represents the model parameters, and F is the function which estimates the predicted scores based on $\mathbf{\Theta}$. The function F is used to find an optimal list of items for an individual user according to her/his preferences.

Traditionally, MF methods define function F based on element-wise product of $\mathbf{p}_u$ and $\mathbf{q}_i$ to predict $\hat{R}_{ui}$ as demonstrated by [18], where $\mathbf{p}_u$ and $\mathbf{q}_i$ defines the hidden latent factors of $\mathbf{u}$ and $\mathbf{i}$, respectively.

$$\hat{R}_{ui} = F(u, i|\Theta) = \mathbf{p}_u^T \mathbf{q}_i \tag{3}$$

However, CoNCARS uses outer product to calculate the interactions between users and items over time. Formally, four sets are identified: U and I as previously described, $T_{it} = \{t_{it} \in \mathbb{R}^{N \times T} | t_{it} = 0, 1\}$, and $T_{ut} = \{t_{ut} \in \mathbb{R}^{M \times T} | t_{ut} = 0, 1\}$. $T_{it} = 1$ denotes an interaction between item $\mathbf{i}$ and user $\mathbf{u}$ in the timestamp $\mathbf{t}$, and $T_{it} = 0$ means no interaction is observed. Likewise, $T_{ut} = 1$ denotes an interaction between user $\mathbf{u}$ and item $\mathbf{i}$ in the timestamp $\mathbf{t}$, and $T_{it} = 0$ means no interaction is observed. Assuming, the user preferences may repeat over time, a discretization on time is performed to avoid the set from being extremely sparse. The time is divided in $\mathbf{D}$ intervals in total, where $\mathbf{D} = \{0, 1, 2, ..., 23\}$ representing the daily hours when user $\mathbf{u}$ interected with item $\mathbf{i}$. We rewrite the Equation 3 as Equation 4 to incorporate the user and item interactions according to the timestamp.

$$\hat{R}_{uit} = F(\mathbf{u}, \mathbf{i}, \mathbf{it}, \mathbf{ut}|\Theta) = \mathbf{e}^1 \oplus \mathbf{e}^2 \oplus \mathbf{e}^3 \oplus \mathbf{e}^4 \tag{4}$$

$\mathbf{e}$ is the deep representation of the users and items embedding interaction learned by the n-th layer in the CNN.

Based on the aforementioned notations, we formally define the problem statement as: *Given a user* $\mathbf{u}$*, an item* $\mathbf{i}$*, and a timestamp* $\mathbf{t}$*, predict the list of preferred items by user* $\mathbf{u}$ *within* $\mathbf{i}$.

4 PROPOSED MODEL

CoNCARS has multiple layers as illustrated in Figure 2, which defines the prediction model according to Equation 2. We describe the detailed architecture of the model in the following sections.

4.1 Input Layer

Considering the model as a classifier, we aim to determine whether the user-item pair $(\mathbf{u}, \mathbf{i})$ should have a higher score than $(\mathbf{u}, \mathbf{j})$ for specific time $\mathbf{t}$. We define one-hot representation vectors for user $\mathbf{u}$ and item $\mathbf{i}$, which are further pre-processed as embedded vectors $\mathbf{P}$ and $\mathbf{Q}$, respectively. In addition to the embedded vectors, CoNCARS combines the binary interaction vectors $\mathbf{X}$ and $\mathbf{Y}$ from the observed interactions for $\mathbf{u}$ and $\mathbf{i}$ in time $\mathbf{t}$ according to Equation 1. The input layer feeds the embedding layer, which further generates four feature vectors for both $\mathbf{u}$ and $\mathbf{i}$.

4.2 Collective Embedding Layer

The layer aims to embed meaningful information regarding the user-item interaction. CoNCARS considers the user-item, item-user, user-time, and item-time vectors interactions. Formally, the Equation 5 defines each user feature vector and Equation 6 defines each item feature vector from the input layer as an embedding vector.

$$\mathbf{p}_u = \mathbf{P}^T \mathbf{u} \tag{5}$$

$$\mathbf{q}_i = \mathbf{Q}^T \mathbf{i} \tag{6}$$

where $\mathbf{P} \in \mathbb{R}^{U \times K}$ denotes the user embedding matrix and $\mathbf{Q} \in \mathbb{R}^{I \times K}$ denotes the item embedding matrix. K defines the number of hidden factors.

