Compositional Coding for Collaborative Filtering*

Chenghao Liu[1,2], Tao Lu[2,5], Xin Wang[4], Zhiyong Cheng[6], Jianling Sun[2,5], Steven C.H. Hoi[1,3]
[1]Singapore Management University, [2]Zhejiang University, [3]Salesforce Research Asia, [4]Tsinghua University,
[5]Alibaba-Zhejiang University Joint Institute of Frontier Technologies,
[6]Shandong Computer Science Center (National Supercomputer Center in Jinan)
Qilu University of Technology (Shandong Academy of Sciences)
{twinsken,3140102441,sunjl}@zju.edu.cn,xin_wang@tsinghua.edu.cn,jason.zy.cheng@gmail.com,chhoi@smu.edu.sg

ABSTRACT

Efficiency is crucial to the online recommender systems, especially for the ones which needs to deal with tens of millions of users and items. Because representing users and items as binary vectors for Collaborative Filtering (CF) can achieve fast user-item affinity computation in the Hamming space, in recent years, we have witnessed an emerging research effort in exploiting binary hashing techniques for CF methods. However, CF with binary codes naturally suffers from low accuracy due to limited representation capability in each bit, which impedes it from modeling complex structure of the data.

In this work, we attempt to improve the efficiency without hurting the model performance by utilizing both the accuracy of real-valued vectors and the efficiency of binary codes to represent users/items. In particular, we propose the Compositional Coding for Collaborative Filtering (CCCF) framework, which not only gains better recommendation efficiency than the state-of-the-art binarized CF approaches but also achieves even higher accuracy than the real-valued CF method. Specifically, CCCF innovatively represents each user/item with a set of binary vectors, which are associated with a sparse real-value weight vector. Each value of the weight vector encodes the importance of the corresponding binary vector to the user/item. The continuous weight vectors greatly enhances the representation capability of binary codes, and its sparsity guarantees the processing speed. Furthermore, an integer weight approximation scheme is proposed to further accelerate the speed. Based on the CCCF framework, we design an efficient discrete optimization algorithm to learn its parameters. Extensive experiments on three real-world datasets show that our method outperforms the state-of-the-art binarized CF methods (even achieves better performance than the real-valued CF method) by a large margin in terms of both recommendation accuracy and efficiency. We publish our project at https://github.com/3140102441/CCCF.

CCS CONCEPTS

• **Information systems → Recommender System**; • **Human-centered computing → Collaborative filtering**;

KEYWORDS

Recommendation, Collaborative Filtering, Discrete Hashing

ACM Reference Format:
Chenghao Liu[1,2], Tao Lu[2,5], Xin Wang[4], Zhiyong Cheng[6], Jianling Sun[2,5], Steven C.H. Hoi[1,3]. 2019. Compositional Coding for Collaborative Filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331206

1 INTRODUCTION

Real-world recommender systems often have to deal with large numbers of users and items especially for online applications, such as e-commerce or music streaming services [2, 3, 13, 14, 27]. For many modern recommender systems, a de facto solution is often based on Collaborative Filtering (CF) techniques, as exemplified by Matrix Factorization (MF) algorithms [8]. The principle of MF is to represent users' preferences and items' characteristics into r low-dimensional vectors, based on the $m \times n$ user-item interaction matrix of m users and n items. With the obtained user and item vectors (in the offline training stage), during the online recommendation stage, the preference of a user towards an item is computed by the dot product of their represented vectors. However, when dealing with large numbers of users and items (e.g., millions or even billions of users and items), a naive implementation of typical collaborative filtering techniques (e.g., based on MF) will lead to very high computation cost for generating preferred item ranking list for a target user [11]. Specifically, recommending the top-k preferred items for a user from those n items costs $O(nr + n \log k)$ with real-valued vectors. As a result, this process will become a critical efficiency bottleneck in practice where the recommender systems typically require a real-time response for large-scale users simultaneously. Therefore, a fast and scalable yet accurate CF solution is crucial towards building real-time recommender systems.

Recent years have witnessed extensive research efforts for improving the efficiency of CF methods for scalable recommender systems. One promising paradigm is to explore the hashing techniques [18, 33, 35] to represent users/items with binary codes instead of the real-value latent factors in traditional MF methods. In this way, the dot-products of user vector and item vector in MF can be completed by fast bit-operations in the Hamming space [35]. Furthermore, by exploiting special data structures for indexing all items, the computational complexity of generating top-K preferred items can achieve sub-linear or even constant [26, 33], which significantly accelerates the recommendation process.

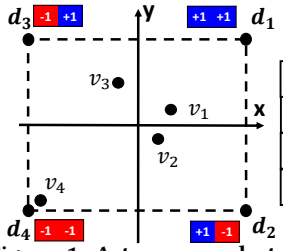
However, learning the binary codes is generally NP-hard [5] due to its discrete constraints. Given this NP-hardness, a two-stage

*Jianling Sun is the corresponding author.

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331206

optimization procedure [18, 33, 35], which first solves a relaxed optimization problem through ignoring the discrete constraints and then binarizes the results by thresholding, becomes a compromising solution. Nevertheless, this solution suffers from a large quantization loss [30] and thus fails to preserve the original data geometry (user-item relevance and user-user relationship) in the continuous real-valued vector space. As accuracy is arguably the most important evaluation metric for recommender systems, researchers put lots of efforts to reduce the quantization loss by direct discrete optimization [12, 16, 30]. In spite of the advantages of this improved optimization method, compared to real-valued vectors, CF with binary codes naturally suffers from low accuracy due to limited representation capability in each bit, which impedes it from modeling complex relationship between users and items.



		Film Title	Genres
v_1	d_1	Star Wars	Action, Adventure
v_2	d_2	Star Wars II	Action, Adventure
v_3	d_3	The Matrix Reloaded	Action, Sci-Fi
v_4	d_4	Titanic	Drama, Romance

Figure 1: A toy example to illustrate the limitation of binary codes of Discrete Collaborative Filtering (DCF) [30]. $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, and $\mathbf{v}_4$ denote the real-valued vectors for item embeddings, and $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$, and $\mathbf{d}_4$ denote the binary codes for item embeddings. According to the film title and genres, v_1 is the most similar to v_2, followed by v_3, while they are all dissimilar to v_4. However, the binary codes learned by DCF cannot preserve the intrinsic similarity due to the limited representation capability of binary codes.

Figure 1 gives an example to illustrate the limit of binary codes. From the "film title" and "genres", we can see that *Star Wars* is the most similar with *Star Wars II*, followed by *The Matrix Reloaded*, and all of them are action movies while *Titanic* is remarkably dissimilar to them which is categorized as a *Drama* movie. The real-valued vectors could easily preserve the original data geometry in the continuous vector space (e.g., intrinsic movie relationships), like $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, and $\mathbf{v}_4$. However, if we preserve the geometric relations of *Star Wars*, *Star Wars II* and *The Matrix Reloaded* by representing them using the binary codes $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ in the Hamming space, the binary code of movie *Titanic* $\mathbf{d}_4$ will become close to $\mathbf{d}_2$ and $\mathbf{d}_3$, which unfortunately leads to a large error.

