J Intell Inf Syst (2019) 52:337–365
https://doi.org/10.1007/s10844-018-0514-7

Reliable TF-based recommender system for capturing complex correlations among contexts

Byungkook Oh[1] · Sangjin Shin[1] · Sungkwang Eom[1] · Jooik Jung[1] · Minjae Song[1] · Seungmin Seo[1] · Kyong-Ho Lee[1]

Received: 25 July 2017 / Revised: 4 June 2018 / Accepted: 4 June 2018 /
Published online: 25 June 2018

Abstract Context-aware recommender systems (CARS) exploit multiple contexts to improve user experience in embracing new information and services. Tensor factorization (TF), a type of latent factor model, has achieved remarkable performance in CARS. TF learns latent representations of contexts by decomposing an observed rating tensor and combines the latent representations as a vector form to represent contextual influence on users and items. However, due to the limitation of the contextual expression power, they have difficulties in effectively capturing complex correlations among multiple contexts, and also the meaning of each context is diluted. To address the issue, we propose a reliable TF-based recommender system based on a proposed context tensor (CT-CARS), which incorporates a variety of correlations among contexts. CT-CARS contains a novel recommendation rating function and a learning algorithm. Specifically, the proposed context tensor elaborately captures the influences of both individual contexts and context combinations. Moreover, we

✉ Kyong-Ho Lee
khlee89@cs.yonsei.ac.kr

Byungkook Oh
bkoh@icl.yonsei.ac.kr

Sangjin Shin
sjshin@icl.yonsei.ac.kr

Sungkwang Eom
skeom@icl.yonsei.ac.kr

Jooik Jung
jijung@icl.yonsei.ac.kr

Minjae Song
mjsong@icl.yonsei.ac.kr

Seungmin Seo
smseo@icl.yonsei.ac.kr

[1] Department of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Republic of Korea

introduce a novel parameter initialization based on past-learned results to improve the reliability of recommendations. The overall time complexity of our parameter learning algorithm grows linearly as dataset size increases. Experiments on six real-world datasets including two large-scaled datasets show that CT-CARS outperforms the existing state-of-the-art models in terms of both accuracy and reliability.

Keywords Context-aware · Recommender system · Collaborative filtering · Latent factor model · Matrix factorization · Optimization

1 Introduction

Nowadays, with the increasing volume and variety of information and services, users have difficulties in selecting what they truly desire. To effectively offer what users really want, context-aware recommender systems (CARS) are proposed in various domains such as movie (Harper and Konstan 2015; Bennett and Lanning 2007), place (Zheng and Xie 2011), shopping (Banerjee et al. 2016), tourism (Christensen et al. 2016), and food (Tran et al. 2017). CARS analyzes contextual influences on a user's past activities to accurately predict his or her future activities. The contextual influences are different depending on individual contexts and even on the combinations among multiple contexts, users, and items. Therefore, in order to improve the performance of recommendations, it is crucial to capture complex correlations among contexts and analyze their influences on a user activity.

Matrix factorization (MF) which is a fundamental type of latent factor model has been spotlighted in recommender systems in that it ensures high accuracy and scalability even with a sparse user-item rating matrix. Relationships among users, items, and multiple contexts can be represented as a user-item-contexts rating tensor which has several context dimensions in the user-item rating matrix. Tensor factorization (TF), an extension of MF, decomposes the rating tensor into two latent matrices of users and items and several latent matrices of contexts (i.e., as many as the number of context types). An unknown relationship (i.e., rating by a user on an item under contexts) is predicted by combining the latent representations. Each context type has several context instances, e.g., weather context type has context instances {snow, sunny, rain}.

The conventional TF-based models decompose a tensor into several latent matrices simultaneously in a learning stage and combine them all together in an inference stage. Thus, the models cannot grasp the contextual influences on users and items, respectively. Recently, stepwise approaches integrate multiple contexts, and then combine the integrated context with users and items, respectively. The approach can model both user-contexts relations (contextual influence on users) and item-contexts relations (contextual influence on items). However, since they represent the contextual influence as a single vector, the meaning of individual context instance may be diluted. Furthermore, the single vector is generated by a weighted sum which utilizes a fixed weight value per context type. Thus, the fixed weights limit the complex correlations among contexts. For example, in the process of integrating contexts, two fixed weights are applied to the {sunny, rain, snow} for weather type and {friends, spouse} for companion type, respectively. A user preference is mainly determined by the weather type, but may also be determined by the companion type depending on the user. Moreover, the importance of {spouse} may differ depending on {sunny, rain, snow}.

Table 1 Example of training dataset with Top-1

No.	User ID	Item ID (Top-1)	Context type (weather)			Context type (companion)	
			Snow	Sunny	Rain	Friends	Spouse
1	Jones	Hot dog	1	–	–	1	–
2	Jones	Steak	–	1	–	–	1
3	Davis	Hot dog	1	–	–	–	1
4	Davis	Steak	–	1	–	–	1
5	Davis	Steak	–	–	1	1	–
6	Davis	Sandwich	–	–	1	1	–
7	Smith	Hot dog	1	–	–	–	1
8	Smith	Hot dog	–	–	1	–	1
9	Brown	Sandwich	–	–	1	1	–
10	Brown	Hot dog	1	–	–	1	–
11	Brown	Steak	–	1	–	–	1
...	...	...	...	...	...	...	...

Dominant context instances or combinations play an important role in determining a user activity. This means that multiple contexts have various correlations. For example, Table 1 shows an example of a training dataset with top-1 items for users under weather and companion context types. Previous models assign two fixed weight values to the context types to integrate the contexts using a weighted sum. However, as shown in Table 2, snow and sunny context instances are dominant, because they properly identify top-1 items better than {rain}. Thus, the fixed value for weather type cannot reflect that the influences of the snow and sunny instances are higher than the influence of rain. Moreover, as shown in Table 3, the influence of a context instance varies depending on context combinations. The sunny instance in {sunny, spouse} is more dominant than the snow instance in {snow, friends} and {snow, spouse}. As described in the example, it is clear that the fixed weighted sum cannot elaborately captures various correlations among multiple contexts.

One of the previous models learns a context-dependent matrix per context combination in order to capture all correlations. Since numerous contexts cause the exponential number of context-dependent matrices, this results in a high computational overhead. In a typical learning algorithm, another problem is that learned parameters absolutely depend on initial parameters. Optimal parameters may not be learned from randomly generated initial parameters. Since the performance of recommender systems varies according to the initial parameters, the random initialization leads to the low reliability of recommendations.

Table 2 Count of weather type for Top-1 item (effects of dominant context instances in weather type from Table 1)

Context type (weather)	Item ID (Top-1)	Count (#)
Sunny	Steak	3
Snow	Hot dog	4
Rain	Steak	1
Rain	Sandwich	2
Rain	Hot dog	1

Table 3 Count of {weather, companion} for Top-1 item (effects of dominant context instance or combinations in {weather, companion} combinations from Table 1)

Context type (weather)	Context type (companion)	Item ID (Top-1)	Count (#)
Sunny	Spouse	Steak	3
Snow	Friends	Hot dog	2
Snow	Spouse	Hot dog	2
Rain	Friends	Steak	1
Rain	Friends	Sandwich	2
Rain	Spouse	Hot dog	1

To cope with the problems, this article presents a novel recommender system named CT-CARS, which is based on a proposed tensor factorization. The proposed CT-CARS factorizes a user-item-contexts rating tensor into several latent matrices and the proposed context tensor. In order to grasp the contextual influences of users and items separately, we adopt a stepwise approach. The context tensor is used in the step of integrating contexts. The contributions of CT-CARS are summarized as follows: (i) we formulate a novel recommendation rating function and propose learning algorithm for the rating function by using the context tensor. Also, (ii) we propose a method of initializing parameters from past-learned results for the learning algorithm, and (iii) we manage the past-learned results for parameter initialization. Finally, (iv) we present the overall architecture of CT-CARS. Hereafter, we use the terms LVC and LVE to indicate the latent vectors of context instances and entities, respectively. In addition, PLVC indicates the LVC in the past-learned results.

In the first step of the proposed approach, in order to elaborately integrate LVCs in accordance with a context combination, LVCs are combined with the proposed context tensor. Unlike a fixed weight value which is used in a simple weighted sum, the context tensor is able to capture correlations among multiple contexts for various context combinations. Moreover, the context tensor generates the influence of a context combination in a matrix form, unlike a single vector form in previous works. In order to model a user-contexts relation and an item-contexts relation, at the next step, the contextual influence is combined with a user and an item. Since the contextual influence is represented as a matrix, which contains more information than a single vector, we preserve the meaning of each context as much as possible. The context tensor is fixed as a three-dimensional matrix, irrespective of the number of context types. Therefore, the number of context types has little effect on the overall computational overhead.

The same context instances in different datasets have similar contextual influences on the same recommendation domain. Thus, all LVCs of context instances show similar patterns across multiple datasets. Whenever parameters are learned, the CT-CARS system configures initial parameters based on the PLVCs that showed the best performance. The configuration ensures the initial parameters with the patterns of PLVCs, improving the reliability of recommendations. Hereafter, we define the degree of reflecting PLVCs on a new learning as the term RW (reusing weight). A user's preference commonly decreases as time goes on. RW depends on the elapsed time. Thus, The older PLVC, the lower the RW on a new learning.

The rest of this article is organized as follows: Section 2 reviews related work. Section 3 introduces some preliminaries for CARS. In Section 4, we describe the overall architecture of CT-CARS. Section 5.1 presents a novel recommendation rating function based on the proposed context tensor. Section 5.2 presents a learning algorithm and explains our parameter initialization with PLVCs, and Section 6 shows experimental results to evaluate the performance of the proposed recommender system. Finally, we conclude our work and discuss a few interesting future works in Section 7.

2 Related work

The goal of context-aware recommender systems (CARS) is to identify unobserved interactions between entities (e.g., users and items) under multiple contexts. The interaction is commonly rated as a n star scale, and the set of interactions is represented as a user-item-contexts rating tensor. Collaborative filtering (CF) is a typical CARS method to capture the unobserved interactions between entities under contexts. The existing CF for recommendations can be categorized into neighborhood-based CF, which is also known as memory-based CF (Deshpande and Karypis 2004), model-based CF (Koren et al. 2009), and hybrid CF (Pennock et al. 2000). The model-based CF is generally realized by latent factor models based on MF, which uses machine learning techniques to identify user-item relationships. MF factorizes a rating matrix into two low dimensional latent matrices, which represent potential properties of users and items, respectively. MF is used in various research fields such as recommendation, semantic web, social network, and classification (Liu et al. 2015c).

As a previous work based on MF, collective matrix factorization (CMF) utilizes relations between entities as contexts, and decomposes the relation into two latent matrices (Singh and Gordon 2008). However, CMF has a problem that some context types may be ignored in multiple contexts, and shows low accuracy. To solve the problem, Heterogeneous matrix factorization (HeteroMF) learns a context-dependent matrix per context combination, and computes context-specific latent vectors of entities from the context-specific latent matrix with latent vectors of entities (Jamali and Lakshmanan 2013). All the context types are considered in the learning process without being disregarded, but as the number of context types increases, the number of context combinations increases exponentially. Thus, the high computational complexity is incurred. Probabilistic matrix factorization (PMF) analyzes Gaussian noise while learning latent factors, and reflects the Gaussian noise on regularization (Ma et al. 2008; Salakhutdinov and Mnih 2008). Each domain contains both domain-specific and domain-independent information for entities and context. Man et al. and Zhang et al. utilize the information to alleviate the data sparsity (Man et al. 2015; Zhang et al. 2016).

Factorization machine (FM) (Rendle et al. 2011; Nguyen et al. 2014) and Multiverse Recommendation (Karatzoglou et al. 2010) decompose a user-item-contexts rating tensor into several latent matrices at once. These approaches cannot capture the user-contexts relation and the item-contexts relation separately. In order to capture the respective relations, recent approaches adopt a stepwise approach based on tensor multiplication and represent a context instance as a LVC. Contextual operating tensor (COT) (Liu et al. 2015a) and CARS2 (Shi et al. 2014) learn two tensors, which represent the contextual influence on users and items. Liu et al. use a sigmoid function to learn LVCs, and combines the LVCs with users' and items' LVEs, respectively (Liu and Wu 2015). Chen et al. analyze tags and ratings among entities to mine a user's implicit preference based on topic modeling (Chen et al. 2016). Hierarchical interaction representation model (HIR) analyzes the interaction between two entities and combines the interactions of multiple entities repeatedly (Liu et al. 2015b). However, the conventional models disregard the sophisticated influence of context instances due to a simple summation with fixed weight values. Thus, they cannot elaborately capture complex correlations among multiple contexts.