The interaction vectors $\mathbf{X}$ and $\mathbf{Y}$ are considered as another group of latent factors associated with temporal dynamics $\mathbf{x}_u$ for user $\mathbf{u}$ and $\mathbf{y}_i$ for item $\mathbf{i}$, respectively. Consider, the user $\mathbf{u}$ interacted with item $\mathbf{i}$ in different times $\mathbf{t}$, formally defined as:

$$\mathbf{y}_i = \frac{\sum_{\mathbf{i} \in Y} \mathbf{f}_i}{\sqrt{|Y|}} \tag{7}$$

$$\mathbf{x}_u = \frac{\sum_{\mathbf{u} \in X} \mathbf{g}_u}{\sqrt{|X|}} \tag{8}$$

where $\mathbf{y}_i$ is the *time-based item embedding* considering the $\mathbf{f}_i$ latent factors and $\mathbf{x}_u$ is the *time-based user embedding* considering the $\mathbf{g}_u$ latent factors. Assuming the distinct items may interfere in distinct user's preference, we rewrite user-item interaction applying an attention mechanism as:

$$\mathbf{y}_i = \sum_{\mathbf{i} \in Y} \alpha_i \mathbf{f}_i \tag{9}$$

$$\mathbf{x}_u = \sum_{\mathbf{u} \in X} \alpha_u \mathbf{g}_u \tag{10}$$

where α_i denotes the importance degree for item $\mathbf{i}$ rated by the user $\mathbf{u}$ at time $\mathbf{t}$ and α_u denotes the importance degree for user $\mathbf{u}$ interaction with item $\mathbf{i}$ at time $\mathbf{t}$. We define the attention score for item $\mathbf{i}$ as:

$$\mathbf{h}_i = tanh(\mathbf{W}_c \mathbf{f}_i + \mathbf{b}_a) \tag{11}$$

$$\alpha_i = \frac{exp(\mathbf{h}_i^T \mathbf{h}_a)}{\sum_{i \in Y} exp(\mathbf{h}_i^T \mathbf{h}_a)} \tag{12}$$

where $\mathbf{W}_c$ denotes the weight matrix, $\mathbf{b}_a$ denotes the bias vector, and $\mathbf{h}_a$ is the time contextual vector. $\mathbf{h_i}$ denotes a latent representation for $\mathbf{f}_i$ given by the pre-trained input vectors. α_i denotes the attention score of item $\mathbf{i}$, which calculates the similarity between $\mathbf{h}_i$ and the time contextual vector $\mathbf{h}_a$ via a softmax function. We apply a similar computation given by Equation 13 and 14 to calculate the *time-based user embedding* $\mathbf{x}_u$, such as:

$$\mathbf{h}_u = tanh(\mathbf{W}_c \mathbf{g}_u + \mathbf{b}_a) \tag{13}$$

$$\alpha_u = \frac{exp(\mathbf{h}_u^T \mathbf{h}_a)}{\sum_{\mathbf{u} \in X} exp(\mathbf{h}_u^T \mathbf{h}_a)} \tag{14}$$

where $\mathbf{W}_c$, $\mathbf{b}_a$, and $\mathbf{h}_a$ has the same semantic as in Equation 11 and 12.

4.3 Interaction Layer

The output from the embedding layer feeds the interaction layer to model correlations between user $\mathbf{u}$ and item $\mathbf{i}$ in time $\mathbf{t}$. We model the user-item interactions considering the temporal context and obtain four representation vectors as formally defined in Equation 15, 16, and 17.

$$\mathbf{e}^n = \phi_L^n(...(\phi_2^n(z_0[n]))...) \tag{15}$$

$$\phi_l^n = \sigma_l^n(\mathbf{W}_l^n \mathbf{z}_{l-1}^n + \mathbf{b}_l^n), \quad l \in [1, L] \tag{16}$$

$$\mathbf{z}_0 = [\mathbf{p}_u \otimes \mathbf{y}_i, \mathbf{p}_u \otimes \mathbf{q}_i, \mathbf{x}_u \otimes \mathbf{y}_i, \mathbf{x}_u \otimes \mathbf{q}_i] \tag{17}$$

where $n \in \{1, 2, 3, 4\}$; $\mathbf{e}^n$ is the deep representation of embedding interaction learned by the n-th layer in the CNN, ϕ_l^n is the l-th layer in the convolutional network n; $\mathbf{W}_l^j$ denotes the weight matrix; $\mathbf{z}_{l-1}^n$ denotes the user, item, and time correlation from the previous layer; $\mathbf{b}_l^j$ denotes the bias; σ_l^n denotes the activation function; and $\mathbf{z}_0$ denotes the concatenation for pair-wise correlations of user, item, and time collective embedding.

4.4 Convolutional Layer

The output of the interaction layer feeds the CNNs in the hidden layer. The model is composed of four CNNs, however as they are built using the same structure, we describe only one of them in this paper. CNN is responsible for learning non-linear interactions from the interaction layer. We define the convolutional layer as $\mathbf{c} = F_\Theta(\mathbf{I})$, where F_Θ is the model of hidden layers with parameters $\mathbf{\Theta}$, and $\mathbf{c}$ is the feature maps vector used to predict the final score.

