Behavior-Aware CNN for Sequential Heterogeneous One-Class Collaborative Filtering

Jing Xiao

Abstract

In recent years, recommender systems have played an important role in solving the information overload problem. With the continuous development of recommendation technology, sequential heterogeneous one-class collaborative filtering (SHOCCF) has become an emerging and crucial task and has received more and more attention. RNN is known for its effectiveness in modeling sequential data, and many existing classical models are based on RNN or its variants aiming to capture the transitions between items. It should be noticed that the output of RNN at each time step depends on the past actions, which relegates the efficiency of parameter optimization and easily brings the problem of vanishing gradient. In contrast, CNN does not stipulate that information should be transmitted sequentially along the time steps, which is of greater flexibility. As a response, in this paper, we propose a novel solution named Caser++ that applies CNN to model heterogeneous sequences in SHOCCF. Given the historical interaction sequences between users and items, the two main goals of our model are to introduce information that represents different behaviors and to learn the sequential patterns via different convolutional filters. Moreover, we use a filtering mechanism to reduce the noise that may exist in users' examination behaviors. Finally, we conduct experiments on four public datasets and showcase the effectiveness of our Caser++.

Keywords：Sequential Heterogeneous One-Class Collaborative Filtering, CNN, Behavior Types.

1 Introduction

With the rapid development of the Internet, recommender systems have become an indispensable part of today's life. It is widely used in major online platforms, which can greatly alleviate the problem of information overload and help users pick up the information they desire from the flood of data. Traditional recommendation methods mainly include collaborative filtering (CF) and content-based (CB) approaches. Most of them focus on mining the static associations between users and items from their historical interactions, thereby capturing users' stable long-term preferences. However, in the real world, users' behaviors are inherently sequential, and their preferences generally change dynamically over time. For example, a user used to like to listen to upbeat songs, but the latest songs are always melancholy since she just experienced a lovelorn. Furthermore, in most practical life scenarios, users have diverse behaviors on items, such as browses, adds to shopping carts and purchases. To model the above phenomenon, sequential heterogeneous one-class collaborative filtering (SHOCCF) came into being, which makes full use of sequential information and aims to predict the items a user will prefer in the future.

For the study of heterogeneous data, the early algorithms are SVD++[1] and FM[2], which utilize implicit feedback to assist rating data, thereby improving the accuracy of rating predictions. Later, a line of work seeks

to apply two types of feedback to provide users with personalized recommendations. RoToR[3] learns from users' browse feedback and transfers that information into purchase feedback. TJSL[4] integrates the similarity of purchase feedback and browse feedback into a unified prediction formula based on FISM[5]. EHCF[6] further considers the relationship between different feedbacks and adopts an efficient training strategy. These models can achieve better performance than models based on homogeneous data as they can acquire more information. However, they ignore the sequential dependencies, and cannot model a user's dynamic preferences well.

Existing studies on SHOCCF are rare but very noteworthy. RLBL[7] uses LBL to capture local features of sequences, and aggregates these features through an recurrent neural networks (RNN) layer to obtain global features. RIB[8] encodes each item and the corresponding behavior, and uses GRU to capture the transitions between different behaviors. Although RNN has natural advantages in processing sequential data, it relies on the hidden states of the entire history, leading to the problems of vanishing gradient and difficulty in parallel training. In contrast, convolutional neural network (CNN) only considers the current action and achieves excellent results in areas such as image and video processing. A previous classic work Convolutional Sequence Embedding (Caser)[9] treats the item embedding matrix as an "image" and applies convolution operations to model high-order MCs, which defeats many models of the same period. However, it does not consider the effect of different behaviors on preference prediction.

In this paper, we propose a novel CNN-based model for the SHOCCF problem named Caser++ since it is an extension of Caser[9]. We first use the position embedding vector and the behavior-aware embedding vector to represent the position information and the user's different behaviors, respectively. In addition, we noticed that some behaviors of users have semantic uncertainty, and direct use may introduce noise, so we adopt some filter-enhanced blocks to reduce the noise in the data[10]. Inspired by Caser, we use horizontal and vertical convolution operations to model the heterogeneous data. In this way, based on the rich and clean heterogeneous data, we further fully capture the user's preferences through convolution operations.

2 Related works

2.1 General Recommendation

Traditional recommender systems do not consider sequential order information, in which collaborative filtering methods occupy an essential position, with the central idea of "bird of a feather, flock together.". It is mainly divided into neighborhood-based methods, matrix factorization (MF), and hybrid-based methods. Among them, MF-based methods are widely used, which factorize the user-item interaction score matrix into latent representations of users and items.

2.2 Sequential One-Class Collaborative Filtering

Early works on SOCCF mostly rely on the Markov chains (MCs) to capture the dependencies of items in a sequence. For example, factorizing personalized Markov chains (FPMC)[11] combines the power of Matrix Factorization (MF) and first-order MCs to model both general tastes and sequential patterns. Based on this idea, some improved versions with high-order MCs have been proposed to consider more previous actions.