Bayesian probabilistic tensor factorization (BPTF) factorizes a rating tensor via Bayesian interpretation (Xiong et al. 2010). BPTF analyzes the frequency and tendency of users' activities in terms of the probability concept. BPTF effectively learns the LVCs and LVEs with a Markov chain Monte Carlo method, and shows high scalability and low computational overhead. Some works apply TF to semantic data such as linked open data (Nickel et al. 2011, 2012; Drumond et al. 2012). Linked data is represented in the form of subject-property-object (SPO) triples. Thus, the data is converted into a third-order tensor, which has a subject, a property, and an object axes. Unknown link is learned and predicted by decomposing the tensor. As discussed above, latent factor models mostly show better performance than the others of CARS. However, it is not easy to explain the recommendation process and to interpret each latent factor by using real-world examples. Contextual sparse linear model (CSLIM) easily explains the recommendation process in view of the real world (Zheng et al. 2014). The approach mines the contextual influence from the deviations of a user's ratings in different context combinations.

In general, multiple contexts improve the performance of recommendations in CARS. However, the excessive number of contexts often causes poor performance. Thus, several models are proposed to exploit contextual information among multiple contexts efficiently. If the context combinations are similar, the recommended lists are also similar. Similarity-learning model (SLM) learns the context similarity along with optimizing the parameters of a recommendation rating function (Zheng et al. 2015a). Differential context relaxation (DCR) model utilizes additional contextual information, which is generated from analyzing common features among multiple users (Zheng et al. 2012a). Moreover, the influence of a specific context on recommendations may be different depending on context types. Zheng et al. propose differential context weighting (DCW) model based on DCR to analyze the weights of contextual features (Zheng et al. 2013). Time-dependent contextual information may have a limitation that a user's tendency changes over time. In order to reflect the changes, some approaches apply different weights over time in the same context based on the forgetting pattern of human beings (Niederee et al. 2015; Chen et al. 2010). However, the above models do not offer a universal framework available in various domains.

3 Preliminaries

To begin, Table 4 summarizes the definition of main symbols used in the recommendation rating function and learning algorithm of CT-CARS. We also explain some background of typical TF-based recommender systems. Boldface upper-case letters $\mathbf{V}$, $\mathbf{M}$, $\mathbf{C}$, $\mathbf{T}$ represent matrices or tensors, and its lower-case italic letters v, c represent vectors. We also define several mathematical expressions. Let $\mathbf{X}$ be a 2D matrix, $\mathbf{X}^{[:,:]}$ denotes the matrix $\mathbf{X}$. $\mathbf{X}^{[a,:]}$ and $\mathbf{X}^{[:,b]}$ denote a-th row and b-th column of the matrix $\mathbf{X}$, respectively. $\mathbf{X}^{[a,b]}$ denotes an element. This expression can also be applied to a tensor as shown in Fig. 1. In addition, $vertCat(\bigcup_{i=1}^{m} x_i)$ vertically concatenates a sequence of vectors or elements $\{x_1, \ldots, x_m\}$. Likewise, $horzCat(\bigcup_{i=1}^{m} x_i)$ denotes a horizontal concatenation. For example, $m \times n$ matrix $\mathbf{X}$ can be represented as $vertCat(\bigcup_{i=1}^{m} x^{[i,:]})$, where $x^{[i,:]}$ indicates i-th row vector of $\mathbf{X}$. $horzCat(\bigcup_{i=1}^{n} x^{[:,i]})$ also represents the matrix $\mathbf{X}$.

A set of relationships among n_u users, n_v item, and n_c contexts is represented as a user-item-contexts rating tensor $\mathbf{R} \in \mathbb{R}^{n_u \times n_v \times (1 \times \ldots \times n_c)}$. In TF, the latent representation of user i and item j are represented as vectors $\mathbf{V}_u^{[:,i]} \in \mathbb{R}^{d_{uv}}$ and $\mathbf{V}_v^{[:,j]} \in \mathbb{R}^{d_{uv}}$, which are selected from latent matrices of users $\mathbf{V}_u \in \mathbb{R}^{d_{uv} \times n_u}$ and items $\mathbf{V}_v \in \mathbb{R}^{d_{uv} \times n_v}$. The context instances are also represented as d_c-dimensional vectors. A specific combination of context instances

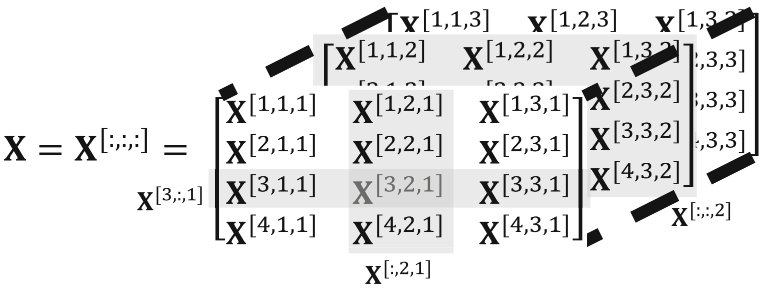
Table 4 Definition of main symbols

Symbols	Definition
LVC	Latent vector of a context instance
LVE	Latent vector of a user or an item
PLVC	LVC in past-learned results
RW	Degree of reflecting PLVC on a new learning
k	Context combination
d_c	The number of latent factors of LVC
d_{uv}	The number of latent factors of LVE
n_c	The number of context types
n_u	The number of users
n_v	The number of items
$\mathbf{V}_u, \mathbf{V}_v$	Matrix representation of $\mathbf{V}_u^{[:,i]}, \mathbf{V}_v^{[:,j]}$
$\mathbf{V}_u^{[:,i]}, \mathbf{V}_v^{[:,j]}$	Latent vector of user i and item j
$\mathbf{V}_{u,k}^{[:,i]}, \mathbf{V}_{v,k}^{[:,j]}$	Latent vector of user i and item j under k (context-specific LVE of user i and item j)
$\mathbf{C}_k$	Matrix representation of k
$\mathbf{T}_c$	Correlations among contexts
$\mathbf{M}_u, \mathbf{M}_v$	Common characteristics of users and items
$\mathbf{M}_{c,k}$	Contextual influence of k
$\mathbf{M}_{u,k}, \mathbf{M}_{v,k}$	Contextual influences of k on users and items

is named context combination k, which is constructed by n_c context instances (one for each context type). The relationship of a rating tensor $\mathbf{R}$ indicates an observed rating of user i for item j under context combination k, denoted as $r_{i,j,k}$. $\mathbf{R}$ consists of real-valued ratings $r_{i,j,k}$, where the value 0 indicates an unobserved rating. Since most ratings are left empty, $\mathbf{R}$ is mostly a sparse tensor.

3.1 Tensor factorization

We briefly review a traditional TF approach for collaborative filtering. To factorize a rating tensor $\mathbf{R}$ having c_{n_c} context types, the approach assigns a d-dimensional latent vector to every user, item, and context instance, denoted as u_i, v_j, and $c_1, \ldots, c_{n_c}$, on a shared



$\mathbf{X} = \mathbf{X}^{[:,:,:]} =$
$\mathbf{X}^{[1,1,1]}$ $\mathbf{X}^{[1,2,1]}$ $\mathbf{X}^{[1,3,1]}$
$\mathbf{X}^{[2,1,1]}$ $\mathbf{X}^{[2,2,1]}$ $\mathbf{X}^{[2,3,1]}$
$\mathbf{X}^{[3,1,1]}$ $\mathbf{X}^{[3,2,1]}$ $\mathbf{X}^{[3,3,1]}$
$\mathbf{X}^{[4,1,1]}$ $\mathbf{X}^{[4,2,1]}$ $\mathbf{X}^{[4,3,1]}$
$\mathbf{X}^{[1,1,2]}$ $\mathbf{X}^{[1,2,2]}$
$\mathbf{X}^{[2,3,2]}$ $\mathbf{X}^{[3,3,2]}$ $\mathbf{X}^{[4,3,2]}$
$\mathbf{X}^{[1,1,3]}$ $\mathbf{X}^{[1,2,3]}$
$\mathbf{X}^{[3,:,1]}$
$\mathbf{X}^{[:,2,1]}$
$\mathbf{X}^{[:,:,2]}$

Fig. 1 Mathematical expressions in this article

latent vector space. The tensor product of the latent vectors of user i, item j and context combination k should be approximated to $r_{i,j,k}$ in $\mathbf{R}$ as follows:

$$\mathbf{R} = \bigcup_{i,j,k} (r_{i,j,k} \approx \hat{r}_{i,j,k} = u_i \otimes v_j \otimes c_1 \otimes \cdots c_{n_c}), \tag{1}$$

where $\hat{r}_{i,j,k}$ denotes a recommendation rating, and $\otimes$ denotes a tensor product. u_i and v_j (i.e., LVEs) and $c_1, \ldots, c_{n_c}$ (i.e., LVCs) are simultaneously combined all together (i.e., $u_i \otimes v_j \otimes c_1 \otimes \cdots c_{n_c}$). Thus, the approach cannot capture the user-contexts and item-contexts relationships, respectively. Recently, a stepwise TF approach requires the different dimensions of LVC and LVE. The approach integrates n_c context instances for a context combination k. Then, the integrated context information is combined with a user and an item, separately:

$$r_{i,j,k} \approx \hat{r}_{i,j,k} = \left(u_i \otimes (c_1 \otimes \cdots c_{n_c})\right) \otimes \left(v_j \otimes (c_1 \otimes \cdots c_{n_c})\right), \tag{2}$$

where $u_i \otimes (c_1 \otimes \cdots c_{n_c})$ and $v_j \otimes (c_1 \otimes \cdots c_{n_c})$ indicate a user-contexts and item-contexts relationships, respectively. The user-contexts relationship is represented as a latent vector of a user i under a context combination k, which is called a context-specific latent vector of s user i (i.e., context-specific LVE of a user i). The item-contexts relationship is also represented as a context-specific latent vector of an item j (i.e., context-specific LVE of an item j). Then, the rating $\hat{r}_{i,j,k}$ can be calculated by an inner product of the two context-specific latent vectors.

3.2 Recommendation rating

A recommendation rating $\hat{r}_{i,j,k}$ is a prediction score which is calculated by combining the factorized latent vectors of a user, an item, and contexts. Collaborative filtering automatically provides items based on a user preference determined by collecting preferences from many other users' ratings. However, each user preference has a tendency which is biased to certain items. Since the tendency varies from individual to individual, some users give higher ratings than the others. In order to predict a rating without the tendency, bias terms for each user, item, and context instance adjust the rating. Thus, the typical recommendation rating with bias terms is written as follows:

$$\hat{r}_{i,j,k} = b_g + b_i + b_j + b_k + \vec{q}_{i,k} \cdot \vec{p}_{j,k}, \tag{3}$$

where $\vec{q}_{i,k}$ and $\vec{p}_{j,k}$ are context-specific LVEs of a user i and an item j, respectively. Here, b_g indicates a global bias, b_i and b_j correspond to entity biases of a user i and an item j, and b_k denotes a bias of a context combination k.