CNN architecture consists of complex concepts such as stride and padding as explained by Goodfellow *et al.* [8]. In this paper, we focus on describing the most important settings concerning our network. The embedding size of the input layer CNN is 64×64. The channel has 6 hidden layers with 32 feature maps. The feature map $\mathbf{c}$ in the hidden layer l is represented as a $2D$ matrix of the interaction layer $\mathbf{I}^{le}$. The stride is set as $[1, 2, 2, 1]$ which represents the *example*, *height*, *width*, and *channel*, respectively. Considering these settings, the size of $\mathbf{I}^{le}$ is half of its previous layer $l-1$. We denote the feature maps of the first layer as a $3D$ tensor ε^l. The tensor in CNN allows CoNCARS to capture specific temporal patterns considering the common latent space. We define ε^l as:

$$\varepsilon^l = [\mathbf{e}_{i,j,s}^l]_{32 \times 32 \times 32}, \quad \text{where}$$
$$\mathbf{e}_{i,j,s}^l = ReLU(\mathbf{b}_1 + \sum_{a=0}^{c-1} \sum_{b=0}^{c-1} e_{2i+a, 2j+b} \cdot f_{1-a, 1-b, s}^1), \tag{18}$$

where $\mathbf{b}_1$ is the bias for the first layer, and $[f_{a,b,s}^1]_{2 \times 2 \times 32}$ is a $3D$ tensor is the convolutional filter used to generate the learning representation in the first layer. We use *ReLU* for activation function, which convert negative values to zero. The same operation is applied in the following layers. However, now there is a $3D$ tensor ε^l as input to the next layer $l + 1$ as formally described as:

$$\varepsilon^{l+1} = [\mathbf{e}^{l+1}_{i,j,g}]_{v\times v\times 32}, \quad \text{where} \quad 1 \le l \le 5, \quad v = \frac{64}{2^{l+1}}$$
$$\mathbf{e}^{l+1}_{i,j,g} = ReLU(\mathbf{b}_{l+1} + \sum_{a=0}^{c-1}\sum_{b=0}^{c-1} e^{l}_{2i+a,2j+b} \cdot f^{l+1}_{1-a,1-b,g}), \tag{19}$$

where $\mathbf{b}_{l+1}$ is the bias, while $[f^{l+1}_{a,b,s,g}]_{2\times 2\times 32\times 32}$ is the $4D$ convolutional filter for the following layer $l+1$. The output generated by the last layer is a tensor of dimension $1\times 1\times 32$ used by the prediction layer to calculate the final score with a weight vector $\mathbf{w}$.

4.5 Fusion Layer

The fusion layer is above the convolutional layer, being responsible for combining four latent feature vectors into a single one as specified in Equation 20.

$$\mathbf{z}_f = \mathbf{e}^1 \oplus \mathbf{e}^2 \oplus \mathbf{e}^3 \oplus \mathbf{e}^4 \tag{20}$$

where $\mathbf{z}_f$ is the concatenation of four latent interaction representations.

4.6 Prediction Layer

The prediction layer uses the output vector $\mathbf{z}_f$ from the fusion layer. The prediction score is calculated considering the recommender system as a classification problem as defined by Rendle *et al.* [23]. First, let $\hat{r}_{uit} = \mathbf{w}^T\mathbf{z}_f$, where vector $\mathbf{w}$ denotes the weights for the user-item-time interactions in $\mathbf{z}_f$. Then, CoNCARS calculate the prediction score for user $\mathbf{u}$ and item $\mathbf{i}$ in time $\mathbf{t}$, resulting in a tuple $(\mathbf{u},\mathbf{i},\mathbf{t})$. A positive result of the tuple $(\mathbf{u},\mathbf{j},\mathbf{t})$ denotes that $\hat{r}_{uit}$ should be larger than $\hat{r}_{ujt}$ to have the correct label of +1. On the other hand, a negative result has 0 as label.

4.7 Learning Algorithm

For a better optimization regarding top-N task, CoNCARS applies BPR [23] as the optimization method.