Fusing similarity models with Markov chains (Fossil)[12] integrates higher-order Markov chains and factored item similarity to capture short-term dynamic. Translation-based recommendation (TransRec)[13] embeds items into a ”translation space” and models users as a translation vector to learn the transition between the current and the following items. However, all these methods only consider the influence of recent items on users' subsequent behavior and ignore the power of long historical sequences.

Recently, RNN has made great achievements in processing sequential data, and a series of models based on RNN and its variants, e.g., gated recurrent Unit (GRU) and long short-term memory (LSTM) have emerged in sequential recommendation[14-17]. Among them, session-based GRU recommendation (GRU4Rec)[14] is a pioneering work that applies RNN to session-based recommendation. It adopts GRU to obtain a user's representations and also utilizes session-parallel mini-batches to speed up training. In addition to RNN, there are some other deep learning models that have been designed for sequential recommendation. Caser[9] learns rich sequential patterns (i.e., point-level, union-level and skipping patterns) by sliding horizontal and vertical convolution filters to get the user's interest. Self-attentive sequential recommendation (SASRec)[18] leverages the self-attention mechanism to gain item-to-item correlations and achieves state-of-the-art performance. Fusing item similarity models with self-attention networks (FISSA)[19] extracts a user's long-term and short-term preferences by fusing self-attention mechanism and item similarity model. Furthermore, graph neural network (GNN) has been designed to model complex transitions between sequential items by constructing sequences of user-item interactions into graph structures and then going on to learn the embedding representation of each node in the graph to mine the sequential information.[20-22]. For example, session-based recommendation with GNN (SRGNN)[20] proposes to apply a gated graph neural network (GGNN) and a self-attention network to model session sequences. However, these models only consider a single type of user behavior and do not sufficiently extract knowledge from the user-item interaction sequence, largely missing some helpful information.

2.3 Sequential Heterogeneous One-Class Collaborative Filtering

Researches on SHOCCF aim to exploit various types of user behaviors to tap into users' complex shopping interests and enhance the model's predictive power for target behaviors. There are relatively few studies on SHOCCF, which are mostly based on deep learning. Recurrent log-bilinear model (RLBL)[7] divides a heterogeneous sequence into several time windows, combines the advantages of RNN and LBL to capture a user's long-term and short-term preferences respectively, and introduces behavior-specific transition matrices to model different types of behaviors. Recommendation with sequences of micro behaviors (RIB)[8] encodes each item and behavior in a user's historical interaction sequence into a corresponding embedding vector, captures the sequential information through GRU, and uses an attention layer to obtain the varied effects of different behaviors. Behavior-intensive neural network (BINN)[23] proposes a novel context-aware LSTM network, which can simultaneously memorize the item and behavior information in a heterogeneous sequence. Based on the above analysis, we can see that most existing works are based on RNN to model heterogeneous sequences. However, it should be noticed that at each time step, RNN accepts the hidden state of the last step and current action to obtain the current hidden state, which makes it difficult to train the model in parallel.

Moreover, adjacent actions in a sequence are not necessarily correlated. Since CNN only considers the current input, and the CNN-based model Caser has made remarkable achievements, we focus on generalizing CNN to learn a user's preferences for the studied problem and adopt Caser as our backbone model.

3 Proposed method: Caser++

In this section, we describe our proposed Caser++ model for the SHOCCF problem in detail. We have a set of interaction sequences $\mathcal{S}_u=\{(i_u^1,f_u^1),...,(i_u^t,f_u^t),...,(i_u^{|\mathcal{S}_u|},f_u^{|\mathcal{S}_u|})\}$, where i_u^t and f_u^t represent the item and the corresponding behavior w.r.t. user u at time step t, respectively. Our goal is to recommend top-N items from all the items for the user u that he or she will most likely to purchase at time step $t+1$.

The highlight of our model is that we design a behavior-aware embedding layer to jointly the information of different behavior types[24] and position. Moreover, considering the impact of noise on the model effectiveness, we use a filter layer in the frequency domain to simulate the filtering mechanism to denoise the data[10]. As illustrated in Fig. 1, our Caser++ consists of four main components, including an embedding layer, a filter layer, a convolutional layer and a fully-connected layer, of which the last two layers are exactly the same as that of Caser.