4 Proposed methodology

Figure 2 shows the overall architecture of CT-CARS, which consists of *data storage* and the three main models of *initialization*, *context*, and *learning*. The *data storage* collects history data from a variety of user activities, such as application usage and point of interest (POI) visits. When a user performs an activity, contextual information such as location and time is recorded. The frequency of the same activity is converted into a rating value. The user, user activity, contexts, and user's rating are used as a training data. The *context model* manages two datasets to be utilized in the *initialization model*. One dataset, past-learned results (Ω), is used to check the reusability of PLVCs and to initialize parameters with PLVCs. Each

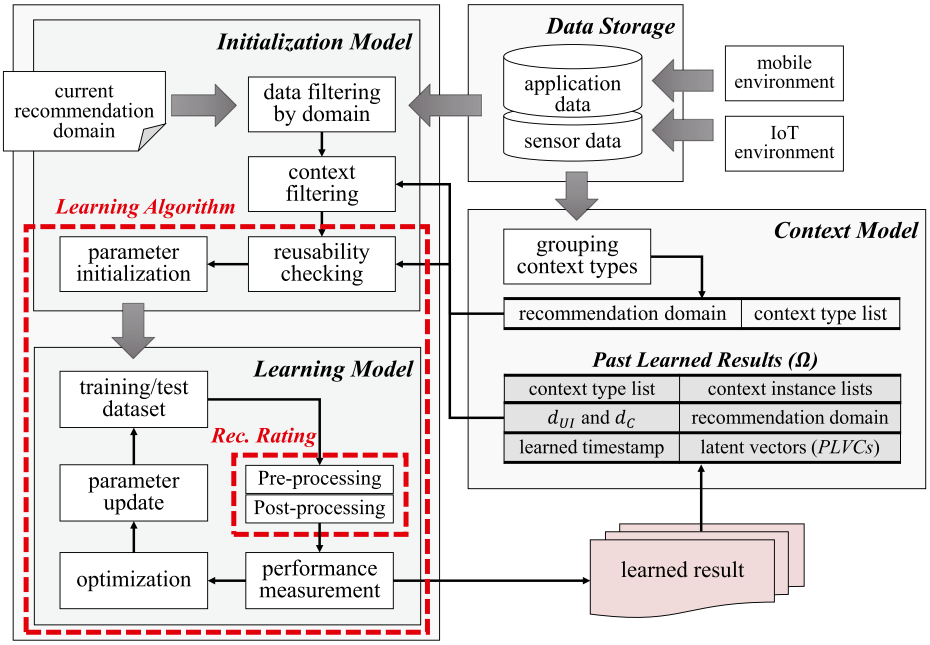


Fig. 2 Overall architecture of CT-CARS

tuple in Ω is generated at every learning, which is performed in the *learning model*. The tuple has PLVCs, a timestamp of learning, a recommendation domain, the number of latent factors of PLVC, and lists of context types and instances. The other dataset, not covered in this article, filters out uninfluential context types.

The proposed recommendation rating function predicts a user's rating. Our learning algorithm finds the optimal parameters of the recommendation rating function. The *learning model* conducts the learning processes repetitively, which include performance measurement, optimization, and parameter update. The amount of parameter update is determined with the performance. To configure a training dataset, the *initialization model* selects data from the *data storage* according to the current recommendation domain. In this model, PLVCs in the *context model* are used to initialize parameters under several reusable conditions. This improves the reliability of the learning. Specifically, to apply a user's tendency over time, the *initialization model* reduces the utilization of PLVCs over time. The *initialization model* will be described in detail in Section 5.2.1.

5 CT-CARS: a detail view

5.1 Recommendation rating function

Figure 3 illustrates the overall process of calculating a recommendation rating in the *learning model*. The proposed recommendation rating function consists of two major steps: pre-processing and post-processing steps.

– **Pre-processing step** captures a multi-context relationship based on the proposed context tensor $\mathbf{T}_c$ which can effectively integrate LVCs with complex correlations among

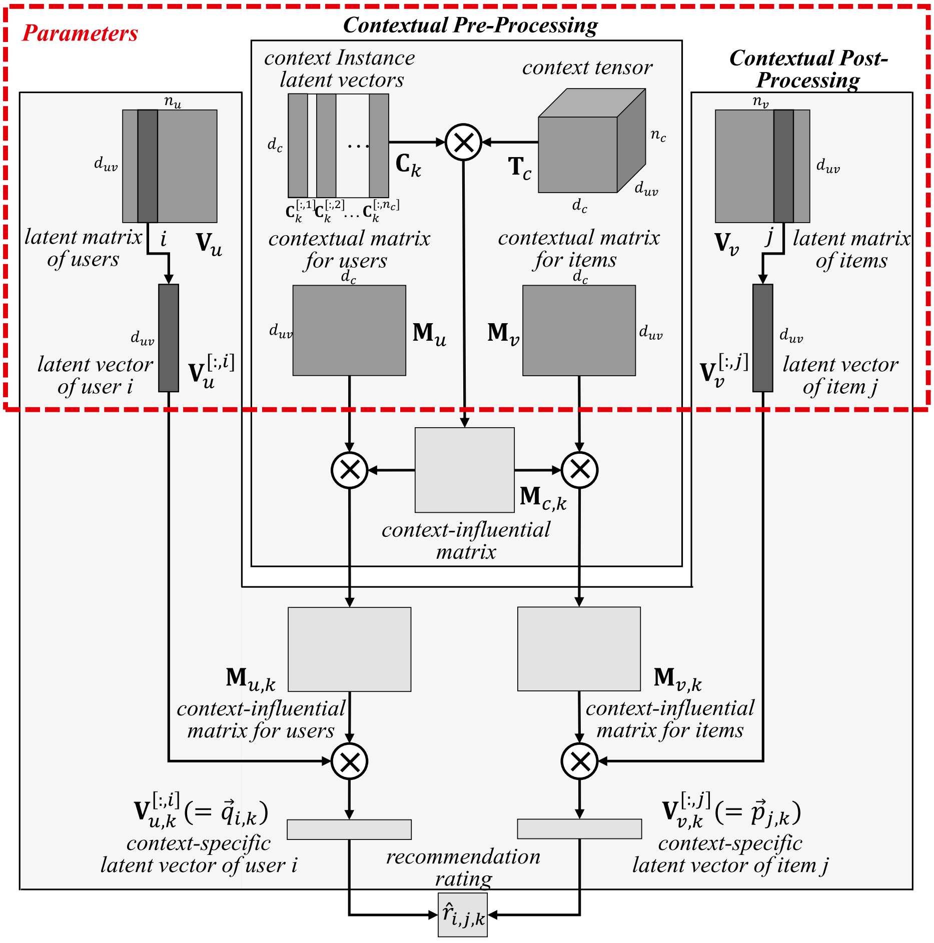

Parameters
Contextual Pre-Processing
Contextual Post-Processing
context Instance latent vectors
context tensor
$\mathbf{C}_k$
$\mathbf{T}_c$
$\mathbf{c}_k^{[:,1]}\mathbf{c}_k^{[:,2]}\dots\mathbf{c}_k^{[:,n_c]}$
latent matrix of users
$\mathbf{V}_u$
latent matrix of items
$\mathbf{V}_v$
contextual matrix for users
$\mathbf{M}_u$
contextual matrix for items
$\mathbf{M}_v$
latent vector of user i
$\mathbf{V}_u^{[:,i]}$
latent vector of item j
$\mathbf{V}_v^{[:,j]}$
$\mathbf{M}_{c,k}$
context-influential matrix
$\mathbf{M}_{u,k}$
context-influential matrix for users
$\mathbf{M}_{v,k}$
context-influential matrix for items
$\mathbf{V}_{u,k}^{[:,i]} (= \vec{q}_{i,k})$
context-specific latent vector of user i
$\mathbf{V}_{v,k}^{[:,j]} (= \vec{p}_{j,k})$
context-specific latent vector of item j
recommendation rating
$\hat{r}_{i,j,k}$

Fig. 3 Recommendation rating process for CT-CARS

LVCs and preserve the meaning of each context instance. Then, the contextual influence of a context combination k is generaated as a matrix $\mathbf{M}_{c,k}$.

– **Post-processing step** aims to capture user-contexts and item-contexts relationships. This step applies the contextual influence $\mathbf{M}_{c,k}$ to the LVEs of a user and an item, and generates the context-specific LVEs of a user and an item. As a result, the recommendation rating is calculated by an inner product between the two context-specific LVEs.

5.1.1 *Pre-processing step*

This step captures a multi-context relationship. In order to effectively integrate contexts, we use the proposed context tensor. The result is a contextual influence for both users and items. Each context type consists of multiple context instances. A specific combination of context instances is named context combination k, which is constructed by n_c context instances (one for each context type). The set-builder notation of k can be explicitly described as:

$$k = \{(x_1, \ldots, x_{n_c}) |\ x_i \in i^{th} context\ type \text{ for } i = 1, \ldots, n_c\}. \tag{4}$$

All context instances in k are replaced by LVCs which are column vectors. Then, the LVCs are vertically concatenated. Each LVC is a d_c-dimensional vector with d_c latent factors. The context combination k can be represented by a matrix representation $\mathbf{C}_k$ as follows:

$$\mathbf{C}_k = vertCat\left(\bigcup_{i=1}^{n_c} LVC_i \in k\right), \tag{5}$$

where $\mathbf{C}_k$ consists of LVCs and forms a $d_c \times n_c$ matrix.

The initial LVCs in the conventional method are only randomly generated. In CT-CARS, if several reusable conditions are satisfied, the LVCs are replaced with PLVCs as described in Section 5.2.1. Since the LVCs in $\mathbf{C}_k$ are simply concatenated together, there is no relationship among LVCs. To capture the relationship, previous works set a fixed value (i.e., weight) for each context type and do a weighted sum to integrate $\mathbf{C}_k$. However, the simple weighted sum cannot capture various correlations among the LVCs. Instead of the weighted sum, CT-CARS elaborately integrate $\mathbf{C}_k$ with the correlations among the LVCs based on the proposed context tensor. The context tensor is combined with $\mathbf{C}_k$ as follows:

$$\mathbf{M}_{c,k} = \begin{bmatrix} horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]} \mathbf{T}_c^{[:,m,1]} \\ \vdots \\ horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]} \mathbf{T}_c^{[:,m,d_{uv}]} \end{bmatrix}, \tag{6}$$

which can be rewritten as follows:

$$\mathbf{M}_{c,k} = vertCat\left(\bigcup_{n=1}^{d_{uv}} horzCat\left(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]} \mathbf{T}_c^{[:,m,n]}\right)\right), \tag{7}$$

where $\mathbf{M}_{c,k}$ forms a $d_{uv} \times d_c$ matrix called a context-influential matrix, denoting the contextual influence of context combination k. The context tensor $\mathbf{T}_c$ forms a $n_c \times d_{uv} \times d_c$ matrix and contains various correlations among LVCs. As discussed above, previous works model a contextual influence as a single vector. Thus, each context may not be reflected well in a single vector. As shown in (6) and (7), CT-CARS models the context influence as the matrix $\mathbf{M}_{c,k}$ rather than a vector. This effectively preserves the meaning of each context instance and can increase the contextual expression power by using a matrix.

5.1.2 Post-processing step

This step captures user-contexts and item-contexts relationships and combines them. Finally a predicted user rating of an item under a specific context combination (i.e., a recommendation rating) is calculated. LVEs of a user and an item should be changed under a context combination k. To do that, the contextual influence of k obtained from the pre-processing step is combined with the LVEs. However, since users and items have different characteristics, the contextual influences of k on users and items must be identified, respectively. Thus, we do not directly combine the contextual influence of k with LVEs. Note that the contextual influence of k is different with the contextual influences of k on users and items.