In this work, we attempt to improve the efficiency without hurting the model performance. We propose a new user/item representation named "Compositional Coding", which utilizes both the accuracy of real-valued vectors and the efficiency of binary codes to represent users and items. To improve the representation capability of the binary codes, each user/item is represented by G components of r-dimensional binary codes (r is relatively small) together with a G-dimensional sparse weight vector. The weight vector is real-valued and indicates the importance of the corresponding component of the binary codes. Compared to the binary codes with same length (equal to Gr), the real-valued weight vector significantly enriches the representation capability. Meanwhile, the sparsity of the weight vector could preserve the high efficiency. To demonstrate how it works, we derive the Compositional Coding for Collaborative Filtering (CCCF) framework. To tackle the intractable discrete optimization of CCCF, we develop an efficient alternating optimization method which iteratively solves mixed-integer programming subproblems. Besides, we develop an integer approximation strategy for the weight vectors. This strategy can further accelerate the recommendation speed. We conduct extensive experiments in which our promising results show that the proposed CCCF method not only improves the accuracy but also boosts retrieval efficiency over state-of-the-art binary coding methods.

2 PRELIMINARIES

In this section, we first review the two-stage hashing method for collaborative filtering. Then, we introduce the direct discrete optimization method, which has been used in Discrete Collaborative Filtering (DCF) [30]. Finally, we discuss the limitation of binary codes in representation capability.

2.1 Two-stage Hashing Method

Matrix Factorization (MF) is the most successful and widely used CF based recommendation method. It represents users and items with real-valued vectors. Then an interaction of the corresponding user and item can be efficiently estimated by the inner product. Formally, given a user-item interaction matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ with m users and n items. Let $\mathbf{u}_i \in \mathbb{R}^r$ and $\mathbf{v}_j \in \mathbb{R}^r$ denote the latent vector for user i and item j respectively. Then, the predicted preference of user i towards item j is formulated as $\hat{r}_{ij} = \mathbf{u}_i^\top \mathbf{v}_j$. To learn all user latent vectors $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_m]^\top \in \mathbb{R}^{m \times r}$ and item latent vectors $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{u}_n]^\top \in \mathbb{R}^{n \times r}$, MF minimizes the following regularized squared loss on the observed ratings:

$$\arg\min_{\mathbf{U},\mathbf{V}} \sum_{(i,j)\in\mathcal{V}} (R_{ij} - \mathbf{u}_i^\top \mathbf{v}_j)^2 + \lambda R(\mathbf{U}, \mathbf{V}), \tag{1}$$

where $\mathcal{V}$ denotes the all observed use-item pairs and $R(\mathbf{U}, \mathbf{V})$ is the regularization term with respect to $\mathbf{U}$ and $\mathbf{V}$ controlled by $\lambda > 0$. To improve recommendation efficiency, after we obtain the optimized user/item real-valued latent vectors, the two-stage hashing method use binary quantization (rounding off [33] or rotate [18, 35]) to convert the continuous latent representations into binary codes. Let us denote $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m]^\top \in \{\pm 1\}^{m \times r}$ and $\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_n]^\top \in \{\pm 1\}^{n \times r}$ respectively as r-length user/item binary codes, then the inner product between the binary codes of user and item can be formulated as $\mathbf{b}_i^\top \mathbf{d}_j = 2H(\mathbf{b}_i, \mathbf{d}_j) - r$, where $H(\cdot)$ denotes the Hamming similarity. Based on fast bit operations, Hamming distance computation is extremely efficient. However, this method usually incurs a large quantization error since the binary bits are obtained by thresholding real values to integers, and thus it cannot preserve the original data geometry in the continuous vector space [30].

2.2 Direct Discrete Optimization Method

To circumvent the above issues, direct discrete optimization method has been proposed in Discrete Collaborative Filtering (DCF) [30] and widely studied in other researches[12, 16, 31]. More formally, it learns the binary codes by optimizing the following objective

function:

$$\arg\min_{\mathbf{B},\mathbf{D}} \sum_{(i,j)\in\mathcal{V}} (R_{ij} - \mathbf{b}_i^\top \mathbf{d}_j)^2 + \lambda R(\mathbf{B},\mathbf{D})$$
$$s.t. \quad \mathbf{B} \in \{\pm 1\}^{m\times r}, \mathbf{D} \in \{\pm 1\}^{n\times r}. \tag{2}$$

This objective function is similar to that of a conventional MF task, except the discrete constraint on the user/item embeddings. By additionally imposing balanced and de-correlated constraints, it could derive compact yet informative binary codes for both users and items.

However, compared to real-valued vectors, CF with binary codes naturally suffers from low accuracy due to limited representation capability in each bit. Specifically, the r-dimensional real-valued vector space have infinite possibility to model users and items, while the number of unique binary codes in Hamming space is 2^r. An alternative way to resolve this issue is to use a longer code. Unfortunately, it will adversely hurt the generalization of the model especially in the sparse scenarios.

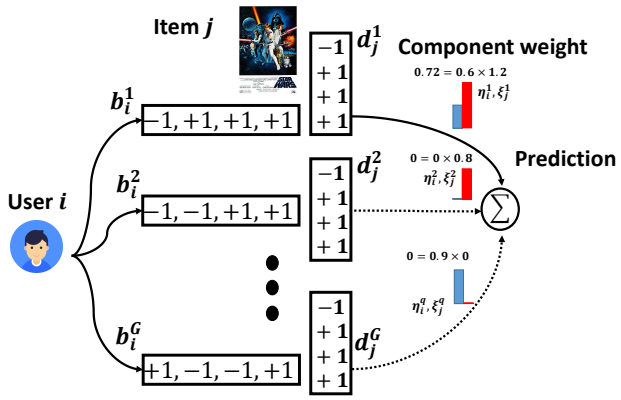


Figure 2: Illustration of how the compositional coding framework makes a prediction for user i given item j.

3 COMPOSITIONAL CODING FOR COLLABORATIVE FILTERING

3.1 Intuition

In this work, we attempt to improve the efficiency of CF without hurting the prediction performance. We propose a new user/item representation named "Compositional Coding", which utilizes both the accuracy of real-valued vectors and the efficiency of binary codes to represent users and items. Since the fixed distance between each pair of binary codes impedes them from modeling different magnitudes of relationships between users and items, we assign real-valued weight to each bit to remarkably enrich their representation capability. Besides, this importance parameters could implicitly prune unimportant bits by setting their importance parameters close to 0, which naturally reduces the computation cost.

3.2 Overview

In general, the proposed compositional coding framework assumes each user or item is represented by G components of r-dimensional binary codes (r is relatively small) and one G-dimensional sparse weight vector. Formally, denote by

$$\mathbf{b}_i^{(1)}, \ldots, \mathbf{b}_i^{(G)} \in \{\pm 1\}^r, \boldsymbol{\eta}_i = \left(\eta_i^{(1)}, \cdots, \eta_i^{(G)}\right) \in \mathbb{R}^G$$

the compositional codes for the i-th user, and

$$\mathbf{d}_j^{(1)}, \ldots, \mathbf{d}_j^{(G)} \in \{\pm 1\}^r, \boldsymbol{\xi}_j = \left(\xi_j^{(1)}, \cdots, \xi_j^{(G)}\right) \in \mathbb{R}^G$$

the compositional codes for the j-th item, respectively. Then the predicted preference of user i for item j is computed by taking the weighted sum of the inner product with respect to each of the G components:

$$\hat{r}_{ij} = \sum_{k=1}^{G} w_{ij}^{(k)} (\mathbf{b}_i^{(k)})^\top \mathbf{d}_j^{(k)}, \tag{3}$$

where $w_{ij}^{(k)} = \eta_i^{(k)} \cdot \xi_j^{(k)}$ is the importance weight of the k-th component of binary codes with respect to user i and item j.