During the representation learning CoNCARS uses the mini-batch gradient descent to calculate the gradient with a small batch of samples, and formulate the training set as S in Equations 21, 22, 23, and 24.

$$\mathrm{S}_{p,x} = (p,x,x')|p \in \mathbf{P} \wedge x \in \mathbf{X}^+_p \wedge x' \in \mathbf{X}\backslash\mathbf{X}^+_p \tag{21}$$

$$\mathrm{S}_{p,q} = (p,q,q')|p \in \mathbf{P} \wedge q \in \mathbf{Q}^+_p \wedge q' \in \mathbf{Q}\backslash\mathbf{Q}^+_p \tag{22}$$

$$\mathrm{S}_{y,x} = (y,x,x')|y \in \mathbf{T} \wedge x \in \mathbf{X}^+_y \wedge x' \in \mathbf{X}\backslash\mathbf{X}^+_y, \tag{23}$$

$$\mathrm{S}_{y,q} = (y,q,q')|y \in \mathbf{T} \wedge q \in \mathbf{Q}^+_y \wedge q' \in \mathbf{Q}\backslash\mathbf{Q}^+_y, \tag{24}$$

where $\mathbf{p}$ denotes the users latent factors, $\mathbf{t}$ represents the time, $\mathbf{q}$ denotes the positive feedback, and $\mathbf{q}'$ represents the non-interacted items. The objective function is defined as in Equation 25.

$$\begin{aligned}\mathbf{L}_{BPR}(\mathrm{S}) = &\sum_{(p,x,x')\in \mathrm{S}_{px}} \ln\sigma(\mathbf{p_u}) \\ &+\lambda_1 \sum_{(p,q,q')\in \mathrm{S}_{pq}} \ln\sigma(\mathbf{q_i}) \\ +\lambda_2 \sum_{(y,x,x')\in \mathrm{S}_{yq}} \ln\sigma(\mathbf{y_i}) &+ \lambda_3 \sum_{(y,q,q')\in \mathrm{S}_{yx}} \ln\sigma(\mathbf{x_u}) - \lambda_\Theta||\mathbf{\Theta}||^2_F,\end{aligned} \tag{25}$$

where σ is the sigmoid function; $\mathbf{\Theta} = \{\mathbf{P},\mathbf{Q},\mathbf{X},\mathbf{Y}\}$ and $\lambda_\Theta = \{\lambda_4,\lambda_5,\lambda_6\,\lambda_7\}$. The gradient of the objective function is given by Equation 25 and maximized based on the first order derivatives as formulated in Equation 26.

$$\begin{aligned}\nabla_\Theta \mathbf{L}_{BPR}(\mathrm{S}) = \sigma(-\mathbf{p_u}\frac{\partial \mathbf{p_u}}{\partial \mathbf{\Theta}}) + \lambda_1\sigma(-\mathbf{q_i}\frac{\partial \mathbf{q_i}}{\partial \mathbf{\Theta}}) \\ +\lambda_2\sigma(-\mathbf{y_i}\frac{\partial \mathbf{y_i}}{\partial \mathbf{\Theta}}) + \lambda_3\sigma(-\mathbf{x_u}\frac{\partial \mathbf{x_u}}{\partial \mathbf{\Theta}}) - \lambda_\Theta||\mathbf{\Theta}||^2_F,\end{aligned} \tag{26}$$

CoNCARS is trained using a mini-batch as shown in Algorithm 1. First, we randomly initialized the parameters, then for each iteration, we compute the gradients according to Equation (26) with a batch of 5 positive samples and 5 negative samples to construct the preference pairs, and update the parameters. Finally, CoNCARS predicts the top-N items.

Algorithm 1: Mini-batch gradient descent method

Input : user vector $\mathbf{u}$, item vector $\mathbf{i}$ user-time vector $\mathbf{ut}$, item-time vector $\mathbf{it}$, regularization coefficients $\lambda_\mathbf{\Theta}$, batch size $\mathbf{b}$, learning rate η, number of epochs *epoch_max*, and convergence criteria.

Output : top-n prediction given by the prediction score $\hat{r}$.

```
initialize Θ randomly;
epoch = 0;
p,q,x,y are calculated according to Eq. 5, 6, 9, and 10;
while not converged && epoch < epoch_max do
    epoch+=1;
    shuffle all observed interaction;
    split all item-user interactions into b-size mini-batches;
    get the mini-batch in a sequential way;
    foreach mini-batch do
        foreach record in current batch do
            select 5 non-observed items q', and x' randomly
              from X\X_p^+, Q\Q_p^+, X\X_y^+, and Q\Q_y^+;
            add the negative samples to the current batch;
        end
        calculate the gradient according to Equation 26 for
          current batch to update the iterations for each
          CNN as defined in Eq. 18 and 19;
        Θ = Θ + η∇_Θ L_BPR(S);
    end
    concatenate the output according to Equation 20;
    calculate r̂ and predict the top-N items;
end
return the top-N items
```

4.8 Model Training

We train CoNCARS based on the BPR objective [23] with Adagrad optimizer [6]. Prior to computing the Algorithm 1, we pre-trained the embedding layer of CoNCARS using MF to avoid local optima solutions, since the objective function is non-convex. After obtaining the parameters $\mathbf{\Theta}$, the first epoch of CoNCARS is trained using $L2$-regularization. Once the network is trained CoNCARS ranks the personalized list of items **i** for a user **u** in time **t** based on the value of $\hat{r}_{uit} = \mathbf{w}^T \mathbf{z}_f$ and $\hat{r}_{ujt} = \mathbf{w}^T \mathbf{z}_f$ on the set of items.