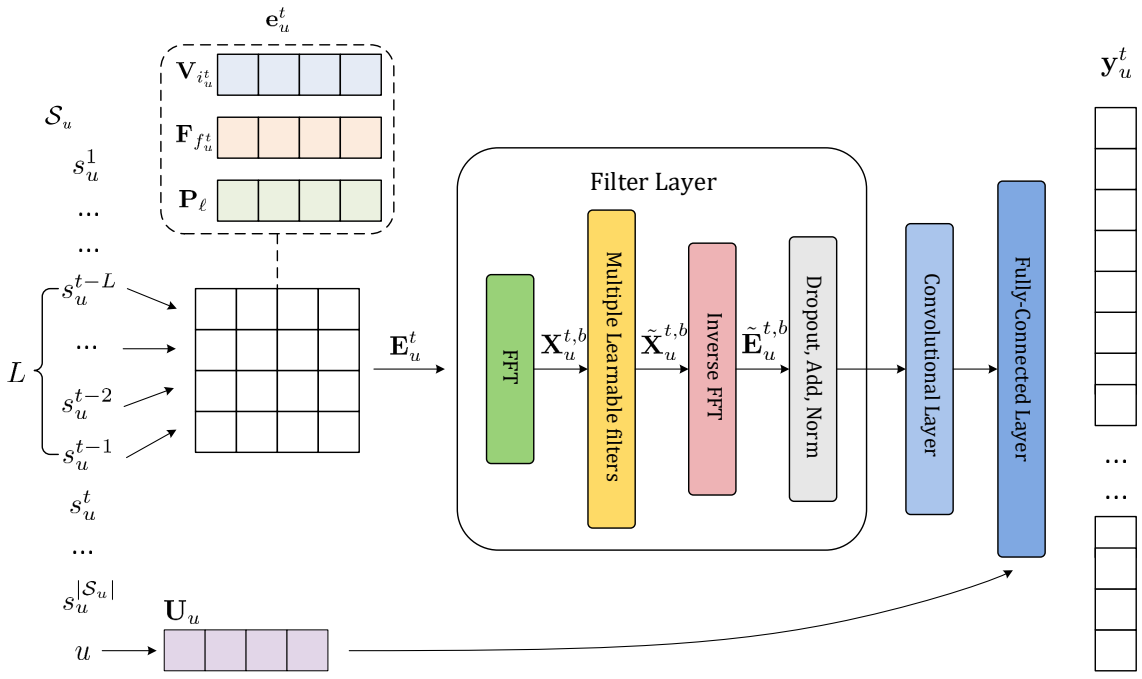

Figure 1: The model architecture of Caser++.

3.1 Embedding Layer

First of all, we create $\mathbf{V}_{i_u^t} \in \mathbb{R}^{1\times d}$ and $\mathbf{U}_u \in \mathbb{R}^{1\times d}$ to be the embedding vector of item i_u^t and user u, respectively, where d is the embedding size. Next, we can see that different behaviors represent different levels of user intentions, e.g., purchase is positive feedback stronger than examination. To capture such fine-grained user preferences, we use a behavior type embedding vector $\mathbf{F}_{f_u^t} \in \mathbb{R}^{1\times d}$ to model the different types of behaviors of user u in a heterogeneous sequence.

In addition, the behaviors at different positions usually affect a user's next purchase differently. For example, although a user was not interested in electronic products in the past, when he or she buys a computer,

he or she is likely to purchase some related accessories such as a mouse and headphones. While convolution operation is not sensitive to sequence order, we thus generate a learnable position embedding vector $\mathbf{P}_\ell \in \mathbb{R}^{1\times d}$, where ℓ represents the ℓ-th position.

In this way, we make use of the sequential information, and the final input embedding vector $\mathbf{e}_u^t \in \mathbb{R}^{1\times d}$ for the item i is obtained by summing these three vectors. Given the interaction sequence of user u, we form the input embedding matrix $\mathbf{E}_u^t \in \mathbb{R}^{L\times d}$ via stacking the L embedding vectors $\left\{\mathbf{e}_u^{t-L}, \mathbf{e}_u^{t-L+1}, ..., \mathbf{e}_u^{t-1}\right\}$ before time step t:

$$\mathbf{E}_u^t = \left[\mathbf{V}_{i_u^{t-L+\ell}} + \mathbf{F}_{i_u^{t-L+\ell}} + \mathbf{P}_\ell\right]_{\ell=1,2,...,L} \tag{1}$$

3.2 Filter Layer

In real life, heterogeneous sequences often have some noise. For instance, a user may accidentally click some items that are not of interest to him or her. Thus, we add a filter layer after the embedding layer, which is composed of multiple filter-enhanced blocks stacked[10]. At the b-th block, the embedding matrix is converted from the time domain to the frequency domain through a fast Fourier transform (FFT), and then multiplied by a learnable filter of the same size in the frequency domain, i.e., $\mathbf{K}$, which is calculated as[10]:

$$\tilde{\mathbf{X}}_u^{t,b} = \mathbf{K} \odot \mathcal{FFT}\left(\mathbf{E}_u^{t,b}\right) \tag{2}$$

where $\mathcal{FFT}$ represents the one-dimensional FFT, and $\odot$ denotes the element-wise product. Notice that $\mathbf{K}$ is continuously updated in each training epoch, resulting in adaptive filtering. With the filtered sequence $\tilde{\mathbf{X}}_u^{t,b}$, the inverse Fourier transform is applied to it to obtain the sequence representations in the time domain after removing the noise. And by doing this, we can obtain a high-quality embedding matrix:

$$\tilde{\mathbf{E}}_u^{t,b} = \mathcal{FFT}^{-1}\left(\tilde{\mathbf{X}}_u^{t,b}\right) \tag{3}$$