Figure 2 illustrates how to estimate a user-item interaction using compositional codes. The inner product of each component of binary codes $(\mathbf{b}_i^{(k)})^\top \mathbf{d}_j^{(k)}$ can be efficiently computed in Hamming space using the fast bit operations. The importance weight of the k-th component of binary codes $w_{ij}^{(k)}$ is obtained through a multiplication operation over $\eta_i^{(k)}$ and $\xi_j^{(k)}$, which ensures that the importance weight will be assigned a high value if and only if both user and item weights are large and will become zero if either of them is zero. It is worth noting that we use the multiplication instead of addition operation over the user and item weight to achieve the high sparsity of the sparse weight vector, which can lead to a significant reduction of computation cost during online recommendation.

Compared to representing users/items with binary codes, the key advantage of the proposed framework is that the sparse real-valued weight vector substantially increases the representation capacity of user/item embeddings by the compositional coding scheme. Recalling the example shown in Figure 1, we can preserve the movie relations by assigning them with different weight vectors, so as to predict user-item interactions with a lower error. Another benefit is mainly from the idea of compositional matrix approximation [10, 32], which is more suitable for real-world recommender systems, since the interaction matrix is very large and composed of diverse interaction behaviours. Under this view, the proposed compositional coding framework is characterized by multiple components of binary codes. Each component of binary codes can be employed to discover the localized relationships among certain types of similar users and items and thus can be more accurate in this particular region (e.g., young people viewing action movies while old people viewing romance movies). Therefore, the proposed compositional coding framework can encode users and items in a more informative way.

In addition to the improved representation and better accuracy, the compositional coding framework does not lose the high efficiency advantage of binary codes, and can even gain more efficiency when imposing sufficient sparsity. In order to show this, we analyze the time cost of the proposed framework and compare it against the conventional binary coding framework. In particular, assume the time cost of calculating the Hamming distance of r-bit codes is T_h and the time cost of weighted summation of all of inner product

results is T_s, then the total cost for a user-item similarity search is

$$nnz(\mathbf{w}) \times T_h + T_s,$$

where $nnz(\mathbf{w})$ denotes the average number of non-zero values in component weight vector $\mathbf{w} \in \mathbb{R}^G$, which is much smaller than G in our approach due to the sparsity of the user and item weights. Similarly, the cost by conventional hashing based CF models using $(G \times r)$-bit binary codes is $G \times T_h$ which can be higher than ours.

Remark. The proposed compositional coding framework is rather general. When identical weight vectors are used, it is degraded to the binary codes of conventional hashing based CF methods. Compositional codes could also be regarded as real-valued latent vectors of CF model, if we adopt the identical binary codes for each component. In summary, compositional code is a flexible representation which could benefit from both the strong representation capability of real-valued vectors and the fast similarity search of binary codes.

3.3 Formulation

In this section, we instantiate the compositional coding framework on Matrix Factorization (MF) models and term the proposed method as the Compositional Coding for Collaborative Filtering (CCCF). Note that the proposed compositional coding framework can be potentially applicable to other types of CF models beyond MF.

Follow the intuition of compositional coding, we assume that there exists a distance function d that measures distances in the space of users ($i = 1, \ldots, m$) or items ($j = 1, \ldots, n$). The distance function leads to the notion of neighborhoods of user-user and item-item pairs. Its assumption states that the rating of user-item pair (i, j) could be approximated particularly well in its neighbourhood. Thus, we identify G components surrounding K anchor points $(i'_1, j'_1), \ldots, (i'_G, j'_G)$. Follow the setting of compositional matrix approximation [9, 10], user weight $\eta_i^{(k)}$ can be instantiated with an Epanechnikov kernel[1] [25], which is formulated as:

$$\eta_i^{(k)} = \frac{3}{4}\Big(1 - d(i, i'_t)^2\Big)\mathbf{1}\big[d(i, i'_t) < h\big], \tag{4}$$

where $h > 0$ is a bandwidth parameter, $\mathbf{1}[\cdot]$ is the indicator function and $d(\cdot)$ is a distance function to measure the similarity between i and i'_t (we will discuss it in Section 3.6). A large value of h implies that $\boldsymbol{\eta}_i$ has a wide spread, which means most of the user component weights are non-zero. In contrast, a small h corresponds to narrow spread of $\boldsymbol{\eta}_i$ and most of the user components will be zero. Item weight $\xi_j^{(k)}$ follows the analogous formulation. In this way, we could endow the weight vector $w_{ij}^{(k)}$ for user-item pair, which is defined as multiplication over user and item weight, with the sparsity. The method for selecting anchor points for each component is important as it may affect the values of component weights and further affect the generalization performance. A natural way is to uniformly sample them from the training set. In this work, we run k-means clustering with the user/item latent vectors and use the G centroids as anchor points.

To learn the compositional codes, we adopt the squared loss to measure the reconstruction error as the standard MF method. For each set of binary codes, in order to maximize the entropy of each bit and make it as independent as possible, we impose balanced partition and decorrelated constraints. In summary, learning the compositional codes for users and items can be formulated into the following optimization task:

$$\begin{aligned}
\min \sum_{(i,j)\in\mathcal{V}} \Big(R_{ij} - \sum_{k=1}^{G} w_{ij}^{(k)} (\mathbf{b}_i^{(k)})^\top \mathbf{d}_j^{(k)}\Big)^2 \\
\text{s.t.} \underbrace{\mathbf{1}_m \mathbf{B}^{(k)} = 0, \mathbf{1}_n \mathbf{D}^{(k)} = 0}_{\text{balanced partition}}, \underbrace{(\mathbf{B}^{(k)})^\top \mathbf{B}^{(k)} = m\mathbf{I}_r, (\mathbf{D}^{(k)})^\top \mathbf{D}^{(k)} = n\mathbf{I}_r}_{\text{decorrelation}} \\
\mathbf{B}^{(k)} \in \{\pm 1\}^{m\times r},\ \mathbf{D}^{(k)} \in \{\pm 1\}^{n\times r}, \quad k = 1, \ldots, G.
\end{aligned} \tag{5}$$

where we denote $\mathbf{B}^{(k)} = [\mathbf{b}_1^{(k)}, \ldots, \mathbf{b}_m^{(k)}]^\top \in \{\pm 1\}^{m\times r}$ and $\mathbf{D}^{(k)} = [\mathbf{d}_1^{(k)}, \ldots, \mathbf{d}_n^{(k)}]^\top \in \{\pm 1\}^{n\times r}$ respectively as user and item binary codes in the k-th set of binary codes. The problem formulated in (5) is a mixed-binary-integer program which is a challenging task since it is generally NP-hard and involves a combinatorial search over $O(2^{Gr(m+n)})$. An alternative way is to impose auxiliary continuous variables $\mathbf{X}^{(k)} \in \mathcal{B}$ and $\mathbf{Y}^{(k)} \in \mathcal{D}$ for each set of binary codes, where $\mathcal{B}^{(k)} = \{\mathbf{X}^{(k)} \in \mathcal{R}^{m\times r} | \mathbf{1}_m \mathbf{X}^{(k)} = 0, (\mathbf{X}^{(k)})^\top \mathbf{X}^{(k)} = m\mathbf{I}_r\}$ and $\mathcal{D}^{(k)} = \{\mathbf{Y}^{(k)} \in \mathcal{R}^{n\times r} | \mathbf{1}_n \mathbf{Y}^{(k)} = 0, (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = n\mathbf{I}_r\}$. Then the balanced and de-correlated constraints can be softened by $\min_{\mathbf{X}^{(k)}\in\mathcal{B}^{(k)}} \|\mathbf{B}^{(k)} - \mathbf{X}^{(k)}\|_F$ and $\min_{\mathbf{Y}^{(k)}\in\mathcal{D}^{(k)}} \|\mathbf{D}^{(k)} - \mathbf{Y}^{(k)}\|_F$, respectively. Finally, we can solve problem (5) with respect to the k-the set of codes in a computationally tractable manner:

$$\begin{aligned}
\min_{\mathbf{B}^{(k)},\mathbf{D}^{(k)},\mathbf{X}^{(k)},\mathbf{Y}^{(k)}} \sum_{(i,j)\in\mathcal{V}} \Big(R_{ij} - \sum_{k=1}^{G} w_{ij}^{(k)} (\mathbf{b}_i^{(k)})^\top \mathbf{d}_j^{(k)}\Big)^2 \\
+ \alpha_1 \|\mathbf{B}^{(k)} - \mathbf{X}^{(k)}\|_F + \alpha_2 \|\mathbf{D}^{(k)} - \mathbf{Y}^{(k)}\|_F \\
\text{s.t.} \quad \mathbf{B}^{(k)} \in \{\pm 1\}^{m\times r}, \mathbf{D}^{(k)} \in \{\pm 1\}^{n\times r},
\end{aligned} \tag{6}$$

where α_1 and α_2 are tuning parameters. Since $\mathrm{tr}((\mathbf{B}^{(k)})^\top \mathbf{B}^{(k)}) = \mathrm{tr}((\mathbf{X}^{(k)})^\top \mathbf{X}^{(k)}) = mr$, and $\mathrm{tr}((\mathbf{D}^{(k)})^\top \mathbf{D}^{(k)}) = \mathrm{tr}((\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)}) = nr$, the above optimization task can be turned into the following

$$\begin{aligned}
\min_{\mathbf{B}^{(k)},\mathbf{D}^{(k)},\mathbf{X}^{(k)},\mathbf{Y}^{(k)}} \sum_{(i,j)\in\mathcal{V}} (R_{ij} - \sum_{k=1}^{G} w_{ij}^{(k)} (\mathbf{b}_i^{(k)})^\top \mathbf{d}_j^{(k)})^2 \\
- 2\alpha_1 \mathrm{tr}((\mathbf{B}^{(k)})^\top \mathbf{X}^{(k)}) - 2\alpha_2 \mathrm{tr}((\mathbf{D}^{(k)})^\top \mathbf{Y}^{(k)}) \\
\text{s.t.} \quad \mathbf{1}_m \mathbf{X}^{(k)} = 0, \mathbf{1}_n \mathbf{X}^{(k)} = 0, (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = m\mathbf{I}_r, (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = n\mathbf{I}_r \\
\mathbf{B}^{(k)} \in \{\pm 1\}^{m\times r}, \mathbf{D}^{(k)} \in \{\pm 1\}^{n\times r},
\end{aligned} \tag{7}$$

which is the proposed learning model for CCCF. Note that we do not discard the binary constraints but directly optimize the binary codes of each component. Through joint optimization for the binary codes and the auxiliary real-valued variables, we can achieve nearly balanced and un-correlated binary codes. Next, we will introduce an efficient solution for the mixed-integer optimization problem in Eq. (7).

3.4 Optimization

We employ alternative optimization strategy to solve the above problem. Each iteration alternatively updates $\mathbf{B}^{(k)}$, $\mathbf{D}^{(k)}$, $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$. The details are given below.

Learning $\mathbf{B}^{(k)}$ and $\mathbf{D}^{(k)}$: In this subproblem, we update $\mathbf{B}^{(k)}$ with fixed $\mathbf{D}^{(k)}$, $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$. Since the objective function in Eq. (7) is based on summing over users of each component independently, so we can update binary codes for each user and item in

[1] Similar to [9], we also tried uniform and triangular kernel, but the performance was worse than Epanechnikov kernel, in agreement with the theory of kernel smoothing [25].

parallel. Specifically, learning binary codes for user i with respect to component k is to solve the following optimization problem:

$$\min_{\mathbf{b}_i^{(k)}\in\{\pm1\}^r} (\mathbf{b}_i^{(k)})^\top \Big(\sum_{j\in\mathcal{V}_i} (w_{ij}^{(k)})^2 \mathbf{d}_j^{(k)} (\mathbf{d}_j^{(k)})^\top \Big) \mathbf{b}_i^{(k)} - 2\Big(\sum_{j\in\mathcal{V}_i} \tilde{r}_{ij} (\mathbf{d}_j^{(k)})^\top \Big) \mathbf{b}_i^{(k)} - 2\alpha_1 (\mathbf{x}_i^{(k)})^\top \mathbf{b}_i^{(k)}, \tag{8}$$

where $\tilde{r}_{ij} = r_{ij} - \sum_{\tilde{k}\neq k} w_{ij}^{\tilde{k}} (\mathbf{b}_i^{\tilde{k}})^\top \mathbf{d}_j^{\tilde{k}}$ is the residual of observed rating excluding the inner product of component k.

Due to the discrete constraints, the optimization is generally NP-hard, we adopt the bitwise learning method called Discrete Coordinate Descent [21, 22] to update $\mathbf{b}_i^{(k)}$. In particular, denoting $b_{iq}^{(k)}$ as the qth bit of $\mathbf{b}_i^{(k)}$ and $\mathbf{b}_{i\bar{q}}^{(k)}$ as the rest codes excluding $b_{iq}^{(k)}$, DCD update $b_{iq}^{(k)}$ while fixing $\mathbf{b}_{i\bar{q}}^{(k)}$. Thus, the updating rule for user binary code $b_{iq}^{(k)}$ can be formulated as

$$b_{iq}^{(k)} \leftarrow sgn(O(-\hat{b}_{ik}^{(k)}, b_{iq}^{(k)})) \tag{9}$$

where $\hat{b}_{iq}^{(k)} = \sum_{j\in\mathcal{V}_i} (\tilde{r}_{ij} - (w_{ij}^{(k)})^2 (\mathbf{d}_{j\bar{q}}^{(k)})^\top \mathbf{b}_{i\bar{q}}^{(k)}) d_{jq}^{(k)} + \alpha_1 x_{iq}^{(k)}$, $\mathbf{d}_{j\bar{q}}^{(k)}$ is the rest set of item codes excluding d_{jk} and $O(x, y)$ is a function that $O(x, y) = x$ if $x \neq 0$ and $O(x, y) = y$ otherwise. We iteratively update each bit until the procedure convergence. Note that the computational complexity of updating $\mathbf{B}^{(k)}$ is $O\big(\#iter(mnr^2)\big)$ which is a critical efficiency bottleneck when m or n is large. To efficiently compute $\hat{b}_{iq}^{(k)}$, we rewrite $\hat{b}_{iq}^{(k)}$ as

$$\hat{b}_{iq}^{(k)} = \sum_{j\in\mathcal{V}_i} \tilde{r}_{ij} d_{jq}^{(k)} - \sum_{j\in\mathcal{V}_i} (w_{ij}^{(k)})^2 (\mathbf{d}_j^{(k)})^\top \mathbf{b}_i^{(k)} d_{jq}^{(k)} + \sum_{j\in\mathcal{V}_i} b_{iq}^{(k)} + \alpha_1 x_{iq}^{(k)},$$

which reduces the computational cost to $O\big(\#iter(m+n)r^2\big)$.