5 EMPIRICAL EVALUATION

The empirical evaluation aims to answer the following research questions:

RQ1 Does the proposed method CoNCARS outperform the state-of-art methods for item recommendations?

RQ2 How do the number of feature maps **c** and λ affect the performance of CoNCARS?

5.1 Experimental Settings

Datasets. The experiments are performed on three datasets: MovieLens 1M[1], Yelp [2], and Pinterest [3]. They are public benchmark datasets in the recommender systems research community. We convert the datasets to implicit feedback following the Equation 1, where 1 denotes a user interaction with an item, and 0 denotes otherwise. Table 2 summarizes the statistics of each dataset. To address the sparseness of the datasets we apply the settings recommended by He *et al.* [12] and thereby only retrieving users with minimum 5 interactions.

Table 2: Statistics of the Datasets

Statistics	Movielens	Yelp	Pinterest
# of Users	6,040	25,815	55,187
# of Items	3,706	25,677	52,400
# of Interactions	1,000,209	730,791	1,5000,809
Sparsity	95.53%	99.89%	99.73%

- **Movielens 1M.** Is a movie dataset which has been widely used to evaluate the accuracy of collaborative filtering methods. This version has more than one million user interactions, where each user rated at least 5 movies. The dataset contains explicit feedback which we converted to implicit feedback to investigate the performance of our model in this domain. [17].
- **Yelp.** Is a restaurant dataset obtained during the Yelp Challenge. In order to provide new items to a user, we merged the repetitive ratings at different timestamps to the earliest one as recommended by He *et al.* [12].
- **Pinterest.** Is an implicit image dataset provided by Geng *et al.*[7] to evaluate content-based image recommendation. Each user-item interaction denotes whether the user has brought the image to her/his collection.

[1]https://grouplens.org/datasets/movielens/1m/
[2]https://github.com/hexiangnan/sigir16-eals
[3]https://sites.google.com/site/xueatalphabeta/academic-projects

5.2 Evaluation Protocol

The evaluation of the item recommendation task using implicit feedback is performed using an adapted version of leave-one-out evaluation protocol [4, 12, 28]. The latest user-item interaction is held-out considering it as a positive testing set, and the remaining interactions that the user did not rate before as training set. This strategy is computationally exhaustive. Therefore, to minimize the drawback, we follow the strategy applied by Koren *et al.* [17], where it randomly samples 100 items which are not interacted by the users. During the training step, CoNCARS ranks the list of all non-interacted items in the training set according to their prediction score. The top-N evaluation for item recommendation is based on the N number of items in the ranking list. The metrics used to evaluate the ranking list are Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) [12]. HR measures whether the testing item is in the top-N list, considering the presence of an item as a hit. NDCG, on the other hand, measures the quality of the prediction based on the position of an item in the recommended list. In both metrics, high values denotes a better performance.

5.3 Baseline Methods

The following state-of-art methods are used as baselines to evaluate CoNCARS performance.

- BPR [23] optimizes the MF model of Equation 2 by applying a pairwise loss function to learn from implicit feedback. We applied a fixed learning rate, which had the best performance in our experiments.
- CAMF [1] extends MF by using contextual features. In our experiments, we use time as a context to represent users interactions.
- TF [14] identifies the correlations between the user, item, and time using a three dimensional latent space. We set the embedding size with the best performance in our experiments.
- Collective Hybrid Non-negative Matrix Factorization (CHNMF) [5] decomposes the user-item interaction into a common latent space combining users interactions, item's attributes, and contextual information. In our experiment, we consider time as the only contextual feature and use the embedding size that had the best performance.
- NeuMF [12] pre-trains multi-layer perceptron (MLP) network and MF separately, and then ensembles those two models to predict the user's preferences. We set the embedding as 16 for each layer, since larger values caused overfitting and dropped the performance.
- Convolutional Matrix Factorization (ConvMF) [15] integrates convolutional neural network (CNN) into probabilistic matrix factorization (PMF). ConvMF aims to capture contextual information of documents and further enhances the rating prediction accuracy. In our experiments, ConvMF is trained to predict the top-N items for implicit feedback.

5.4 Parameters Settings

We developed the CoNCARS model in Python based on the Tensorflow framework. We tune the hyperparameters by keeping one training interaction for each user in the validation set. We optimize

Table 1: Top-N recommendation performance at $N = 10$ and $N = 20$. The bold font indicates the best results.