In order to model the contextual influences of k on users and items from the context-influential matrix $\mathbf{M}_{c,k}$ (i.e., contextual influence of k), we utilize two additional matrices $\mathbf{M}_u$ and $\mathbf{M}_v$ which indicate common characteristics of users and items, respectively. Then, we compute matrix multiplications between the contextual influence of k and the two additional matrices as follows:

$$\mathbf{M}_{u,k} = \mathbf{M}_{c,k}\mathbf{M}_u, \qquad \mathbf{M}_{v,k} = \mathbf{M}_{c,k}\mathbf{M}_v, \tag{8}$$

where $\mathbf{M}_u$ and $\mathbf{M}_v$ form $d_c \times d_{uv}$ matrices, called a contextual matrix for users and a contextual matrix for items. $\mathbf{M}_u$ and $\mathbf{M}_v$ indicate the common characteristics of users and items, respectively. $\mathbf{M}_{u,k}$ and $\mathbf{M}_{v,k}$ form $d_{uv} \times d_{uv}$ matrices, called context-influential matrices for users and items. $\mathbf{M}_{u,k}$ and $\mathbf{M}_{v,k}$ indicate the contextual influences of k on users and items. As discussed above, $\mathbf{M}_{u,k}$ and $\mathbf{M}_{v,k}$ are combined with LVEs of a user and an item, respectively, and generate LVEs of a user and an item under k as follows:

$$\mathbf{q}_{i,k} = \mathbf{V}_{u,k}^{[:,i]} = \mathbf{M}_{u,k}\mathbf{V}_u^{[:,i]}, \qquad \mathbf{p}_{j,k} = \mathbf{V}_{v,k}^{[:,j]} = \mathbf{M}_{v,k}\mathbf{V}_v^{[:,j]}, \tag{9}$$

where $\mathbf{V}_u^{[:,i]} \in \mathbb{R}^{d\times 1}$ and $\mathbf{V}_v^{[:,j]} \in \mathbb{R}^{d\times 1}$ indicate LVEs of a user and an item, which are selected from latent matrices of users $\mathbf{V}_u \in \mathbb{R}^{d\times n_u}$ and items $\mathbf{V}_v \in \mathbb{R}^{d\times n_v}$. Note that $\{\mathbf{C}_k, \mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v, \mathbf{V}_u, \mathbf{V}_v\}$ are parameters which we need to learn, while $\{\mathbf{M}_{c,k}, \mathbf{M}_{c,k}, \mathbf{M}_{v,k}\}$ are intermediate variables. The above equations can be rewritten by replacing with the parameters without the intermediate variables as follows:

$$\mathbf{q}_{i,k} = \mathbf{V}_{u,k}^{[:,i]} = \begin{bmatrix} horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]}\mathbf{T}_c^{[:,m,1]})\mathbf{M}_u\mathbf{V}_u^{[:,i]} \\ \vdots \\ horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]}\mathbf{T}_c^{[:,m,d_{uv}]}\mathbf{M}_u\mathbf{V}_u^{[:,i]} \end{bmatrix}, \tag{10}$$

$$\mathbf{p}_{j,k} = \mathbf{V}_{v,k}^{[:,j]} = \begin{bmatrix} horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]}\mathbf{T}_c^{[:,m,1]})\mathbf{M}_v\mathbf{V}_v^{[:,j]} \\ \vdots \\ horzCat(\bigcup_{m=1}^{d_c} \mathbf{C}_k^{[m,:]}\mathbf{T}_c^{[:,m,d_{uv}]}\mathbf{M}_v\mathbf{V}_v^{[:,j]} \end{bmatrix}, \tag{11}$$

where $\mathbf{V}_{u,k}^{[:,i]}$ and $\mathbf{V}_{v,k}^{[:,j]}$ form d_{uv}-dimensional vectors, called context-specific LVEs of a user i and an item j. $\mathbf{V}_u^{[:,i]}$ indicates a LVE of a user i, and $\mathbf{V}_v^{[:,j]}$ indicates a LVE of an item j. Finally, the recommendation rating $\hat{r}_{i,j,k}$ is calculated by an inner product between $\mathbf{V}_{u,k}^{[:,i]}$ and $\mathbf{V}_{v,k}^{[:,j]}$. The typical recommendation rating function with biases in (3) is rewritten by replacing with (4)–(11) in the pre-processing and post-processing steps as follows:

$$\hat{r}_{i,j,k} = b_g + b_i + b_j + b_k + \overbrace{\underbrace{\mathbf{M}_{c,k}\mathbf{M}_u}_{\mathbf{M}_{u,k}}\mathbf{V}_u^{[:,i]}}^{\mathbf{q}_{i,k}=\mathbf{V}_{u,k}^{[:,i]}} \cdot \overbrace{\underbrace{\mathbf{M}_{c,k}\mathbf{M}_v}_{\mathbf{M}_{v,k}}\mathbf{V}_v^{[:,j]}}^{\mathbf{p}_{j,k}=\mathbf{V}_{v,k}^{[:,j]}}, \tag{12}$$

5.1.3 Analysis of two steps

The LVC in $\mathbf{C}_k$ has latent factors $\{f_{c,1}, \ldots, f_{c,d_c}\}$, and the LVE in $\mathbf{V}_u$ and $\mathbf{V}_v$ have latent factors $\{f_{u,1}, \ldots, f_{u,d_{uv}}\}$ and $\{f_{v,1}, \ldots, f_{v,d_{uv}}\}$, respectively. As shown in Fig. 4, $\{\mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v\}$ also have latent segments according to their matrix slices that combine with the latent factors of LVC or LVE. Each step has an *internal action* which one latent segment or factor does not combine with the others in different position. In the pre-processing step, each latent factor (i.e., $f_{c,1}$, $f_{c,2}$, $f_{c,3}$, $f_{c,4}$) of $\mathbf{C}_k$ combines with the same latent segment

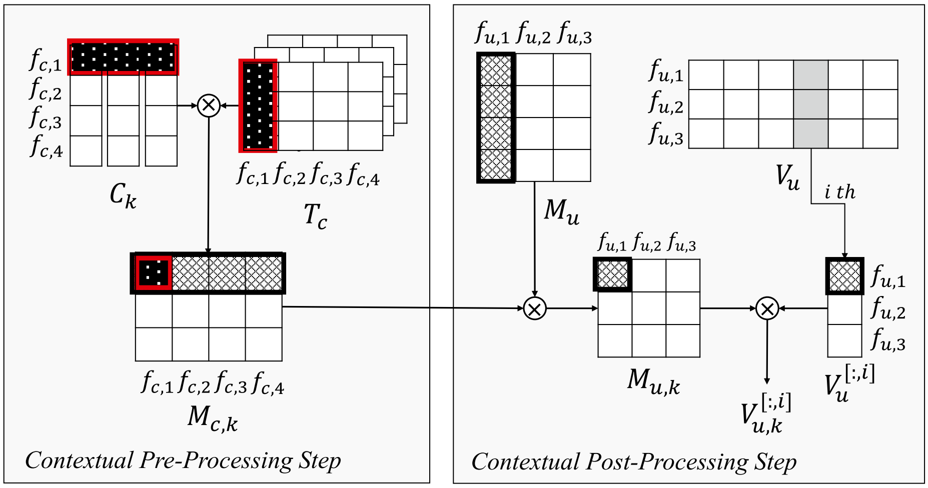

$f_{c,1}$
$f_{c,2}$
$f_{c,3}$
$f_{c,4}$
C_k
$f_{c,1} f_{c,2} f_{c,3} f_{c,4}$
T_c
$f_{c,1} f_{c,2} f_{c,3} f_{c,4}$
$M_{c,k}$
Contextual Pre-Processing Step
$f_{u,1} f_{u,2} f_{u,3}$
M_u
$f_{u,1}$
$f_{u,2}$
$f_{u,3}$
V_u
i th
$f_{u,1} f_{u,2} f_{u,3}$
$M_{u,k}$
$V_{u,k}^{[:,i]}$
$V_u^{[:,i]}$
Contextual Post-Processing Step

Fig. 4 Latent factors and segments for user i in two steps

of $\mathbf{T}_c$, and generates the same latent segment of $\mathbf{M}_{c,k}$. Likewise, $\mathbf{V}_u$ in the post-processing step undergoes the same process. This reflects the semantically independent combination among the latent factors of LVC and LVE in real world. For example, the latent factors (e.g., action, horror, romance) are independent of each other. In other words, the meaning of each latent factor does not interfere with each other. The whole process, including both the pre-processing and post-processing steps, has an *external action*. all latent factors (i.e., $f_{c,1}$, $f_{c,2}$, $f_{c,3}$, $f_{c,4}$) of $\mathbf{C}_k$ in the pre-processing step are combined with one latent factor and segment in the post-processing step. This means that all contextual information is properly reflected in each latent factor of a user and an item.

5.2 Learning algorithm

In this section, we present our learning algorithm. We first introduce a parameter initialization under several reusable conditions by using past-learned LVCs (i.e., PLVCs). Then, we present derivatives for each parameter to find the optimized parameters. To do effectively that, we employ optimization method. We also discuss the complexity of updating parameters in our learning algorithm.

5.2.1 Parameter initialization

Optimized parameters are dependent on the initial parameters. In order to improve the reliability of recommender systems, an appropriate parameter initialization for a training dataset is necessary. To cope with the issue, a transfer learning has been spotlighted in many research fields including natural language processing. For example, the pre-trained vectors from a word2vec model (Mikolov et al. 2013) can be used as input in a new learning, and in some cases, may yield better performance.

Generally, each LVE and LVC in learned parameters tend to have a certain pattern (i.e., direction, magnitude) depending on a training dataset. Among LVE and LVC, the influences of LVC are similar despite the different datasets. Thus, LVCs in learned parameters have a similar pattern, and are possible to reuse across real-world datasets in same recommendation domain. However, several conditions must be met in order to reuse the learned parameters (i.e., transfer learning) to a recommendation system. We check whether several reusable conditions are the same between the current training dataset and the past-learned result which we will reuse, such as: (1) context instance lists of all context types, (2) recommendation domain, and (3) d_c and d_{uv}. The recommendation domain means the same types of items, such as movie, place, or shopping, as described in Section 1. If all reusable conditions are satisfied, the parameters are initialized based on PLVCs, which showed the best performance. Then, a new learning process is conducted. Ω_τ is multiple past-learned results in Ω, which satisfies all reusable conditions depending on the current training dataset τ. Each past-learned result has 6 elements as shown in Fig. 2. PLVCs have a time-dependent property. In a new learning, the influence (i.e., RW) of old PLVCs should be diminished. Also, the influence of frequently learned data should be increased. We calculate the PLVC to be reused as $\text{PLVC}_{reuse,\tau} = \text{RW}_\tau \times \text{PLVC}_{last,\tau}$, where $\text{PLVC}_{last,\tau}$ is latest PLVC in Ω_τ. RW is determined by two factors. One is the elapsed time between the present and the latest past learning. The other is the frequency of learnings in Ω_τ. RW_τ is calculated as follows:

$$RW_\tau = \frac{1}{1 + e^{\frac{lap_{last,\tau} - 4\alpha}{\alpha}}}, \tag{13}$$

$$lap_{last,\tau} = lap_{last,\tau} + \sqrt{\frac{\sum_{i\in\Omega_\tau}(\sqrt{lap_{i,\tau}} - \sqrt{lap_{last,\tau}})^2}{|\Omega_\tau|}}, \tag{14}$$

where the elapsed time $lap_{last,\tau}$ determines RW_τ based on a sigmoid function. α controls the shape of (13) and indicates the degree of reduction for RW_τ over time. $lap_{i,\tau}$ denotes the elapsed time between the present and i-th past learning in Ω_τ. $lap_{last,\tau}$ corresponds to the latest past learning among $lap_{i,\tau}$. In order to apply the frequency of learnings to RW_τ, (14) adjusts $lap_{last,\tau}$ with standard deviation among square root of the elapsed times. As shown in Fig. 5, RW_τ converges to 0 at that the elapsed time is 8α or more. Especially, even though the latest past learnings of $\Omega_{\tau1}$ and $\Omega_{\tau2}$ are the same, $\text{RW}_{\tau2}$ for $\Omega_{\tau2}$ (frequently learned) is bigger than $\text{RW}_{\tau1}$.

5.2.2 *Derivatives for parameters*

As discussed in Fig. 2, the *learning model* receives a set of initial parameters, $\Theta = \{\mathbf{C}_k, \mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v, \mathbf{V}_u, \mathbf{V}_v\}$, which are randomly generated or reused. Then, the initial parameters are learned with our learning algorithm to accurately predict user's ratings. In

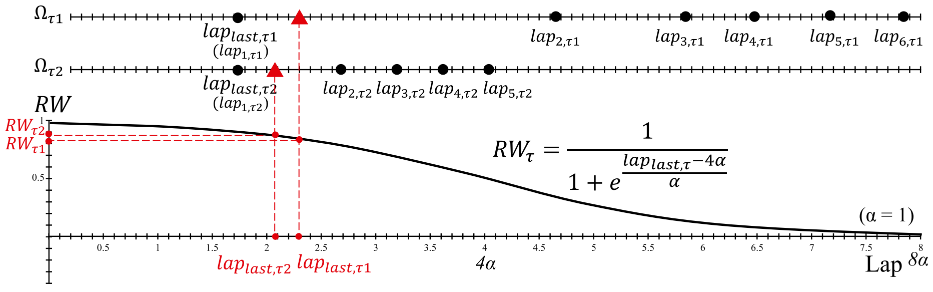


Fig. 5 RW for parameter initialization

order to obtain the optimal parameters, (i) the *learning model* first predicts the recommendation ratings with the initial parameters by (12), and (ii) finds the differences between observed ratings in a training dataset and predicted ratings. The difference is called an error. (iii) The performance of the parameters is measured with a loss function. In our learning algorithm, we formulate the loss function as the sum of the squared errors for all tuples in a training dataset. (iv) Then, the parameters are updated, in the direction of reducing the cost of a loss function.