Similarly, we could learn binary code for item j in component k by solving

$$\min_{\mathbf{d}_j^{(k)}\in\{\pm1\}^r} (\mathbf{d}_j^{(k)})^\top \Big(\sum_{i\in\mathcal{V}_j} (w_{ij}^{(k)})^2 \mathbf{b}_i^{(k)} (\mathbf{b}_i^{(k)})^\top \Big) \mathbf{d}_j^{(k)} - 2\Big(\sum_{i\in\mathcal{V}_j} \tilde{r}_{ij} (\mathbf{b}_i^{(k)})^\top \Big) \mathbf{d}_j^{(k)} - 2\alpha_2 (\mathbf{y}_j^{(k)})^\top \mathbf{d}_j^{(k)}.$$

Denote $d_{jq}^{(k)}$ as the q-th bit of $\mathbf{d}_j^{(k)}$ and $\mathbf{d}_{j\bar{q}}^{(k)}$ as the rest codes excluding $d_{jq}^{(k)}$, we update each bit of $\mathbf{d}_j^{(k)}$ according to

$$d_{jq}^{(k)} \leftarrow sgn(O(-\hat{d}_{jq}^{(k)}, d_{jq}^{(k)})) \tag{10}$$

where $\hat{d}_{jq}^{(k)} = \sum_{i\in\mathcal{V}_j} (\tilde{r}_{ij} - (w_{ij}^{(k)})^2 (\mathbf{b}_{i\bar{q}}^{(k)})^\top \mathbf{d}_{j\bar{q}}^{(k)}) b_{iq}^{(k)} + \alpha_2 y_{jq}^{(k)}$.

Learning $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$: When fixing $\mathbf{B}^{(k)}$, learning $\mathbf{X}^{(k)}$ could be solved via optimizing the following objective function:

$$\max_{\mathbf{X}^{(k)}} tr(\mathbf{B}^{(k)}(\mathbf{X}^{(k)})^\top), \quad \mathbf{1}_m^\top \mathbf{X}^{(k)} = 0, (\mathbf{X}^{(k)})^\top \mathbf{X}^{(k)} = m\mathbf{I}_r.$$

It can be solved by the aid of SVD according to [17]. Let $\bar{\mathbf{B}}^{(k)}$ be a column-wise zero-mean matrix, where $\bar{B}_{ij}^{(k)} = B_{ij}^{(k)} - \frac{1}{m}\sum_i B_{ij}^{(k)}$. Assuming $\bar{\mathbf{B}}^{(k)} = \mathbf{P}_b^{(k)} \Sigma_b^{(k)} (\mathbf{Q}_b^{(k)})^\top$ as its SVD, where each column of $\mathbf{P}_b^{(k)} \in \mathbb{R}^{m\times r'}$ and $\mathbf{Q}_b^{(k)} \in \mathbb{R}^{r\times r'}$ represents the left and right singular vectors corresponding to r' non-zero singular values in the diagonal matrix $\Sigma_b^{(k)}$. Since $\bar{\mathbf{B}}^{(k)}$ and $\mathbf{Q}_b^{(k)}$ have the same row, we have $\mathbf{1}^\top \mathbf{P}^{(k)}{}_b = 0$ due to $\mathbf{1}^\top \bar{\mathbf{B}}^{(k)} = 0$. Then we construct matrices $\hat{\mathbf{P}}_b^{(k)}$ of size $m \times (r - r')$ and $\hat{\mathbf{Q}}_b^{(k)}$ of size $r \times (r - r')$ by employing a Gram-Schmidt process such that $(\hat{\mathbf{P}}^{(k)})_b^\top \hat{\mathbf{P}}_b^{(k)} = \mathbf{I}_{r-r'}$, $[\mathbf{P}_b^{(k)}\ \mathbf{1}]^\top \hat{\mathbf{P}}_b^{(k)} = 0$, and $(\hat{\mathbf{Q}}_b^{(k)})^\top \hat{\mathbf{Q}}_b = \mathbf{I}_{r-r'}$, $[\mathbf{Q}_b^{(k)}\ \mathbf{1}]^\top \hat{\mathbf{Q}}_b^{(k)} = 0$. Now we obtain a closed-form update rule for $\mathbf{X}^{(k)}$:

$$\mathbf{X}^{(k)} \leftarrow \sqrt{m}[\mathbf{P}_b^{(k)},\ \hat{\mathbf{P}}_b^{(k)}][\mathbf{Q}_b^{(k)},\ \hat{\mathbf{Q}}_b^{(k)}]^\top. \tag{11}$$

In practice, to compute such an optimal $\mathbf{X}^{(k)}$, we perform the eigendecomposition over the small $r \times r$ matrix

$$(\bar{\mathbf{B}}^{(k)})^\top \bar{\mathbf{B}}^{(k)} = [\mathbf{Q}_b^{(k)}\ \hat{\mathbf{Q}}_b^{(k)}] \begin{bmatrix} (\Sigma^{(k)})^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} [\mathbf{Q}_b^{(k)}\ \hat{\mathbf{Q}}_b^{(k)}]^\top,$$

which provides $\mathbf{Q}_b^{(k)}, \hat{\mathbf{Q}}_b^{(k)}, \Sigma^{(k)}$, and we can obtain

$$\mathbf{P}_b^{(k)} = \bar{\mathbf{B}}^{(k)} \mathbf{Q}_b^{(k)} (\Sigma^{(k)})^{-1}.$$

Then matrix $\hat{\mathbf{P}}_b^{(k)}$ can be obtained by the aforementioned Gram-Schmidt orthogonalization. Note that it requires $O\big(r^2 m\big)$ to perform SVD, Gram-Schimdt orthogonalization and matrix multiplication.

When $\mathbf{D}^{(k)}$ fixed, learning $\mathbf{Y}^{(k)}$ could be solved in a similar way:

$$\max_{\mathbf{Y}^{(k)}} tr(\mathbf{D}^{(k)}(\mathbf{Y}^{(k)})^\top), \quad \mathbf{1}_n^\top \mathbf{Y}^{(k)} = 0, (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = n\mathbf{I}_r.$$

We can obtain an analytic solution:

$$\mathbf{Y}^{(k)} \leftarrow \sqrt{n}[\mathbf{P}_d^{(k)},\ \hat{\mathbf{P}}_d^{(k)}][\mathbf{Q}_d^{(k)},\ \hat{\mathbf{Q}}_d^{(k)}]^\top. \tag{12}$$

where each column of $\mathbf{P}_d^{(k)}$ and $\mathbf{Q}_d^{(k)}$ is the left and right singular vectors of $\bar{\mathbf{D}}^{(k)}$ respectively. $\hat{\mathbf{Q}}_d^{(k)}$ are the left singular vectors corresponding to zero singular values of the $r \times r$ matrix $(\bar{\mathbf{D}}^{(k)})^\top \bar{\mathbf{D}}^{(k)}$, and $\hat{\mathbf{P}}_d^{(k)}$ are the vectors obtained via the Gram-Schimdt process. We summarize the solution for CCCF in Algorithm 1.

Algorithm 1 The proposed algorithm for Compositional Coding for Collaborative Filtering (CCCF) .