Furthermore, as a rule of thumb, the deeper the network, the harder it is to train. We add dropout[25] and residual connection[26] after the filter layer to further alleviate the overfitting and effectively mitigate the vanishing gradient issue. Finally, we adopt the layer normalization[27] to make the model more robust:

$$\tilde{\mathbf{E}}_u^t = LayerNorm\left(\mathbf{E}_u^{t,b} + Dropout\left(\tilde{\mathbf{E}}_u^{t,b}\right)\right) \tag{4}$$

3.3 Convolutional Layer

A user's behavior is often union, as we can know from experience that he or she is more likely to buy fried chicken and coke than to buy one of them separately. Thus, we regard the matrix $\tilde{\mathbf{E}}_u^t$ as an "image" G, and adopt some horizontal and vertical convolutional operations to capture the sequential patterns as the same with Caser[9]. To be specific, the horizontal filters slide from top to bottom on G to capture the union-level effect of the previous L items on the subsequent items. Given n horizontal filters $\mathbf{F}_k^{(h)} \in \mathbb{R}^{h\times d}, (1 \leqslant k \leqslant n)$ with $h \in \{1, 2, ..., L\}$ as the height of a filter, we can get the i-th convolution value as:

$$\mathbf{c}_{k,i} = \phi_c\langle \mathbf{E}_{i:i+h-1}^{(u,t)}, \mathbf{F}_k^{(h)} \rangle \tag{5}$$

where $\phi_c(\cdot)$ denotes the activation function of the horizontal convolutional layers, and $\langle\rangle$ is the inner product. When all filters slide to the bottom, the final convolution result is obtained:

$$\mathbf{c}_k = [\mathbf{c}_{k,1}\ \mathbf{c}_{k,2}\ \ldots\ \mathbf{c}_{k,L-h+1}] \tag{6}$$

Moreover, to capture the most important features extracted by the filter, we take max pooling for each $\mathbf{c}_k$ and then concatenate them together to get the output $\mathbf{o} \in \mathbb{R}^{1\times n}$:

$$\mathbf{o} = [max(\mathbf{c}_1), max(\mathbf{c}_2), ..., max(\mathbf{c}_n)] \tag{7}$$

Similarly, the vertical filters slide from left to right on G to capture the point-level influence. Notice that the size of each vertical filter is fixed to $L \times 1$. There are $\tilde{n}$ vertical filters $\tilde{\mathbf{F}}_k \in \mathbb{R}^{L\times 1}$, among which the k-th convolution result is:

$$\tilde{\mathbf{c}}_k = [\tilde{\mathbf{c}}_{k,1}\ \tilde{\mathbf{c}}_{k,2}\ \ldots\ \tilde{\mathbf{c}}_{k,d}] \tag{8}$$

And the output of the vertical convolutional layer $\tilde{\mathbf{o}} \in \mathbb{R}^{1\times d\tilde{n}}$ is given by:

$$\tilde{\mathbf{o}} = [\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2, ..., \tilde{\mathbf{c}}_{\tilde{n}}] \tag{9}$$

After the above convolution operations, the model has learned rich sequential patterns.

3.4 Fully-Connected Layer

The two outputs of the convolutional layer are then concatenated and sent to the fully-connected layer to obtain the vector representing a user's short-term preference[9]:

$$\mathbf{z} = \phi_a\left(\mathbf{W}\begin{bmatrix}\mathbf{o}\\ \tilde{\mathbf{o}}\end{bmatrix} + \mathbf{b}\right) \tag{10}$$

where $\phi_a(\cdot)$ represents the activation function of the fully connected layer, and $\mathbf{W} \in \mathbb{R}^{(n+d\tilde{n})\times d}$ and $\mathbf{b} \in \mathbb{R}^{1\times d}$ are the weights and biases to be learned. Furthermore, to simultaneously capture a user's long-term preferences, we concatenate $\mathbf{U}_u$ and $\mathbf{z}$ into a second fully-connected layer[9]:

$$\mathbf{y}_u^t = \mathbf{W'}\begin{bmatrix}\mathbf{z}\\ \mathbf{U}_u\end{bmatrix} + \mathbf{b'} \tag{11}$$

where $\mathcal{J}$ denotes the collection of all items, and $\mathbf{W}' \in \mathbb{R}^{(2d)\times|\mathcal{J}|}$ and $\mathbf{b}' \in \mathbb{R}^{1\times|\mathcal{J}|}$ are the weights and biases to be learned. The value of $y_u^{t,i}$ reflects the preference of user u on item i at time step t. And we can get a specific score for the likelihood that item i will be the next interacted item as:

$$\tilde{p}_u^{t,i} = \frac{e^{y_u^{t,i}}}{\sum_{j=1}^{|\mathcal{J}|} e^{y_u^{t,j}}} \tag{12}$$

Finally, we train our Caser++ with the cross-entropy loss function as follows:

$$\mathcal{L} = -\sum_{i=1}^{|\mathcal{J}|} Y_u^{t,i} \log\left(\tilde{p}_u^{t,i}\right) \tag{13}$$

where $Y_u^{t,i}$=1 only if the item i is the true interacted item of the user u at time step t, and $Y_u^{t,i}$=0 otherwise.