The above performance measurement and parameter update processes (i–iv) are repeated until the cost of the loss function is converged. As a result, the set of parameters is learned as an optimal solution where the loss function reaches its minimum. To avoid the excessive learning causing the generality problem (i.e., overfitting), we adopt the L_2 regularization. The loss function $\mathscr{L}(\Theta)$ with regularization is as follows:

$$\underset{(\mathbf{C}_k,\mathbf{T}_c,\mathbf{M}_u,\mathbf{M}_v,\mathbf{V}_u,\mathbf{V}_v)\in\Theta}{\arg\min} \mathscr{L}(\Theta) = \frac{1}{2}\sum_{(i,j,k)\in\tau} I_{ijk}(r_{i,j,k} - \hat{r}_{i,j,k})^2 + \lambda\mathscr{R}(\Theta), \tag{15}$$

$$\mathscr{R}(\Theta) = \|\mathbf{C}_k\|^2 + \|\mathbf{T}_c\|^2 + \|\mathbf{M}_u\|^2 + \|\mathbf{M}_v\|^2 + \|\mathbf{V}_u\|^2 + \|\mathbf{V}_v\|^2, \tag{16}$$

where I_{ijk} is an indicator function such that it is 1 if user i rated item j under contexts k and 0 otherwise. $r_{i,j,k}$ is an observed rating in training dataset τ, and $\hat{r}_{i,j,k}$ is a recommendation rating which is a predicted rating calculated by (12). λ denotes an influence of regularization, and *arg min* denotes a finding optimal parameters for $\mathscr{L}(\Theta)$. To find the optimal parameters, initial parameters are updated through derivatives of $\mathscr{L}(\Theta)$ for each parameter as follows:

$$\theta_i := \theta_i - \gamma\frac{\partial}{\partial\theta_i}\mathscr{L}(\Theta) \tag{17}$$

where θ_i is a parameter in Θ, γ denotes a learning rate, and all parameters are simultaneously updated. The amount of parameter update, called a gradient, is calculated with their derivatives for all tuples in a training dataset. In this gradient descent method, (17) is repeated until the optimal parameters are found. The derivatives for each parameter $\theta \in \{\mathbf{C}_k, \mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v, \mathbf{V}_u, \mathbf{V}_v\}$ in our learning algorithm are as follows:

$$\frac{\partial\mathscr{L}}{\partial\theta} = \begin{cases} -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\left(\mathbf{M}_{u,k}\right)^T \mathbf{V}_{v,k}^{[:,j]} + \lambda\mathbf{V}_u^{[:,i]} & (\theta = \mathbf{V}_u^{[:,i]}), \\ -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\left(\mathbf{M}_{v,k}\right)^T \mathbf{V}_{u,k}^{[:,i]} + \lambda\mathbf{V}_v^{[:,j]} & (\theta = \mathbf{V}_v^{[:,j]}), \\ -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\left(\mathbf{M}_{c,k}\right)^T \mathbf{V}_{v,k}^{[:,j]}\left(\mathbf{V}_u^{[:,i]}\right)^T + \lambda\mathbf{M}_u & (\theta = \mathbf{M}_u), \\ -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\left(\mathbf{M}_{c,k}\right)^T \mathbf{V}_{u,k}^{[:,i]}\left(\mathbf{V}_v^{[:,j]}\right)^T + \lambda\mathbf{M}_v & (\theta = \mathbf{M}_v), \\ -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\left(\mathbf{C}_k^{[m,:]}\right)^T \alpha_{i,j,k} + \lambda\mathbf{T}_c^{[:,m,n]} & (\theta = \mathbf{T}_c^{[:,m,n]}), \\ -\sum_{(i,j,k)\in\tau} \mathscr{E}_{i,j,k}\beta_{i,j,k}\left(\mathbf{T}_c^{[:,m,:]}\right)^T + \lambda\mathbf{C}_k^{[m,:]} & (\theta = \mathbf{C}_k^{[m,:]}), \end{cases} \tag{18}$$

where $\mathscr{E}_{i,j,k} = r_{i,j,k} - \hat{r}_{i,j,k}$ is an error in the loss function (15). $\alpha_{i,j,k}$ and $\beta_{i,j,k}$ are intermediate variables for (18) as follows:

$$\alpha_{i,j,k} = \mathbf{M}_u^{[m,:]}\mathbf{V}_u^{[:,i]}\mathbf{V}_{v,k}^{[n,j]} + \mathbf{M}_v^{[m,:]}\mathbf{V}_v^{[:,j]}\mathbf{V}_{u,k}^{[n,i]}, \tag{19}$$

$$\beta_{i,j,k} = \mathbf{M}_u^{[m,:]}\mathbf{V}_u^{[:,i]}\left(\mathbf{V}_{v,k}^{[:,j]}\right)^T + \mathbf{M}_v^{[m,:]}\mathbf{V}_v^{[:,j]}\left(\mathbf{V}_{u,k}^{[:,i]}\right)^T. \tag{20}$$

5.2.3 Optimization method for learning algorithm

Second-order optimization methods for a constrained multivariate loss function effectively explore the non-linear loss function with a large number of parameters like (15). Unlike the fixed learning rate γ in (17), they calculate an optimal learning rate that changes as parameters are updated in a learning. This effectively finds the minimum cost of a loss function and learns the optimal solution for parameters. Thus, as shown in Algorithm 1, we adopt the limited-memory BFGS (L-BFGS) method which are widely used in deep learning. Note that we do not need to configure the learning rate in (17). L-BFGS exploits a loss function and its gradient calculated by (18) according to each training data.

Algorithm 1 Overall learning algorithm for CT-CAR

```
Input: Training dataset $\tau$, Past-learned result $\Omega$,
       Set of initial parameters $\Theta = \{\mathbf{C}_k, \mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v, \mathbf{V}_u, \mathbf{V}_v\}$,
       The number of iteration $Iter$, and The number of batch $b_{num}$.
Output: $\mathbf{C}_k, \mathbf{T}_c, \mathbf{M}_u, \mathbf{M}_v, \mathbf{V}_u, \mathbf{V}_v$.
1  Randomly initialize $\Theta$;
2  Check reusable conditions;
3  if reusable conditions are satisfied then
4      Replace $\mathbf{C}_k$ with PLVCs $\subset \Omega$ by (13), (14);
5  end
6  for $i$ in $Iter$ do
7      Shuffle $\tau$;
8      Divide $\tau$ into $b_{num}$ batches $\mathscr{B}$;
9      foreach batch $\in \mathscr{B}$ do
10         # Start mini-batch L-BFGS
11         Calculate recommendation ratings with (12);
12         Measure performance of $\Theta$;
13         Explore minimum of $\mathscr{L}(\Theta)$ with (18);
14         Determine final gradients of $\Theta$;
15         Updatae $\Theta$ by gradients;
16     end
17 end
```

However, L-BFGS commonly needs batch learning which calculates all of the gradients on entire training dataset. Thus, L-BFGS do not have a scalability with the large-scaled datasets. In order to address the weakness for scalability, we also apply the mini-batch learning for large-scaled datasets as shown in Algorithm 1. At every learning iteration i, the mini-batch learning shuffles training dataset τ, and the shuffled dataset is divided into small chunks $\mathscr{B}$ called mini-batches. Then, each iteration i calculates gradients for all the batches and updates parameters. This is applicable to large-scaled datasets because it calculates the gradient and updates the parameters for each $batch \in \mathscr{B}$, not for each tuple in a dataset. Thus, mini-batch L-BFGS shows a fast optimization when a training dataset is even large such as the MovieLens (Harper and Konstan 2015) and the Netflix (Bennett and Lanning 2007) datasets.

5.2.4 Time complexity analysis

Time complexity analysis plays an important role in assessing the performance of recommender systems. The step of updating parameters (17) in our learning algorithm has the highest complexity among the other steps. Table 5 shows the time complexity of updating each parameter. Since all the parameters are simultaneously updated, the parameters share the overall complexity $\mathcal{O}(d_{uv}^2 \times n_c \times d_c \times |\tau|)$. n_c, d_c, and d_{uv} are fixed values according to a training dataset. This means that the complexity grows linearly as the dataset is larger. Note that the number of context instances and the number of context combinations do not affect the complexity. The reason is that each context instance is learned as a latent vector and combined according to the context combination of a training data. Thus, only the number of context types is related to the complexity.

In typical recommender systems, a parameter learning with $d_c = 4$ and $d_{uv} = 5$ shows high enough performance of recommendations (see Section 6.4.5). Furthermore, the number of context types, n_c, is typically less than 10 in various real-world datasets. As you can see in Table 6, the training datasets have less than $n_c = 6$. For example, MovieLens dataset, which are widely used as a large-scaled dataset, also has $n_c = 3$. Thus $d_{uv}^2 \times n_c \times d_c$ is much small than $|\tau|$. As a result, in most training datasets, our learning algorithm has the linear complexity as $|\tau|$. Our CT-CARS has a scalability for large-scaled datasets.

6 Experiments and results

In this section, We first describe the settings in our experiments, including the real-world datasets, comparisons, and evaluation metrics. Next, we discuss several experimental results. We train the parameters Θ using stochastic gradient descent with shuffled mini-batches and L-BFGS update rule. We configure the batch size to 200. All experiments ARE performed on Intel Core i7-4790K CPU @ 4.00GHz and NVIDIA GeForce GTX 1080 GPU, and implemented in Python.

6.1 Real-world datasets

As shown in Table 6, we conduct the experiments on the six real-world datasets including the two large-sized datasets in various domains. Note that we tried to use Netflix dataset (Bennett and Lanning 2007), but it was not the proper dataset for CARS because of the small amount of context. The ratings of all datasets are ranging from 1 to 5. We process the datasets to convert the tuples in the form of text into a numerical ID. Then, we treat the

Table 5 Time complexity of learning parameters for CT-CARS

Parameter	Time complexity
$\mathbf{V}_u$	$\mathcal{O}(d_{uv} \times (d_{uv} + n_c) \times d_c \times |\tau|)$
$\mathbf{V}_v$	$\mathcal{O}(d_{uv} \times (d_{uv} + n_c) \times d_c \times |\tau|)$
$\mathbf{M}_u$	$\mathcal{O}(d_{uv} \times (d_{uv} + n_c) \times d_c \times |\tau|)$
$\mathbf{M}_v$	$\mathcal{O}(d_{uv} \times (d_{uv} + n_c) \times d_c \times |\tau|)$
$\mathbf{T}_c$	$\mathcal{O}(d_{uv}^2 \times n_c \times d_c \times |\tau|)$
$\mathbf{C}_k$	$O(d_{uv} \times (d_{uv} + n_c) \times d_c \times |\tau|)$

Table 6 Statistics of six real-world datasets, including two large-sized datasets (below the line)

	Entity information			Contextual information				Density		
	#Users	#Items	#Ratings	#Types	#Instances	#Cases	Com.	C. density	R. density	Std. Error
Food	212	20	6,104	2	{2,3}	6	○	100%	24%	0.0852
Hotel	1,202	1,890	4,643	6	{9,6,48,27,4,5}	2,012	○	2.6e-2%	2.7e-8%	0.045
Restaurant	50	40	2,310	3	{3,4,2}	14	▲	58.3%	4.81%	5.976
Movie	97	79	5,035	3	{3,3,4}	13	×	36.1%	1.83%	3.623
ML	6,040	3,706	1M	4	{2,7,21,18}	3,877	○	73.3%	8.4e-6%	2.48
Ex-ML	6,040	3,706	1M	6	{2,7,21,4,2,18}	19,084	○	45.1%	1.1e-6%	2.48

empty context instance by replacing with context instance {general}, which means general characteristic of its context type. We use the following datasets:

- **Food** (Ono et al. 2009) is a dataset with uniform distribution of users, items, and context instances. There are 2 context types which are hunger and virtuality. There is no empty context instance for all context types.
- **Hotel** (Zheng et al. 2012b) also has no empty context instance. However, some context types have a large set of context instances. The dataset has fewer ratings compared to the number of users and items.
- **Movie** (Zheng et al. 2015b) has 3 context types which are time, location, and companion. Since some tuples have no contextual information, the dataset is not completeness.
- **Restaurant** (Ramirez-Garcia and García-Valdez 2014) has 3 context types which are time, location, and occupation. Some tuples have empty context instances in time, location. Thus, we add a context instance {general}.
- **MovieLens 1M**[1] (Harper and Konstan 2015) is used as ML domain in our experiments. The dataset consists of 1 million ratings by 6040 users who rated 20 and more items, and has 4 context types which are gender, age, occupation, and genre (the first of genre list).
- **Ex-MovieLens 1M** is the extended MovieLens dataset. We convert the timestamp of each rating into the local time based on the user's zip-code. Thus, two context types are added: daytime to indicate the part of the day, and isweekend to indicate whether the weekend or weekday. This dataset is used as Ex-ML domain.