Input: $\mathbf{R} \in \mathbb{R}^{m\times n}$
Output: $\mathbf{B}^{(k)} \in \{\pm1\}^{r\times m}, \mathbf{D}^{(k)} \in \{\pm1\}^{r\times n}$
Parameters: number of components G, code length r, regularization coefficient α_1, α_2, bandwidth parameter h
Initialize $\mathbf{B}^{(k)}, \mathbf{D}^{(k)}$ and $\mathbf{X}^{(k)}, \mathbf{Y}^{(k)} \in \mathbb{R}^{m\times n}$ by Eq. (13).
while not converged **do**
  for $k = 1, \cdots, G$ **parallel do**
    Pick anchor points (i'_k, j'_k).
    for $u = 1, \cdots, m$ **do**
      Update $\mathbf{b}_i^{(k)}$ according to (9)
    end for
    for $i = 1, \cdots, n$ **do**
      Update $\mathbf{d}_j^{(k)}$ according to (10).
    end for
    Update $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$ according to (11) and (12).
  end for
end while

3.5 Initialization

Note that the proposed optimization problem involves a mixed-integer non-convex problem, the initialization of model parameters plays an important role for fast convergence and for finding better local optimum solution. To achieve a good initialization in an efficient way, we essentially relax the binary constraints in Eq. (7) into the following optimization:

$$\min_{\mathbf{U}^{(k)},\mathbf{V}^{(k)},\mathbf{X}^{(k)},\mathbf{Y}^{(k)}} \sum_{(i,j)\in\mathcal{V}} \Big(R_{ij} - \sum_{k=1}^{G} w_{ij}^{(k)} (\mathbf{u}_i^{(k)})^\top \mathbf{v}_j^{(k)}\Big)^2 \tag{13}$$
$$-2\alpha_1 \mathrm{tr}((\mathbf{B}^{(k)})^\top \mathbf{X}^{(k)}) - 2\alpha_2 \mathrm{tr}((\mathbf{V}^{(k)})^\top \mathbf{Y}^{(k)}) + \alpha_3 \|\mathbf{U}^{(k)}\|_F^2 + \alpha_4 \|\mathbf{V}^{(k)}\|_F^2$$
$$\text{s.t. } \mathbf{1}_m \mathbf{X}^{(k)} = 0,\ \mathbf{1}_n \mathbf{X}^{(k)} = 0,\ (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = m\mathbf{I}_r,\ (\mathbf{Y}^{(k)})^\top \mathbf{Y}^{(k)} = n\mathbf{I}_r,$$

We first initialize real-valued matrix $\mathbf{U}^{(k)}$ and $\mathbf{V}^{(k)}$ randomly and find the feasible solution for $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$ according to the above learning method with respect to $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$. Then the alternating optimization are conducted by updating $\mathbf{U}$ and $\mathbf{V}$ with traditional gradient descent method and updating $\mathbf{X}$ and $\mathbf{Y}$ with respect to the similar learning method. Once we obtain the solution $(\mathbf{U}_0^{(k)}, \mathbf{V}_0^{(k)}, \mathbf{X}_0^{(k)}, \mathbf{Y}_0^{(k)})$, we can initialize CCCF with respect to component k as:

$$\mathbf{B}^{(k)} \leftarrow \mathrm{sgn}(\mathbf{U}_0^{(k)}), \mathbf{D}^{(k)} \leftarrow \mathrm{sgn}(\mathbf{V}_0^{(k)}), \mathbf{X}^{(k)} \leftarrow \mathbf{X}_0^{(k)}, \mathbf{Y}^{(k)} \leftarrow \mathbf{Y}_0^{(k)}. \tag{14}$$

The effectiveness of the proposed initialization will be discussed in Section 4.1 (illustrated in Figure 14).

3.6 Distance Function

Previously we assume a general distance function d, which is defined to measure the distance between users or items so as to compute the component weights $w_i^{(k)}$ and $v_j^{(k)}$ in Eq. (4). The metric can be constructed with side information, like users' social link [28, 34] or using metric learning techniques [29]. However, many datasets do not include such data. In this work, we follow the idea of [9, 10] which factorizes the observed interaction matrix using MF and obtain two latent representation matrices $\mathbf{U}$ and $\mathbf{V}$ for users and items, respectively. Then the distance between two users can be computed by the cosine distance between the obtained latent representations, which is formulated as $d(u_i, u_j) = arccos\Big(\frac{\langle \mathbf{u}_i, \mathbf{u}_j \rangle}{\|\mathbf{u}_i\| \cdot \|\mathbf{u}_j\|}\Big)$. The distance between two items can be computed in the same way.

3.7 Complexity

The computational complexity of training CCCF is K times the complexity of learning each set of binary codes. It converges quickly in practice, which usually takes about $4 \sim 5$ iterations in our experiments. For each iteration, the computational cost for updating $\mathbf{B}^{(k)}$ and $\mathbf{D}^{(k)}$ is $O\big(\#iter(m+n)r^2\big)$. In practice, $\#iter$ is usually $2 \sim 5$. The computational cost for updating $\mathbf{X}^{(k)}$ and $\mathbf{Y}^{(k)}$ is $O\big(r^2 m\big)$ and $O\big(r^2 n\big)$, respectively. Suppose the entire algorithm requires T iterations for convergence, the overall time complexity for Algorithm 1 is $O(Tqr^2(m+n))$, where we found T empirically is no more than 5. In summary, CCCF is efficient and scalable because it scales linearly with the number of users and items.

3.8 Fast Retrieval via Integer Weight Scaling

Floating-point operations over user and item weight vectors invoke more CPU cycles and are usually much slower than integer computation. The cost of top-k recommendation would be remarkably lower when the scalars in the weight vectors are integers instead of floating numbers. An intuitive way is to adopt integer approximation via rounding the scalars in user and item weight vectors. However, if the weights are too small, it will incur large deviation. To tackle this problem, we scale the original scalars by multiplying each weight by e and then approximate them with integers in preprocessing,

$$\hat{\boldsymbol{\eta}}_i = \Big\{\lfloor e \cdot \eta_i^{(1)} \rceil, \ldots, \lfloor e \cdot \eta_i^{(G)} \rceil\Big\}, \quad \hat{\boldsymbol{\xi}}_j = \Big\{\lfloor e \cdot \xi_j^{(1)} \rceil, \ldots, \lfloor e \cdot \xi_j^{(G)} \rceil\Big\},$$

where $\lfloor e \cdot \eta_i^{(k)} \rceil$ is a round function to obtain an integer approximation with respect to $e \cdot \eta_i^{(k)}$.

4 EXPERIMENTS

In order to validate the effectiveness and efficacy of the proposed CCCF method for recommender systems, we conduct an extensive set of experiments to examine different aspects of our method in comparison to state-of-the-art methods based on conventional binary coding. We aim to answer the following questions:

- **RQ1:** How does CCCF perform as compared to other state-of-the-arts hashing based recommendation methods in terms of both accuracy and retrieval time?
- **RQ2:** How do different hyper-parameter settings (e.g., number of components, and code length) affect the accuracy of CCCF ?
- **RQ3:** How do the sparsity of component weight vectors (controlled by the bandwidth parameter h) and integer scaling (controlled by parameter e) affect both the accuracy and retrieval cost of CCCF? How to choose optimal values ?
- **RQ4:** Does the representation of compositional codes in CCCF enjoy a much stronger representation capability than the traditional binary codes in DCF given the same model size ?

4.1 Experimental Settings

4.1.1 ***Datasets and Settings.*** We run our experiments on three public datasets: Movielens 1M[2], Amazon and Yelp[3] which are widely used in the literature. All of these ratings range from 0 to 5. Considering the severe sparsity of Yelp and Amazon original datasets, we followed the conventional filtering strategy [20] by removing users and items with less than 10 ratings. The statistics of the filtered datasets are shown in Table 1. For each user, we randomly sampled 70% ratings as training data and the rest 30% for test. We repeated for 5 random splits and reported the averaged results.

Dataset	#Ratings	#Items	#Users	#density
Movielens 1M	1,000,209	3900	6040	4.2%
Yelp	696,865	25,677	25,815	0.11%
Amazon	5,057,936	146,469	189,474	0.02%

Table 1: Summary of datasets in our experiments.