Table 1: Statistics of the processed datasets.

Dataset	#Users	#Items	#Examinations	#Purchases	Avg. length	Density
ML1M	5,645	2,357	628,892	223,305	150.96	6.41%
Rec15	36,917	9,621	446,442	223,265	18.41	0.45%
UB	20,858	30,793	470,731	136,250	29.10	0.09%
Tmall	17,209	16,176	831,117	240,901	62.29	0.38%

4 Experiment details

In this section, we first set up the experiments, and then we conduct empirical studies to answer the following three questions.

- **RQ1**: How does our Caser++ perform when compared with the other baselines?
- **RQ2**: How does the number of filter-enhanced blocks affect the performance?
- **RQ3**: How do the components of our Caser++ affect the performance?

4.1 Comparing with released source codes

The source codes of Caser are given in the original paper, but instead of using the source code, we wrote all the code for the reproduction and improvement of Caser ourselves.

4.2 Experimental Setup

4.2.1 Datasets.

We evaluate the methods on the following four public datasets[24]. **ML1M**: A well-known dataset in the field of recommender systems, which is about users' ratings on movies. Following as[28], we consider the (user, item) pairs with a rating = 5 as purchases and the rest as examinations. **Rec15**: A dataset used for RecSys Challenge 2015, which consists of activities from a big e-commerce business within six months. It contains 33,000,944 examinations and 1,150,753 purchase between 36,917 users and 52,739 items. **UB**: An e-commerce dataset with multiple behaviors provided by Taobao.com from November 25, 2017 to December 3, 2017. It contains about 100 million interactions between 987,994 users and 4,162,024 items, for which we only keep the examinations and purchases. **Tmall**: Another public e-commerce dataset provided by Tmall.com, which contains users' shopping logs in the past six months before and on the "Double 11" day. It contains 48,550,713 examinations and 3,292,144 purchase between 424,170 users and 1,090,390 items. Furthermore, considering the impact of the specialty of the "Double 11" day on users' behavior, we remove the shopping data from that day. Same as above, we only keep the examinations and purchases.

For the studied SHOCCF problem, we preprocess these datasets as follows: 1) we discard the cold-start items with fewer than 5 records in ML1M and Rec15, 10 in UB, and 20 in Tmall; 2) we discard the cold-start users with fewer than 10 records in Tmall and 5 in the other two datasets; 3) we order each dataset by the timestamps, and for repeated (user, item, behavior) triples in a sequence, we only keep the first one; 4) following the LOO (leave-one-out)-based validation, in each dataset, we take the last purchase as the test set, the penultimate purchase as the validation set, and the rest as the training set. The statistics of the processed datasets are shown in Table 1.

4.2.2 Evaluation Metrics.

To evaluate the recommendation performance of our proposed model, we adopt two common top-N metrics, i.e., hit ratio (HR) and normalized discounted cumulative gain (NDCG), which are described as follows:

- **HR@*k*** is defined as the fraction of cases that the ground-truth next item is among the top k items recommended.
- **NDCG@*k*** is a position-aware metric, which emphasizes the rank of items, i.e. the top-ranked items are more important.

Note that we empirically set N to 10, since the user always expects to find the target item from the first few items in the list.

4.2.3 Baselines.

To fully demonstrate the effectiveness of our proposed model Caser++, we compare it with some state-of-the-arts from two lines of research, i.e., sequential one-class collaborative filtering methods and sequential heterogeneous one-class collaborative filtering methods.

Sequential One-Class Collaborative Filtering Methods

- **FPMC[11].** It uses both matrix factorization and first-order MCs to predict users' preferences and recommend items [Rendle et al., 2010].
- **Fossil[12].** It integrates the factored similarity model into a high-order MCs to better capture users' short-term preferences [He et al., 2016].
- **GRU4Rec[14].** It applies GRU-based RNN to model user sequences for session-based recommendation [Hidasi et al., 2016].
- **Caser[9].** It takes the embedding matrix of L previous items as an "image" and adopts convolutional operations to model high-order MCs [Tang et al., 2018].

Sequential Heterogeneous One-Class Collaborative Filtering Methods

- **RLBL[7].** It combines RNN and LBL (log-bilinear) to capture long-term and short-term preferences [Liu et al., 2017].
- **RIB[8].** It uses micro behaviors and their effects in a sequence to gain a deeper understanding of users [Zhou et al., 2018].