Each dataset has entity information and contextual information. The contextual information consists of the number of context types (*#Types*), the number of context instances per context type (*#Instances*), the number of observed context combinations (*#Cases*), and *Completeness*. *Completeness* indicates that context instances of a tuple are all recorded (○) or partially recorded (▲) or not recorded (×). In order to analyze the experimental results in detail, we analyze characteristics of each dataset such as the context density (*C. density*), the rating density (*R. density*), and the standard deviation (*Std. Error*). *R. density* is calculated as $\#Ratings/(\#Users \times \#Items \times \prod(\#Instances))$. *C. density* is calculated as $\#Cases/\prod(\#Instances)$. *Std. Error* indicates distribution degree of the number of each user's ratings. The low ratio of *#Ratings* to *#Users* causes the wide distribution and *Std. Error* decreases. On the other hand, if the numbers of user's ratings are not evenly distributed, *Std. Error* will increase. For example, even though the ratio of *#Ratings* to *#Users* is similar, *Std. Error* of Restaurant is higher than Movie. The reason is that the numbers of ratings for all users in Movie are distributed evenly than Restaurant. Therefore, it is easy to analyze Movie which is rich in data for each user than Restaurant. Specifically, since only context types of Ex-ML are extended from ML, *Std. Errors* of ML and Ex-ML are same.

6.2 Comparisons

We mainly compare CT-CARS with the following three previous works. Specifically, we will focus on COT which is a state-of-the-art recommender system.

[1] http://www.grouplens.org/datasets/movielens/.

- **FM** (Rendle et al. 2011) is the recommender system which shows applicability on various contexts and domains. We employ libFM[2] (Rendle 2012) to evaluate FM.
- **HeteroMF** (Jamali and Lakshmanan 2013) learns a context-dependent matrix for each context combination. The complexity and the computational overhead are very high, and accuracy of recommendations is also low.
- **COT** (Liu et al. 2015a) is one of the state-of-the-art recommender systems with a stepwise approach. COT uses contextual operating tensors to capture the user-contexts and item-contexts relationships. However, they cannot elaborately integrate context instances.

6.3 Evaluation metrics

To properly evaluate the recommendation system, both learning accuracy and ranking quality should be evaluated. The learning accuracy indicates how close the predicted ratings are to the observed ratings in datasets. A list of item rankings for a specific user is sorted by user's ratings. The ranking quality indicates how similar the predicted item rankings is to the observed item rankings in datasets.

- **Learning Accuracy** can be estimated by the differences an observed rating in training dataset and a predicted rating. We adopt the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE) which have been widely used to measure the performance of recommendations. The two metrics are as follows:

$$RMSE = \sqrt{\frac{1}{|\tau_{test}|} \sum_{(i,j,k)\in\tau_{test}} (r_{i,j,k} - \hat{r}_{i,j,k})^2}, \tag{21}$$

$$MAE = \frac{1}{|\tau_{test}|} \sum_{(i,j,k)\in\tau_{test}} |r_{i,j,k} - \hat{r}_{i,j,k}|, \tag{22}$$

where τ_{test} denotes the test dataset, and $|\tau_{test}|$ d enotes the number of ratings in τ_{test}.
- **Ranking Quality** is commonly used in information retrieval, recommender systems, and so on. To evaluate the ranking quality, we used the Normalized Discounted Cumulative Gain (nDCG). We first select 500 tuples in each dataset, and get the items of top-M rankings from 500 tuples. Then, nDCG@M is calculated by comparing the rankings of the items before learning (ideal) and the same items after learning. nDCG is as follows:

$$DCG@M = \sum_{i=1}^{M} \frac{R(item_i)}{log_2^{(i+1)}}, \qquad nDCG@M = \frac{DCG@M}{IDCG@M}, \tag{23}$$

where $item_i$ denotes i-th selecting item of top-M, and $R(item_i)$ gets the observed user's rating of $item_i$. Ideal nDCG, IDCG@M, indicates that the rankings of the top-M items after learning are ideal. We select the items of top-10 rankings for nDCG@10. nDCG varies from 0.0 to 1.0 that 1.0 indicates the ideal rankings of predicted items.

[2] http://www.libfm.org/.

6.4 Results and discussions

Our experiments are intended to demonstrate that CT-CARS using the proposed context tensor can improve contextual expression power and fully take advantage of correlations among multiple contexts. To this end, we conduct several experiments to compare the performance of CT-CARS with several state-of-the-art works in terms of learning accuracy and ranking quality. Also, the results contain the utilizing parameter initialization with PLVCs, the scalability analysis. and the impact of d_c and d_{uv}. Note that we adopt a 5-fold cross validation to evaluate the performance of recommender systems. Each experimental result is the average value of the five 5-fold cross validations.

6.4.1 *Learning accuracy and ranking quality*

Table 7 shows the average values of learning accuracy as RMSE, MAE and the ranking quality as nDCG@10 on six datasets including two large-scaled datasets. We also indicate "$\pm$ 2 standard deviations" below the average RMSE, MAE, and nDCG values, which means that a new value has about 95% probability of being within 2 standard deviations of the average. Lower RMSE and MAE and higher nDCG indicate better performance. We configure $d_{uv} = 5$ and $d_c = 4$ because the performances stay nearly stable after the configuration. The percentage values indicate the improved performance ratio of CT-CARS in RMSE, MAE, and nDCG compared with the conventional recommender systems. Note that we mainly compare our CT-CARS with COT which is a state-of-the-art work. CT-CARS outperforms others in learning accuracy and ranking quality, which utilize the context tensor to effectively combine multiple LVCs. Generally, the experimental results show that datasets with lower *Std. Error* and higher *C. density* and *R. density* lead to the high learning accuracy and ranking quality. Among all datasets, Food has highest *C. density*, *R. density* and the even distribution degree of the number of each user's ratings (low *Std. Error*). Thus, Food can easily analyze users, items, and contexts to capture the user-contexts and item-contexts relations, and shows the best performance among six datasets. On the other hand, Hotel has low *C. density*, *R. density*, and ratio of *#Ratings* to *#Users*. Thus, Hotel shows the low performance among six datasets even in low *Std. Error*. As discussed in Section 6.1, since Movie is easy to analyze than Restaurant, Movie shows slightly better performance than Restaurant. We could also see the recommendation performance of Ex-ML is higher than ML, even in the same dataset MovieLens. The reason is that Ex-ML has more context types (daytime and isweekend), which are extended from ML. Thus, we can find that rich contextual information helps to improve the performance of recommendations.

CT-CARS achieve the best among the previous works. FM cannot capture the contextual influence on users and items, respectively, and HeteroMF must learn tremendous context-dependent matrices for all context combinations. Due to the problems, FM and HeteroMF show low accuracy and ranking quality than the others. CT-CARS and COT learn context instances as latent vectors and combine them according to a context combination. They also capture the contextual influence on users and items, respectively. Thus, CT-CARS and COT achieve higher performance than FM and HeteroMF. Moreover, we can observe that CT-CARS outperforms COT on all datasets. This proves that our proposed context tensor which captures various correlations among contexts is helpful. Specifically, the improvement of CT-CARS on the Hotel is the highest. Thus, the context tensor particularly helps to identify the contextual influence of numerous contexts. In the case of large-scaled datasets, the performance improvement from ML to Ex-ML with CT-CARS is higher than previous methods.

Table 7 Comparison with previous works in RMSE, MAE, and nDCG@10 ($d_{uv} = 5, d_c = 4$)

	CT-CARS			FM			HeteroMF			COT		
	RMSE	MAE	nDCG	RMSE	MAE	nDCG	RMSE	MAE	nDCG	RMSE	MAE	nDCG
Food	1.059	0.832	0.631	1.072	0.862	0.537	1.206	0.938	0.527	1.079	0.827	0.581
	(± 0.0087)	(± 0.0069)	(± 0.0032)	(± 0.0092)	(± 0.0082)	(± 0.0039)	(± 0.0101)	(± 0.0102)	(± 0.0031)	(± 0.0090)	(± 0.0073)	(± 0.0028)
	–	–	–	+ 1.2%	+ 3.5%	+ 17.5%	+ 12.2%	+ 11.3%	+ 19.7%	+ 1.9%	+ 0.6%	+ 8.6%
Hotel	1.252	1.170	0.281	1.544	1.318	0.265	1.475	1.386	0.218	1.321	1.248	0.296
	(± 0.0106)	(± 0.0081)	(± 0.0028)	(± 0.0214)	(± 0.0110)	(± 0.0024)	(± 0.0183)	(± 0.0158)	(± 0.0027)	(± 0.0121)	(± 0.0090)	(± 0.0025)
	–	–	–	+ 18.9%	+ 11.2%	+ 6.0%	+ 15.1%	+ 15.6%	+ 28.9%	+ 5.2%	+ 6.3%	− 5.1%
Restaurant	1.155	0.824	0.412	1.247	0.856	0.349	1.222	0.876	0.334	1.163	0.841	0.361
	(± 0.0091)	(± 0.0077)	(± 0.0048)	(± 0.0108)	(± 0.0093)	(± 0.0051)	(± 0.0118)	(± 0.0097)	(± 0.0055)	(± 0.0090)	(± 0.0081)	(± 0.0049)
	–	–	–	+ 7.4%	+ 3.7%	+ 18.1%	+ 5.5%	+ 5.9%	+ 23.4%	+ 0.7%	+ 2.0%	+ 14.1%
Movie	1.148	0.812	0.405	1.236	0.895	0.414	1.339	0.902	0.327	1.165	0.862	0.378
	(± 0.0071)	(± 0.0061)	(± 0.0045)	(± 0.0102)	(± 0.0099)	(± 0.0049)	(± 0.0093)	(± 0.0090)	(± 0.0048)	(± 0.0082)	(± 0.0070)	(± 0.0043)
	–	–	–	+ 7.1%	+ 9.2%	− 2.2%	+ 14.3%	+ 10.0%	+ 2.4%	+ 1.5%	+ 5.8%	+ 7.1%
ML	1.186	0.903	0.493	1.261	0.943	0.401	1.359	0.982	0.371	1.193	0.912	0.471
	(± 0.0112)	(± 0.0092)	(± 0.0053)	(± 0.0172)	(± 0.0116)	(± 0.0055)	(± 0.0158)	(± 0.0139)	(± 0.0049)	(± 0.0118)	(± 0.0101)	(± 0.0057)
	–	–	–	+ 5.9%	+ 4.2%	+ 22.9%	+ 12.7%	+ 8.0%	+ 32.9%	+ 0.6%	+ 1.0%	+ 4.6%
Ex-ML	1.164	0.871	0.496	1.191	0.912	0.415	1.363	0.991	0.382	1.175	0.883	0.472
	(± 0.0107)	(± 0.0081)	(± 0.0047)	(± 0.0142)	(± 0.0111)	(± 0.0049)	(± 0.0133)	(± 0.0121)	(± 0.0049)	(± 0.0111)	(± 0.0083)	(± 0.0046)
	–	–	–	+ 2.3%	+ 4.5%	+ 19.5%	+ 14.6%	+ 12.1%	+ 29.8%	+ 0.9%	+ 1.3%	+ 5.1%

6.4.2 Utilizing past-learned parameters

CT-CARS and COT basically initialize the parameters at random. This initialization causes different learning performance for each learning, and the low reliability of the recommender systems is reduced. To cope with the problem, we initialize the parameters with the past-learned LVC which showed high performance. In order to analyze the impact of PLVC on each recommender system, we compare CT-CARS with COT depending on whether PLVC is used or not. As shown in Fig. 6a, both CT-CARS and COT using PLVC outperform random parameter initialization. Moreover, RMSE of CT-CARS with PLVC is higher than COT. This result proves that PLVCs are effectively reflected in new learning of CT-CARS which has a context tensor. Note that the context tensor is elaborately combined with LVCs. Thus, this lead to high performance and the reliability of the recommender system.