	Movielens				Yelp				Pinterest			
	HR@N		NDCG@N		HR@N		NDCG@N		HR@N		NDCG@N	
	N=10	N=20	N=10	N=20	N=10	N=20	N=10	N=20	N=10	N=20	N=10	N=20
CAMF	0.4816	0.4929	0.1999	0.2517	0.0535	0.1006	0.0514	0.0591	0.5738	0.5911	0.3337	0.3619
TF	0.5395	0.5548	0.3076	0.4189	0.0991	0.2061	0.0953	0.0982	0.6914	0.7028	0.4574	0.4777
CHNMF	0.5500	0.5711	0.3244	0.4365	0.1121	0.2150	0.1001	0.1013	0.7164	0.7502	0.4918	0.5009
BPR	0.5841	0.6573	0.3664	0.4395	0.1558	0.2607	0.1042	0.1379	0.7464	0.8025	0.5119	0.5371
NeuMF	0.6774	0.7300	0.4133	0.4470	0.1841	0.2967	0.1095	0.1538	0.7593	0.8770	0.5324	0.5520
ConvMF	0.6801	0.7558	0.4171	0.4494	0.1937	0.3005	0.1102	0.1547	0.7921	0.8995	0.5383	0.5636
CoNCARS	**0.6974**	**0.8422**	**0.4235**	**0.4596**	**0.2442**	**0.3751**	**0.1220**	**0.1588**	**0.8801**	**0.9691**	**0.5588**	**0.5749**

the regularization terms separately for the embedding layer, convolutional layer, and output layer in the range of $[0.001, 0.01, ..., 100]$. We set the embedding size as 64 for all baseline models and apply the BPR loss using Adagrad optimization with a learning rate of $\eta = 0.05$ and batch size of 512 for a fair comparison. We set the number of layers of NeuMF method between one and three as recommended by He *et al.* [12]. We pre-train the embedding layers of the neural baseline models using BPR and tuned their *L2*-regularization with the best values when achieving the best performance.

5.5 Performance Comparison (RQ1)

Table 1 summarizes the performance comparison for top-N recommendation in the datasets. The analysis considers $N = 10$ and $N = 20$ list of items as they are generally used to express the effectiveness of item recommendation.

CoNCARS outperforms ConvMF by 9% because it applies outer product rather than the element-wise product to capture the user, item, and time correlations. Furthermore, ConvMF does not use temporal features to identify the user's preferences according to different timestamps.

CoNCARS presents a relative improvement of 10.25% when compared to NeuMF. The better performance achieved by CoNCARS relies on three factors the use of time as a feature, use of the outer product in the interaction layer and CNN to learn the non-linear features. Time has proved to be an essential feature to model the user-item interactions, due to providing a more accurate sequence of items in the top-N list. The outer product among the user, item, and time embeddings captures better correlations among the embedded representations than the element-wise product.

NeuMF applies MLP to learn non-linear interactions between the user and item, while CoNCARS uses CNN for this purpose. MLP has a good representation ability as described by Hornik *et al.* [13], however it uses a large number of parameters. For example, assuming the embedding size as 32 and 1-layer MLP would have around 4 million parameters, while our 6-layer CNN has around 20 thousand parameters. The number of parameters required by MLP may drop its performance to converge to the optimal solution when compared with methods using CNN.

Despite CoNCARS and BPR applying the same loss function, we observe a relative improvement of 24% of the first one compared to the last. The use of a collective embedding layer together with a CNN layer by CoNCARS increased its accuracy for top-N recommendation.

By comparing CoNCARS with CHNMF, we observe an average relative improvement of 37.8%. The strategy of applying outer product by CoNCARS rather than point-wise squared loss as used by CHNMF, allowed CoNCARS to present a better learning process of user and item correlations over time. Furthermore, using CNN to identify non-linear correlations from the user, item, and time embeddings.

TF demonstrates a good learning representation for the user, item, and time embeddings. However, it fails to capture temporal dynamics due to the limitation of relying only on the current time-slot. Moreover, TF is not able to identify non-linear correlations. The relative improvement observed gives CoNCARS a better performance of 43%.

CoNCARS has an average relative improvement of 65.5% in comparison to CAMF. CAMF's poor performance may be caused by its use of time as a contextual feature in the conventional MF method, resulting in the inability to capture temporal patterns. Using conventional MF, CAMF is not able to capture the non-linear interactions as the neural models.

Among the baselines, ConvMF performs the best, proving the advantage of CNN to learn non-linear correlations between user-item interactions. CAMF had the weakest performance as it does not use multi-dimensional representation during the learning step. CHNMF and TF represent the correlations among user and items over time in a higher-dimensional representation. However, they do not optimize their objective function applying BPR learning model, causing a weaker performance than ConvMF.