4.2.4 Implementation Details.

We implement all the baseline methods and our Caser++ using Pytorch. For fair comparison, we use the Adam optimizer to train the model with the learning rate of 1e-3 and get the best model via early stopping w.r.t. HR@10 on the validation set. For each method, the ℓ_2 regularizer is searched from {0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1}, the dropout rate is searched from {0.2, 0.5}, and we set the sequence length L to

Table 2: Recommendation performance of four SOCCF methods, two SHOCCF methods and our Caser++. The last row 'Improve' shows the percentage improvement of our Caser++ over Caser, and the best performance on each dataset are marked in bold. Notice that the results of FPMC and Fossil are from[24], and the results of RIB are from[29].

Method	ML1M		Rec15		UB		Tmall	
	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
FPMC	0.1086	0.0510	0.3829	0.2102	0.0467	0.0249	0.0352	0.0191
Fossil	0.1114	0.0528	0.3736	0.2029	0.0508	0.0270	0.0437	0.0241
GRU4Rec	**0.1357**	**0.0663**	0.3616	0.1940	0.0577	0.0311	0.0762	0.0436
Caser	0.1015	0.0466	0.6631	0.3470	**0.1103**	**0.0563**	0.0633	0.0342
RLBL	0.1258	0.0592	0.3411	0.1587	0.0491	0.0254	0.0491	0.0254
RIB	0.1302	0.0549	0.3668	0.1921	0.0660	0.0370	0.0776	**0.0454**
Caser++	0.1185	0.0549	**0.7424**	**0.3840**	0.1086	0.0555	**0.0785**	0.0442
Improve	16.75%	17.81%	11.96%	10.66%	-1.54%	-1.42%	24.01%	29.24%

50, the batch size to 128, and the embedding dimension d to 64 in all methods. For all the baseline methods, we configure the other hyper-parameters following the corresponding original papers. To be specific, for Caser, we set the size of the horizontal filter to 16, the size of the vertical filter to 4, and the height of the horizontal filter to {2, 3, 4}. For RLBL, we set the window length to 5. For our Caser++, we search for the optimal number of filter-enhanced blocks in {1, 2, 3}. We will release the source code and scripts.

4.3 Performance Comparison

Table 2 shows the recommendation performance of our Caser++ compared with all the baselines on the four datasets (**RQ1**). Based on the results, we can draw the following main conclusions: 1) Our Caser++ outperforms the original Caser significantly on most datasets, demonstrating that multi-behavior data can more sufficiently capture users' preferences, and that apply learnable filters to denoise the input data can effectively mitigate overfitting problems and improve model performance. 2) Caser beats Caser++ on UB, we suspect that this is likely owing to Caser++ having more parameters than Caser, making the model more complicated. Notice that UB is highly sparse, which may favor a simple model. 3) Fossil achieves better performance than FPMC in ML1M, UB and Tmall, which indicates that higher-order MCs are more beneficial for sequential recommendation. 4) Caser has a competitive performance with GRU4Rec in Rec15 and UB, which we believe is mainly due to the contribution of CNN to distill information from short-term sequences. 5) For the two RNN-based models, i.e., GRU4Rec and RIB, RIB is superior to GRU4Rec on Rec15, UB, and Tmall. It indicates that making full use of multiple interaction behaviors often leads to better performance, i.e., the SHOCCF methods are better than the SOCCF methods. 6) RLBL and RIB perform significantly better than Caser and Caser++ on ML1M, but far worse than them on Rec15 and UB. This is due to the higher sequential intensity of ML1M compared with other datasets, and the RNN-based model has a natural advantage in handling sequential data.

Since the number of filter-enhanced blocks often has a significant impact on the model performance, we study its impact on the four datasets (**RQ2**), which are shown in Fig. 2. It can be observed clearly that: for ML1M, Rec15 and Tmall, our Caser++ achieves the best performance when the number is 3, while for UB, using two blocks is the most appropriate.

Table 3: Ablation studies of our Caser++ by removing different components.

Method	Rec15	UB	Tmall
Default	0.3840	0.0555	0.0442
- BE & FEB	0.3416	0.0528	0.0354
- PE & FEB	<u>0.3406</u>	<u>0.0512</u>	<u>0.0353</u>
- BE & PE	0.3676	0.0536	0.0415
- BE	0.3731	0.0550	0.0452
- PE	0.3654	0.0528	0.0426
- FEB	*0.3602*	*0.0518*	*0.0353*

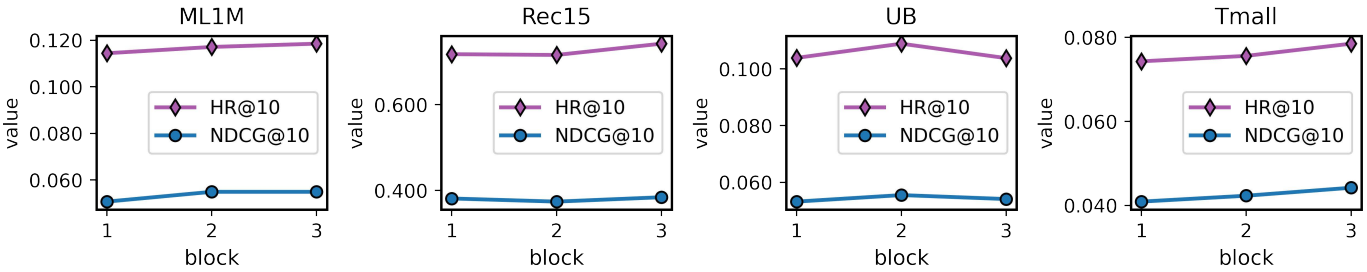

Figure 2: Recommendation performance of our Caser++ with different numbers of filter-enhanced blocks.