The learning algorithm takes PLVCs or random parameters as input, and outputs learned parameters. Figure 6b shows vector similarities between the inputs and the outputs by using a cosine similarity equation. The vector similarity between the PLVCs and the learned parameters of using them is higher than random parameters. This indicate that PLVCs are helpful for the parameter initialization. The similarity of CT-CARS with PLVCs is higher

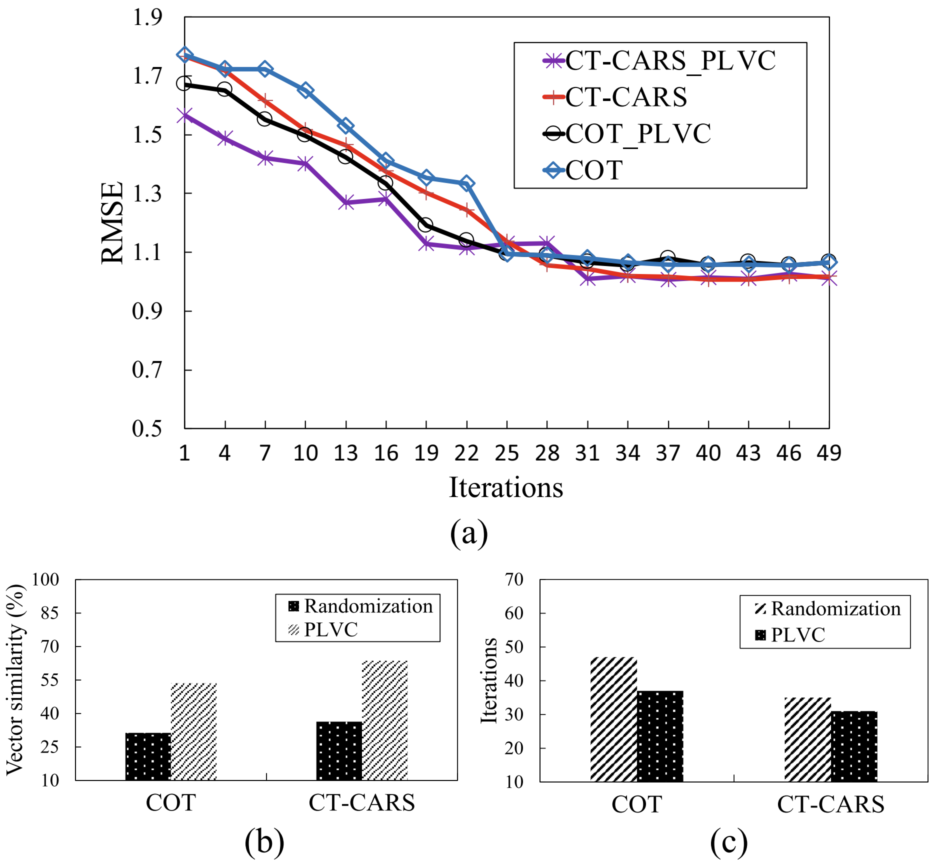


Fig. 6 Impact of past-learned parameters

than COT with PLVCs. This indicates that the context tensor of CT-CARS preserves the meaning of each PLVC more effectively than COT. Figure 6c shows that both CT-CARS and COT with PLVC performed less iterations than the random initialization. Moreover, CT-CARS was faster than COT. This indicates that PLVCs also help to learn faster. As a result, the parameter learning with PLVCs improves the reliability.

6.4.3 RMSE and MAE with iterations

As shown in Figs. 7 and 8, RMSE and MAE show the convergence curves, which decrease according to the learning iterations. In general, RMSE and MAE show similar patterns. Since Food has highest *C. density*, *R. density* and low *Std. Error* among all datasets, the RMSE and MAE are the lowest. We observe that FM and HeteroMF show increasing patterns of RMSE and MAE on some datasets. The reason is that HeteroMF had a higher complexity than others because of too many context-dependent matrices and FM could not capture the context influence on entities, respectively. CT-CARS and COT show high

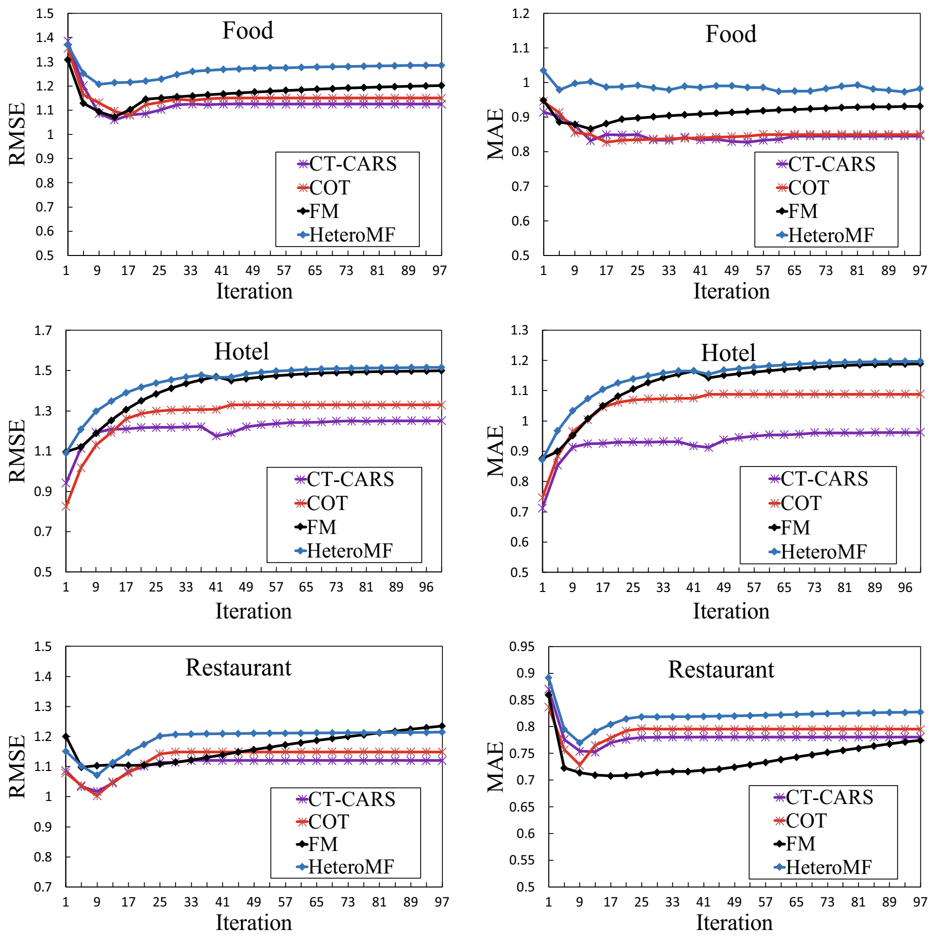


Fig. 7 RMSE and MAE on Food, Hotel, and Restaurant datasets

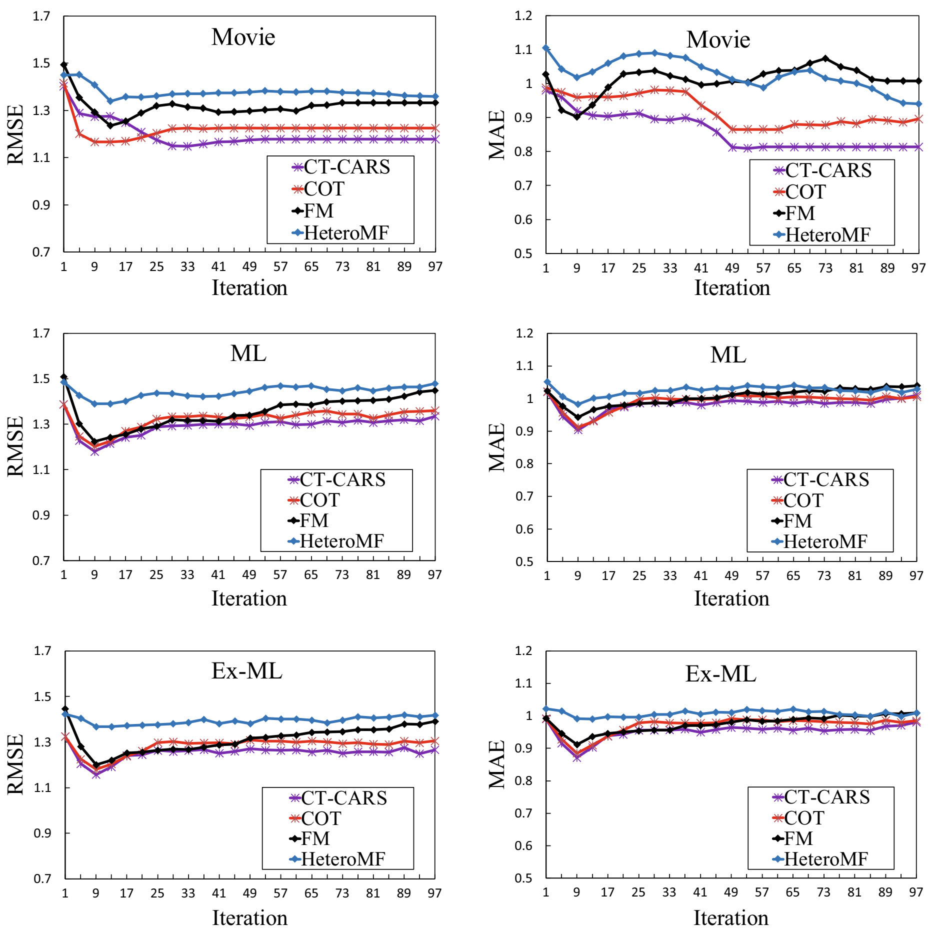


Fig. 8 RMSE and MAE on Movie, ML, and Ex-ML datasets

performance and decreasing patterns, except on the Hotel. As discussed above, Hotel is not easy to analyze because Hotel has the lowest *C. density* and *R. density*. CT-CARS and COT integrate LVCs according to a context combination. They do not need to learn all context combinations like HeteroMF which has the highest complexity. Thus, the low complexities of CT-CARS and COT help to learn faster than HeteroMF. Among CT-CARS and COT, CT-CARS shows a faster and lower convergence than COT. Since the proposed context tensor of CT-CARS can elaborately capture the contextual influences than COT, CT-CARS can more accurately predict a user's perference. Specifically, we find patterns which initially decrease, and gradually increase over time. This results from the over-fitting problem which the generality of a learning decreased as learning iterated. Thus, high iterations do not always lead to the high performance.

6.4.4 Scalability analysis

We conduct an experiment for the complexity of our algorithm. As discussed in Section 6.4, the overall complexity of our parameter learning algorithm is $\mathcal{O}(d_{uv}^2 \times n_c \times d_c \times |\tau|)$.

This means that the complexity is nearly linear depends on the size of the datasets. Thus, our CT-CARS has the scalability on large-scaled datasets. In order to demonstrate the scalability of our CT-CARS, we first construct 10 subsets from the large-scaled dataset, Ex-ML, according to the number of training data. The subsets consist of ranging from 100K to 1M in 100K units. Then, we measure the average runtime of iterations in the learning on each subset. Moreover, we evaluate RMSE of the learnings. Note that we configure $d_{ui} = 5$ and $d_c = 4$, and use mini-batch L-BFGS with both 100 batches and 200 batches.

As shown in Fig. 9, the runtime increases linearly as the dataset is larger in L-BFGS with 100 batches and 200 batches. This shows that CT-CARS has scalability even on large-scaled datasets. The learning with 200 batches is faster than 100 batches. Since parameters are updated per batch, the learning with 200 batches shows fast convergence. This is more prominent in large-scaled datasets. However, the increasin‘g number of batches does not mean that the runtime will decrease linearly. The learning with 200 batches do not show 2$\times$ faster learning than 100 batches. The reason is that each batch does not contain all entities and contexts. Thus, the parameter learning for each batch is limited. We can also see that the larger the training data size, the higher the learning accuracy. Moreover, the learning accuracy with 200 batches is slightly higher than 100 batches.