CoNCARS has shown the best overall performance in comparison to the baselines considering the top-N recommendation task for $N = 10$ and $N = 20$. The good results proved by the experiments show that modeling time as a contextual factor generates an accurate prediction. Moreover, capturing the user and items interaction using outer product leads to a more effective user and item modeling. Finally, using four CNNs to learn non-linear correlations between the user and items considering different timestamps results in effective recommendations.

5.6 Hyper-parameters Analysis (RQ2)

CoNCARS has two hyper-parameters that can affect its effectiveness: λ and $\mathbf{c}$. λ defines the regularization term, and $\mathbf{c}$ denotes the

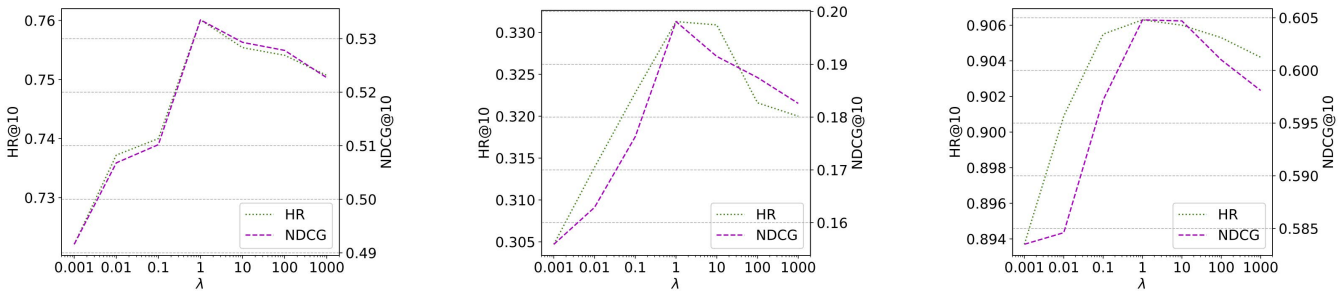

Figure 3: Performance of CoNCARS regarding to hyper-parameter λ on Movielens (left), Yelp (center), and Pinterest (right).

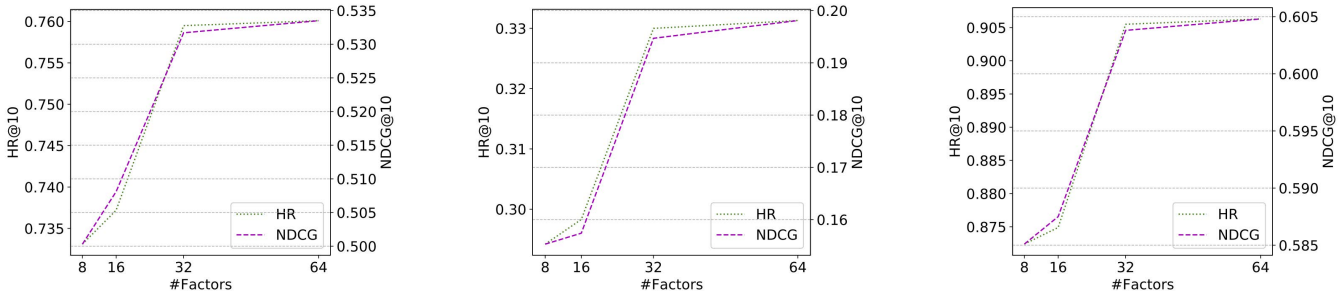

Figure 4: Performance of CoNCARS regarding the number of factors on Movielens (left), Yelp (center), and Pinterest (right).

number of feature maps per convolutional layer. We analyze the results for different values of λ and $\mathbf{c}$ in three datasets: Movielens, Yelp, and Pinterest.

The results regarding the hyper-parameter λ are shown in Figure 3 considering different values of λ. The curve observed in Figure 3 leads to interpreting that values of λ smaller than 1 demonstrate gradual improvements in the performance when we analyze the convergence curve for the Movielens, Yelp, and Pinterest datasets. However, values larger than 1 keep the performance stable until a value of $1,000$.

Figure 4 illustrates the top-N effectiveness for different values of hyper-parameter $\mathbf{c}$. For all values of $\mathbf{c}$, CoNCARS presents a gradual increase in the learning curve and finally reaches the convergence point. However, for $\mathbf{c}$ equals to 32 or 64 the convergence time is smaller than assuming values of 8 or 16. The slight difference in the time convergence leads us to conclude that different values of $\mathbf{c}$ achieve similar performance.

We can conclude that despite the difference in the convergence time, CoNCARS can consider different values for the hyper-parameters λ and $\mathbf{c}$ without compromising its accuracy, making CoNCARS suitable for real-world usage.