4.4 Ablation Studies

Notice that there are some different components in our Caser++, including behavior embedding, position embedding and filter-enhanced blocks. Thus, in this section, we seek to understand how they affect the model via a series of ablation tests (**RQ3**). Specifically, we study their effectiveness by removing some components from the model.

The results of six variants of our Caser++ are shown in Table 3. The value with the largest drop in each column is underlined, the second-to-last value in italics. We can observe that when we remove FEB (filter-enhanced blocks), the performance of the model will be severely degraded, which indicates that it is very helpful to alleviate overfitting by filtering noise in the data. In addition, BE (behavior embedding) and PE (position embedding) also play an important role in improving model performance since we make full use of the information of a sequence. And taking all components into account yields the best model performance in most cases.

5 Conclusion and future work

In this paper, we study an emerging and significant problem called sequential heterogeneous one-class collaborative filtering (SHOCCF). As a response, we design an embedding vector in the latent space to represent the sequence features, which consists of item information, location information and behavior information, and makes full use of the sequential information. We then denoise the data by stacking several learnable filter-enhance blocks[10] to prevent overfitting from fitting it into a more complex structure. Finally, we use a convolutional layer to capture users' dynamic preferences following Caser[9]. We conduct extensive empirical studies on four public datasets, and find that our Caser++ is significantly better than Caser in most cases, and

is also very competitive in the context of the state-of-the-art methods for the studied problem.

For future works, we are interested in further utilizing convolutional operations to learn more fine-grained union-level effects of different behavior combinations on users' subsequent actions.

References

[1] KOREN Y. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model [C/OL]//KDD '08: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2008: 426-434. https://doi.org/10.1145/1401890.1401944. DOI: 10.1145/1401890.1401944.

[2] RENDLE S. Factorization Machines with LibFM[J/OL]. ACM Trans. Intell. Syst. Technol., 2012, 3(3): 57. https://doi.org/10.1145/2168752.2168771. DOI: 10.1145/2168752.2168771.

[3] PAN W, YANG Q, CAI W, et al. Transfer to Rank for Heterogeneous One-Class Collaborative Filtering [J/OL]. ACM Trans. Inf. Syst., 2019, 37(1): 1-20. https://doi.org/10.1145/3243652. DOI: 10.1145/3243652.

[4] PAN W, LIU M, MING Z. Transfer Learning for Heterogeneous One-Class Collaborative Filtering[J]. IEEE Intelligent Systems, 2016, 31(4): 43-49. DOI: 10.1109/MIS.2016.19.

[5] KABBUR S, NING X, KARYPIS G. FISM: Factored Item Similarity Models for Top-N Recommender Systems[C/OL]//KDD '13: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013: 659-667. https://doi.org/10.1145/2487575.2487589. DOI: 10.1145/2487575.2487589.

[6] CHEN C, ZHANG M, ZHANG Y, et al. Efficient Heterogeneous Collaborative Filtering without Negative Sampling for Recommendation[C/OL]//AAAI '20: Proceedings of the 34th AAAI Conference on Artificial Intelligence. 2020: 19-26. https://ojs.aaai.org/index.php/AAAI/article/view/5329. DOI: 10.1609/aaai.v34i01.5329.

[7] LIU Q, WU S, WANG L. Multi-Behavioral Sequential Prediction with Recurrent Log-Bilinear Model [J]. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(6): 1254-1267. DOI: 10.1109/TKDE.2017.2661760.

[8] ZHOU M, DING Z, TANG J, et al. Micro Behaviors: A New Perspective in E-Commerce Recommender Systems[C/OL]//WSDM '18: Proceedings of the 11th ACM International Conference on Web Search and Data Mining. 2018: 727-735. https://doi.org/10.1145/3159652.3159671. DOI: 10.1145/3159652.3159671.

[9] TANG J, WANG K. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding[C/OL]//WSDM '18: Proceedings of the 11th ACM International Conference on Web Search and Data Mining. 2018: 565-573. https://doi.org/10.1145/3159652.3159656. DOI: 10.1145/3159652.3159656.