[2] http://grouplens.org/datasets/movielens
[3] http://www.yelp.com/dataset challenge

4.1.2 ***Parameter Settings and Performance Metrics.*** For CCCF, we vary the number of components and the code length of each component in range $\{4, 8, 12, 16\}$. The hyper-parameters α and β are tuned within $\{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$. Grid search is performed to choose the best parameters on the training split. We evaluate our proposed algorithms by Normalized Discounted Cumulative Gain (NDCG) [6], which is probably the most popular ranking metric for capturing the importance of retrieving good items at the top of ranked lists. The average NDCG at cut off [2, 4, 6, 8, 10] over all users is the final metric. A higher NDCG@K reflects a better accuracy of recommendation performance.

4.1.3 ***Baseline Methods and Implementations.*** To validate the effectiveness of CCCF, we compare it with several state-of-the-art real-valued CF methods and hashing-based CF methods:

- **MF**: This is the classic Matrix Factorization based CF algorithm [8], which learns real-valued user and item latent vectors in Euclidean space.
- **BCCF**: This is a two-stage binarized CF method [35] with a relaxation stage and a quantization stage. At these two stages, it successively solves MF with balanced code regularization and applies orthogonal rotation to obtain user codes and item codes.
- **DCF**: This is the first method [30] directly tackles a discrete optimization problem for seeking informative and compact binary codes for users and items.
- **DCMF**: This is the state-of-the-art binarized method [12] for CF with side information. It extends DCF by encoding the side features as the constraints for user codes and item codes.

The CCCF algorithm empirically converges very fast and using the initialization generally helps as shown in Figure 3.

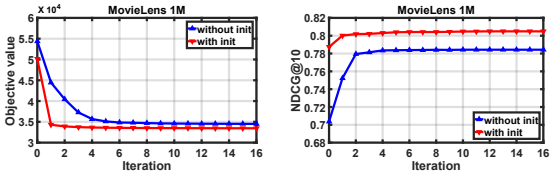


Figure 3: Convergence of the overall objective values and NDCG@10 of CCCF with/without initialization on the Movielens 1M dataset. The use of the proposed initialization leads to faster convergence and better results.

4.2 Experimental Results

4.2.1 **Comparisons with State-of-the-arts (RQ1).** Figure 4 shows the results of top-k recommendation with k setting from 2 to 10. Note that the number of components G is fixed to 8 and the code length of each component varies in $\{4, 6, 8, 10, 12, 14, 16\}$. For fair comparison, the code length of DCF and the rank of MF are equal to the total bits of CCCF, which are equivalent to the summation of each component's code length of CCCF (rG), so that the performance gain is not from increasing model complexity. From Figure 4, we can draw the following major observations:

First of all, we observe that CCCF considerably outperforms BCCF, DCF and DCMF which are the state-of-the-art hashing based CF methods. The performance of CCCF and DCF continiously increase as we increase the code length. Surprisingly, CCCF can even achieve remarkable superior performance using only 32 bits in comparison to the DCF using 128 bits on three datasets. This improvement indicates the impressive effectiveness of learning compositional codes.

Second, between baseline methods, DCF consistently outperforms BCCF, while slightly underperforms DCMF. This is consistent with the findings in [30] that the performance of direct discrete optimization could surpass that of the two-stage methods. Besides, side information makes user codes and item codes more representative, which improves the recommendation performance.

Moreover, it is worth mentioning that CCCF outperforms MF, which is a real-valued CF method, particularly on the Amazon and Yelp dataset. The reasons for this are two-fold. First, compositional structure of CCCF has a much stronger representation capability which could discover complex relationships among users and items. Second, the higher sparsity of the dataset makes MF easy to overfit, whereas the binarized and sparse parameters in CCCF could alleviate this issue. This finding again demonstrates the effectiveness of our method.

Dataset	CCCF	MF		DCF	
	Time	Time	Speedup	Time	Speedup
Movielens 1M	7.12	49.66	×6.97	10.64	×1.49
Amazon	831.34	5917.56	×7.12	1231.35	×1.48
Yelp	184.48	1450.25	×7.86	264.65	×1.43

Table 2: Retrieval time (in seconds) of recommendation methods on three datasets, 'Speedup' indicates the speedup (×) of CCCF ($G = 8, r = 16$) over baselines.

Finally, Table 2 shows the total time cost (in seconds) taken by each method to generate the top-k item list of all items. Overall, the hashing based methods (DCF and CCCF) outperform real-valued methods (MF), indicating the great advantage of binarizing the real-valued parameters. Moreover, CCCF shows superior retrieval time compared to DCF while enjoying a better accuracy. Thus, CCCF is a suitable model for large-scale recommender systems where the retrieval time is restricted within a limited time quota.

4.2.2 **Impact of Hyper-parameter G and r (RQ2).** Our CCCF has two key parameters, the number of components G and code length r, to control the complexity and capacity of CCCF. Figure 5(a) and 5(b) evaluate how they affect the recommendation performance under varied total bits and fixed total bits. In Figure 5(a), we vary the code length r from 4 to 16 and the component number G from 1 to 16. We can see that increasing G leads to continued improvements. When G is larger than 8, the improvement tends to become saturated as the number of components increases. In addition, a larger value of G would lead to relatively longer training time. Similar observations can be found from the results of hyper-parameter r evaluation. In Figure 5(b), we fix the total bits rG in range $\{32, 64, 96, 128\}$ and varies the component number G from 1 to 32. It should be noted that when $G = 1$, CCCF model is identical to DCF model. As we gradually increase component number G, the recommendation performance grows since the real-valued component weight could enhance the representation capability. The best recommendation performance is achieved when $G = 8$ or 16. When G is larger than the optimal values, increasing G will hurt the performance. The main reason is that we fix the total bits rG, so

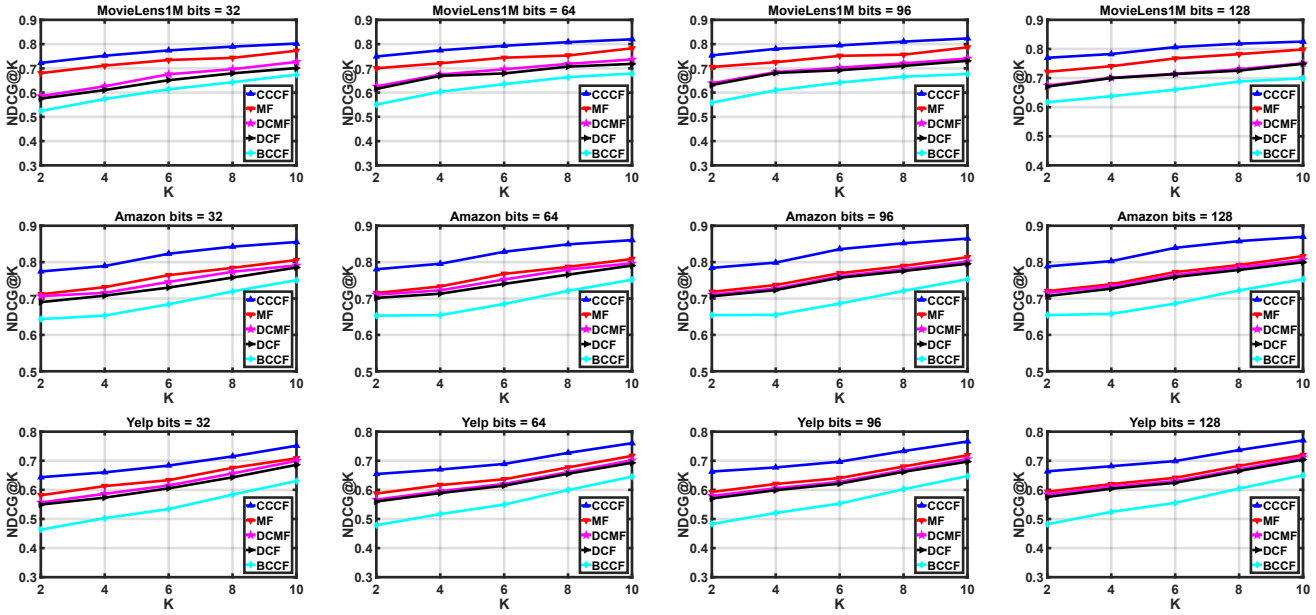
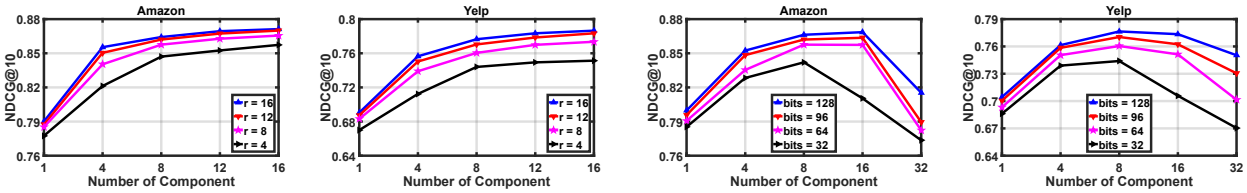