6 CONCLUSION

We proposed a collective embedding for neural context-aware recommender systems for item recommendations using implicit feedback. The paper explains how CoNCARS uses temporal information in the feature embedding and why it is an important feature for the recommendation task. Furthermore, the paper described how the joint embedding and CNNs improve the learning representation of the user and item interaction over time. Finally, we present the experimental results and CoNCARS' performance regarding the hyper-parameters variance.

CoNCARS has proved to be useful for top-N item recommendations considering implicit feedback. Moreover, pre-training the model using MF with temporal information has presented good performance when used as input data to CNNs. The interaction layer allows CoNCARS to capture correlations between user, item, and time. Furthermore, using the CNNs, CoNCARS can learn non-linear deep representations among user and item embedding over time.

The results of our experiments in the three benchmark datasets, has presented the efficiency of CoNCARS for predicting the user's preference. Moreover, our model has shown an stable performance assuming different values for the hyper-parameters.

In the future, we would like to investigate other features that may influence the prediction of the user's preferences, such as item's reviews [9] as richer contextual information due to a broader description given by users and user's viewpoints [22]. Another research direction is to model the user's social relations and influence [24] since similar users tend to influence the decision-making of other users.

ACKNOWLEDGMENTS

The authors wish to acknowledge the financial support and the fellow scholarship given to this research from the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (grant# 206065/2014-0).

REFERENCES

[1] Linas Baltrunas, Marius Kaminskas, Bernd Ludwig, Omar Moling, Francesco Ricci, Aykan Aydin, Karl-Heinz Lüke, and Roland Schwaiger. 2011. Incarmusic: Context-aware Music Recommendations in a Car. In *E-Commerce and Web Technologies*. Springer, 89–100.

[2] Preeti Bhargava, Thomas Phan, Jiayu Zhou, and Juhan Lee. 2015. Who, What, When, and Where: Multi-Dimensional Collaborative Recommendations Using Tensor Factorization on Sparse User-Generated Data. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. 130–140.

[3] Pedro G. Campos, Fernando Díez, and Iván Cantador. 2014. Time-aware Recommender Systems: A Comprehensive Survey and Analysis of Existing Evaluation Protocols. *User Modeling and User-Adapted Interaction* 24, 1-2 (Feb. 2014), 67–119.

[4] Weiyu Cheng, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. 2018. DELF: A Dual-Embedding based Deep Latent Factor Model for Recommendation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 3329–3335.

[5] Felipe Costa and Peter Dolog. 2018. Hybrid Learning Model with Barzilai-Borwein Optimization for Context-aware Recommendations. In *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS 2018, Melbourne, Florida USA., May 21-23 2018*. 456–461.

[6] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159.

[7] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. 2015. Learning Image and User Features for Recommendation in Social Networks. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. 4274–4282.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[9] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. TriRank: Review-aware Explainable Recommendation by Modeling Aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. 1661–1670.

[10] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial Personalized Ranking for Recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. 355–364.

[11] Xiangnan He, Min-Yen Kan, Peichu Xie, and Xiao Chen. 2014. Comment-based Multi-view Clustering of Web 2.0 Items *(WWW '14)*. 771–782.

[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. 173–182.

[13] Kurt Hornik. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.* 4, 2 (March 1991), 251–257.

[14] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. 79–86.

[15] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. 2016. Convolutional Matrix Factorization for Document Context-Aware Recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. 233–240.

[16] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[17] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*. 426–434.

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

[19] Jialu Liu, Chi Wang, Jing Gao, and Jiawei Han. 2013. Multi-view Clustering via Joint Nonnegative Matrix Factorization. In *Proc. of Intl. Conf. on Data Mining*. SIAM, 252–260.

[20] Martin Pichl, Eva Zangerle, and Günther Specht. 2017. Improving Context-Aware Music Recommender Systems: Beyond the Pre-filtering Approach. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR '17)*. 201–208.

[21] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4 (July 2018), 66:1–66:36.

[22] Zhaochun Ren, Shangsong Liang, Piji Li, Shuaiqiang Wang, and Maarten de Rijke. 2017. Social Collaborative Viewpoint Regression with Explainable Recommendations. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. 485–494.

[23] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. 452–461.

[24] Muhammad Aamir Saleem, Felipe Soares da Costa, Peter Dolog, Panagiotis Karras, Toon Calders, and Torben Bach Pedersen. 2018. Predicting Visitors Using Location-Based Social Networks. In *MDM*. 245–250.

[25] Martin Saveski and Amin Mantrach. 2014. Item Cold-start Recommendations: Learning Local Collective Embeddings *(RecSys '14)*. 89–96.

[26] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. 153–162.

[27] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM, 211–222.

[28] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3203–3209.