[10] ZHOU K, YU H, ZHAO W X, et al. Filter-Enhanced MLP is All You Need for Sequential Recommendation[C/OL]//WWW '22: Proceedings of the ACM Web Conference 2022. 2022: 2388-2399. https://doi.org/10.1145/3485447.3512111. DOI: 10.1145/3485447.3512111.

[11] RENDLE S, FREUDENTHALER C, SCHMIDT-THIEME L. Factorizing Personalized Markov Chains for Next-Basket Recommendation[C/OL]//WWW '10: Proceedings of the 19th International Conference on World Wide Web. 2010: 811-820. https://doi.org/10.1145/1772690.1772773. DOI: 10.1145/1772690.1772773.

[12] HE R, MCAULEY J. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation[C]//ICDM '16: Proceedings of the 16th IEEE International Conference on Data Mining. 2016: 191-200.

[13] HE R, KANG W C, MCAULEY J. Translation-Based Recommendation[C/OL]//RecSys '17: Proceedings of the 11th ACM Conference on Recommender Systems. 2017: 161-169. https://doi.org/10.1145/3109859.3109882. DOI: 10.1145/3109859.3109882.

[14] HIDASI B, KARATZOGLOU A. Recurrent Neural Networks with Top-k Gains for Session-Based Recommendations[C/OL]//CIKM '18: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. 2018: 843-852. https://doi.org/10.1145/3269206.3271761. DOI: 10.1145/3269206.3271761.

[15] DONKERS T, LOEPP B, ZIEGLER J. Sequential User-Based Recurrent Neural Network Recommendations[C/OL]//RecSys '17: Proceedings of the 11th ACM Conference on Recommender Systems. 2017: 152-160. https://doi.org/10.1145/3109859.3109877. DOI: 10.1145/3109859.3109877.

[16] HIDASI B, KARATZOGLOU A, BALTRUNAS L, et al. Session-based recommendations with recurrent neural networks[C]//ICLR '16: Proceedings of the 4th International Conference on Learning Representations. 2016.

[17] QUADRANA M, KARATZOGLOU A, HIDASI B, et al. Personalizing Session-Based Recommendations with Hierarchical Recurrent Neural Networks[C/OL]//RecSys '17: Proceedings of the 11th ACM Conference on Recommender Systems. 2017: 130-137. https://doi.org/10.1145/3109859.3109896. DOI: 10.1145/3109859.3109896.

[18] KANG W C, MCAULEY J. Self-Attentive Sequential Recommendation[C]//ICDM '18: Proceedings of the 18th IEEE International Conference on Data Mining. 2018: 197-206. DOI: 10.1109/ICDM.2018.00035.

[19] LIN J, PAN W, MING Z. FISSA: Fusing Item Similarity Models with Self-Attention Networks for Sequential Recommendation[C/OL]//RecSys '20: Proceedings of the 14th ACM Conference on Recommender Systems. 2020: 130-139. https://doi.org/10.1145/3383313.3412247. DOI: 10.1145/3383313.3412247.

[20] WU S, TANG Y, ZHU Y, et al. Session-based recommendation with graph neural networks[C]//AAAI ’19: Proceedings of the 33th AAAI conference on artificial intelligence: vol. 33: 01. 2019: 346-353.

[21] XU C, ZHAO P, LIU Y, et al. Graph Contextualized Self-Attention Network for Session-based Recommendation.[C]//IJCAI: vol. 19. 2019: 3940-3946.

[22] MA C, MA L, ZHANG Y, et al. Memory augmented graph neural networks for sequential recommendation[C]//AAAI ’20: Proceedings of the 34th AAAI conference on artificial intelligence: vol. 34: 04. 2020: 5045-5052.

[23] LI Z, ZHAO H, LIU Q, et al. Learning from History and Present: Next-Item Recommendation via Discriminatively Exploiting User Behaviors[C/OL]//KDD ’18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2018: 1734-1743. https://doi.org/10.1145/3219819.3220014. DOI: 10.1145/3219819.3220014.

[24] ZHAN Z, HE M, PAN W, et al. TransRec++: Translation-based sequential recommendation with heterogeneous feedback.[J]. Frontiers Comput. Sci., 2022, 16(2): 162615.

[25] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting[J/OL]. Journal of Machine Learning Research, 2014, 15(56): 1929-1958. http://jmlr.org/papers/v15/srivastava14a.html.

[26] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[C]//CVPR ’16: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[27] BA J L, KIROS J R, HINTON G E. Layer normalization[J]. CoRR abs/1607.06450, 2016.

[28] PAN W, YANG Q, CAI W, et al. Transfer to Rank for Heterogeneous One-Class Collaborative Filtering [J/OL]. ACM Trans. Inf. Syst., 2019, 37(1): 1-20. https://doi.org/10.1145/3243652. DOI: 10.1145/3243652.

[29] HE M, PAN W, MING Z. BAR: Behavior-Aware Recommendation for Sequential Heterogeneous One-Class Collaborative Filtering[J]. Information Sciences, 2022. DOI: https://doi.org/10.1016/j.ins.2022.06.084.