6.4.5 *The number of latent factors*

Lastly, we discuss the number of latent factors, d_{uv} and d_c, which are one of the determinants of learning accuracy. Since our recommendation rating function is a stepwise approach, the number of the latent factors can be configured separately. The latent factors of LVE and LVC indicate characteristics of entities and contexts. Thus, increasing d_{uv} and d_c can represent a user, an item, and a context in detail. This improve the performance of recommendations. However, all values more than a certain value no longer improve the learning accuracy and only increase the complexity of the learning.

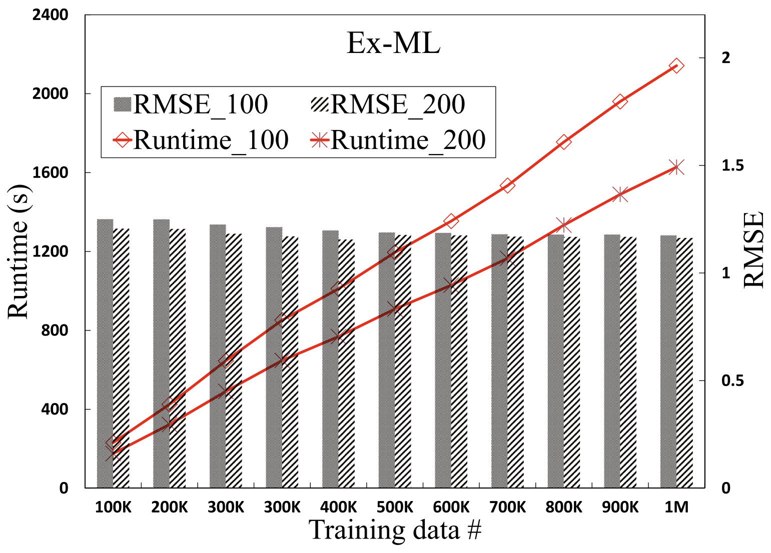


Fig. 9 Runtime of learning based on the number of training data

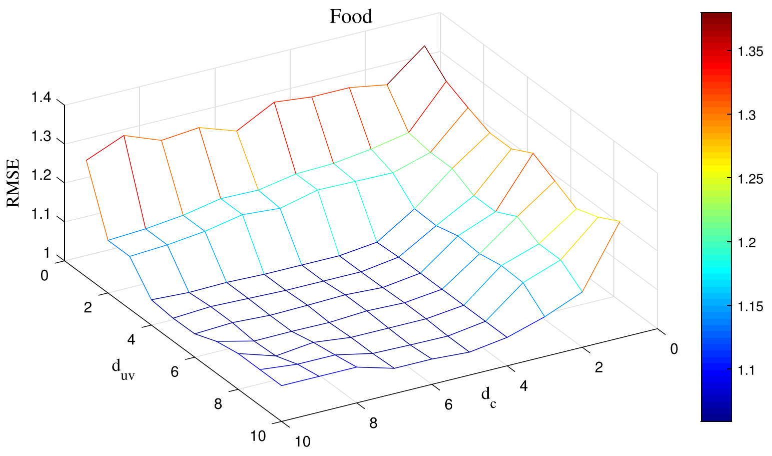


Fig. 10 RMSE of CT-CARS with a variety of d_{uv} and d_c

As shown in Fig. 10, we conduct learnings and evaluate RMSE on the Food dataset while changing d_{uv} and d_c from 1 to 10. The configuration $d_{uv} = 5$ and $d_c = 4$ is enough to get the high performance. All datasets in Table 6 have the much larger number of entity combinations than the number of context combinations. This means that LVEs should be more detailed than LVCs. Especially, the learnings with the configuration beyond $d_{uv} = 5$ and $d_c = 4$ show rather low performances. Too many latent factors have high coverage for a training dataset. on the other hand, coverage for data not participating in learning is low. For this reason, we configure $d_{uv} = 5$ and $d_c = 4$ in all the previous experiments.

7 Conclusion and future work

One of the most challenges in the recent latent factor models is the unreliability of recommendations. Since learned parameters depend on initial parameters, the conventional models often show low reliability. To deal with the problems, we propose a parameter initialization with PLVCs. The initialization improves the reliability and reduces the iterations of learning. Furthermore, previous models integrate the multiple LVCs into a single latent vector by a weighted sum with fixed weight values for context types. Thus, they cannot effectively integrate LVCs, and dilute the meaning of each LVC. Our recommendation rating function and learning algorithm utilize the proposed context tensor in order to preserve the contextual influence as much as possible. Experimental results using six real-world datasets show that CT-CARS outperforms previous models in both accuracy and ranking quality.

There is a limitation of our experiments that we do not consider very large datasets such as MovieLens 20M. Thus, we will perform experiments on the datasets in future works. Abundant data from IoT and mobile environments can be exploited as context data for recommender systems. Since unnecessary context data dilute the expression of latent vectors, sorting out appropriate context data is crucial in order to improve the accuracy of recommendations. Therefore, we will group the influential context types depending on the domain to filter out useless context types. Furthermore, in order to treat the problems of data sparsity and cold start on each recommendation domain, we will augment a rating tensor by using a semantic reasoning to supplement information for entities. Moreover, we will handle the approach which adaptively changes latent segments of contextual information based on its changing rates.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP; Ministry of Science, ICT & Future Planning) (NRF-2016R1A2B4015873).

References

Banerjee, J., Raravi, G., Gupta, M., Ernala, S.K., Kunde, S., Dasgupta, K. (2016). CAPReS: context aware persona based recommendation for shoppers. In *Proceedings of the thirtieth AAAI conference on artificial intelligence, AAAI'16* (pp. 680–686).

Bennett, J., & Lanning, S. (2007). The Netflix prize. In *Proceedings of the KDD cup and workshop*.

Chen, C., Zheng, X., Wang, Y., Hong, F., Chen, D. (2016). Capturing semantic correlation for item recommendation in tagging systems. In *Proceedings of the thirtieth AAAI conference on artificial intelligence, AAAI'16* (pp. 108–114).

Chen, Z., Jiang, Y., Zhao, Y. (2010). A collaborative filtering recommendation algorithm based on user interest change and trust Evaluation. *Journal of Digital Content Technology and its Applications*, *4*(9), 106–113.

Christensen, I., Schiaffino, S., Armentano, M. (2016). Social group recommendation in the tourism domain. *Journal of Intelligent Information Systems*, *47*(2), 209–231.

Deshpande, M., & Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, *22*(1), 143–177.

Drumond, L., Rendle, S., Schmidt-Thieme, L. (2012). Predicting RDF triples in incomplete knowledge bases with tensor factorization. In *Proceedings of the 27th annual ACM symposium on applied computing, SAC'12* (pp. 326–331).

Harper, F.M., & Konstan, J.A. (2015). The MovieLens datasets: history and context. *ACM Transactions on Interactive Intelligent Systems*, *5*(4), 19.

Jamali, M., & Lakshmanan, L. (2013). Heteromf: recommendation in heterogeneous information networks using context dependent factor models. In *Proceedings of the 22nd international conference on World Wide Web, WWW'13* (pp. 643–654).

Karatzoglou, A., Amatriain, X., Baltrunas, L., Oliver, N. (2010). Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on recommender systems, RecSys'10* (pp. 79–86).

Koren, Y., Bell, R., Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.

Liu, X., & Wu, W. (2015). Learning context-aware latent representations for context-aware collaborative filtering. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval* (pp. 887–890).

Liu, Q., Wu, S., Wang, L. (2015a). COT: contextual operating tensor for context-aware recommender systems. In *Proceedings of the twenty-ninth AAAI conference on artificial intelligence, AAAI'15* (pp. 203–209).

Liu, Q., Wu, S., Wang, L. (2015b). Collaborative prediction for multi-entity interaction with hierarchical representation. In *Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM'15* (pp. 613–622).

Liu, L., Tan, P., Liu, X. (2015c). Mf-tree: matrix factorization tree for large multi-class learning. In *Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM'15* (pp. 881–890).

Ma, H., Yang, H., Lyu, M.R., King, I. (2008). Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM'08* (pp. 931–940).

Man, T., Shen, H., Huang, J., Cheng, X. (2015). Context-adaptive matrix factorization for multi-context recommendation. In *Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM'15* (pp. 901–910).

Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781.

Nguyen, T.V., Karatzoglou, A., Baltrunas, L. (2014). Gaussian process factorization machines for context-aware recommendations. In *Proceedings of the 37th international ACM SIGIR conference on Research and development in information retrieval* (pp. 63–72).

Nickel, M., Tresp, V., Kriegel, H.P. (2011). A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning, ICML'11* (pp. 809–816).

Nickel, M., Tresp, V., Kriegel, H.P. (2012). Factorizing YAGO: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web, WWW'12* (pp. 271–280).

Niederee, C., Kanhabua, N., Gallo, F., Logie, R.H. (2015). Forgetful digital memory: Towards brain-inspired long-term data and information management. *ACM SIGMOD Record*, *44*(2), 41–46.

Ono, C., Takishima, Y., Motomura, Y., Asoh, H. (2009). Context-aware preference model based on a study of difference between real and supposed situation data. In *International Conference on User Modeling, Adaptation, and Personalization, UMAP'09* (pp. 102–113).

Pennock, D.M., Horvitz, E., Lawrence, S., Giles, C.L. (2000). Collaborative filtering by personality diagnosis: a hybrid memory-and model-based approach. In *Proceedings of the sixteenth conference on uncertainty in artificial intelligence, UAI'00* (pp. 473–480).

Ramirez-Garcia, X., & García-Valdez, M. (2014). Post-filtering for a restaurant context-aware recommender system. In *Recent advances on hybrid approaches for designing intelligent systems* (pp. 695–707).

Rendle, S. (2012). Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology*, *3*(3), 57.

Rendle, S., Gantner, Z., Freudenthaler, C., Schmidt-Thieme, L. (2011). Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in information retrieval* (pp. 635–644).

Salakhutdinov, R., & Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning, ICML'08* (pp. 880–887).

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A. (2014). Cars2: learning context-aware representations for context-aware recommendations. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management, CIKM'14* (pp. 291–300).

Singh, A.P., & Gordon, G.J. (2008). Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining,* (pp. 650–658).

Tran, T.N.T., Atas, M., Felfernig, A., Stettinger, M. (2017). An overview of recommender systems in the healthy food domain. *Journal of Intelligent Information Systems*, *50*(3), 501–526.

Xiong, L., Chen, X., Huang, T.K., Schneider, J.G., Carbonell, J.G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM international conference on data mining* (pp. 211–222).

Zhang, Z., Jin, X., Li, L., Ding, G., Yang, Q. (2016). Multi-domain active learning for recommendation. In *Proceedings of the thirtieth AAAI conference on artificial intelligence, AAAI'16* (pp. 2358–2364).

Zheng, Y., & Xie, X. (2011). Learning travel recommendations from user-generated GPS traces. *ACM Transactions on Intelligent Systems and Technology*, *2*(1), 2.

Zheng, Y., Burke, R., Mobasher, B. (2012a). Optimal feature selection for context-aware recommendation using differential relaxation. In *Proceedings of the sixth ACM conference on Recommender systems, RecSys'12* (p. 12).

Zheng, Y., Burke, R., Mobasher, B. (2012b). Differential context relaxation for context-aware travel recommendation. In *International Conference on Electronic Commerce and Web Technologies* (pp. 88–99).

Zheng, Y., Burke, R., Mobasher, B. (2013). Recommendation with differential context weighting. In *International conference on user modeling, adaptation, and personalization, UMAP'13* (pp. 152–164).

Zheng, Y., Mobasher, B., Burke, R. (2014). Deviation-based contextual SLIM recommender. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management, CIKM'14* (pp. 271–280).

Zheng, Y., Mobasher, B., Burke, R. (2015a). Integrating context similarity with sparse linear recommendation model. In *International conference on user modeling, adaptation, and personalization, UMAP'15* (pp. 370–376).

Zheng, Y., Mobasher, B., Burke, R. (2015b). Carskit: a java-based context-aware recommendation engine. In *Data mining workshop, ICDMW'15* (pp. 1668–1671).