Figure 4: Item recommendation performance comparison of NDCG@K with respect to code length in bits.



(a) **Varied total bits (rG)** (b) **Fixed total bits (rG)**

Figure 5: Performance of CCCF with respect to code length in bits (r) and number of components (G).

that larger values of G lead to smaller values of r. This will reduce the learning space of CCCF since the component weight is calculated by a predefined distance function which does not consider the rating information.

4.2.3 **Impact of sparsity of component weight and integer weight scaling (RQ3).** To demonstrate the effectiveness of the integer weight scaling in accuracy and retrieval time , we run two versions of CCCF: EXACT and IWS. EXACT does not adopt the proposed integer weight scaling strategy. IWS uses the integer weight scaling described in Section 3.8. Figure 6 summarizes the speedup of the two versions of CCCF. The retrieval cost is not sensitive to the integer scaling parameter e and we set e to 100. We can see the gap between the versions is consistent over Amazon and Yelp dataset. The low cost of IWS validates the effectiveness of the integer scaling which replaces the floating-point operations with the fast integer computation.

Figure 7 shows the impact of hyper-parameter e on the accuracy of CCCF. We can find that when e is smaller than 100, the increase of e leads to gradual improvements. When e is larger than 100, further increasing its value cannot improve the performance. This indicates that IWS is relatively insensitive when e is sufficiently large. We thus suggest to set e to 100.

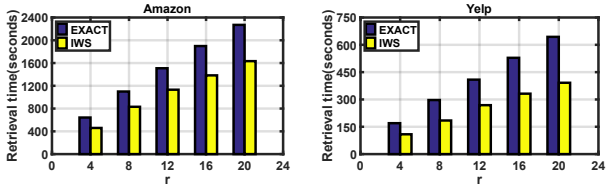


Figure 6: Retrieval cost of the naive version and IWS version.

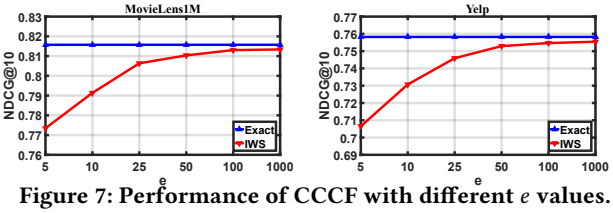


Figure 7: Performance of CCCF with different e values.

To reveal the impact of sparsity of component weight in accuracy and retrieval cost, we vary the bandwidth hyperparameter h from 0.5 to 1. It is obvious that decreasing the value of h will increasing the sparsity of component weights. Figure 8 shows the accuracy and retrieval cost of CCCF for different h. First, we can see that the

retrieval cost of CCCF is continuously reduced as we decrease the values of h since high sparsity lead to fast computation. Second, we observe that the best recomendation performance is achieved when $h = 0.7 \sim 0.8$. When h is smaller than $0.7 \sim 0.8$, increasing the sparsity will make CCCF robust to overfitting. However, when the sparsity level is quite high, the CCF model might not be informative enough for prediction and thus suffer performance degradation.

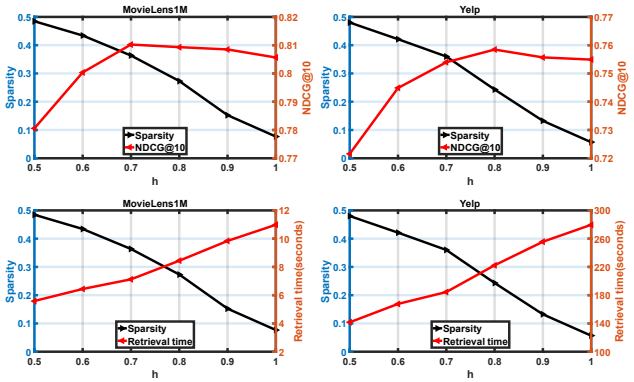


Figure 8: Item recommendation performance and retrieval cost of CCCF with different h values (controlling the sparsity of user/item component weights).

4.2.4 **Item Embeddings Visualization (RQ4).** The key advantage of compositional codes is the stronger representational capability in comparison to binary codes. Therefore we visualize the learned item embeddings of the movielens 1M dataset where items are indicated as movies. We use the item representations learned by DCF and CCCF as the input to the visualization tool t-SNE [19]. As a result, each movie is mapped into a two-dimensional vector. Then we can visualize each item embedding as a point on a two dimensional space. For items which are labelled as different genres, we adopt different colors on the corresponding points. Thus, a good visualization result is that the points of the same color are closer to each other. The visualization result is shown in Figure 9. We can find that the result of DCF is unsatisfactory since the points belonging to different categories are mixed with each other. For CCCF, we can observed clear clusters of different categories. This again validates the advantage of much stronger representation power of the compositional codes over traditional binary codes.

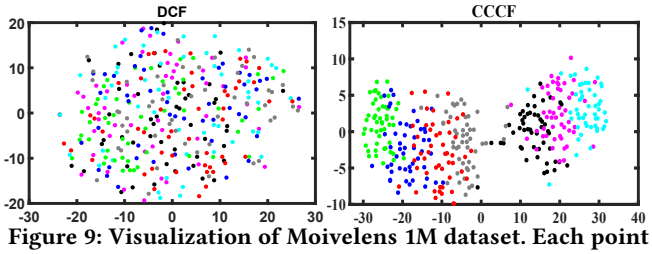


Figure 9: Visualization of Moivelens 1M dataset. Each point indicates one item embedding (movie). The color of a point indicates the genre of the movie.

5 RELATED WORK

As a pioneer work, Locality-Sensitive Hashing has been adopted for generating hash codes for Google News readers based on their click history [4]. Following this work, random projection was applied for mapping learned user/item embeddings from matrix factorization into the Hamming space to obtain binary codes for users and items [7]. Similar to the idea of projection, Zhou et al. [35] generated binary codes from rotated continuous user/item representations by running Iterative Quantization. In order to derive more compact binary codes, the de-correlated constraint over different binary codes was imposed on user/item continuous representations and then rounded them to produce binary codes [18]. The relevant work could be summarized as two independent stages: relaxed learning of user/item representations with some specific constraints and subsequent bianry quantization. However, such two-stage approaches suffer from a large quantization loss according to [30], so direct optimization of matrix factorization with discrete constraints was proposed. To derive compact yet informative binary codes, the balanced and de-correlated constraints were further imposed [30]. In order to incorporate content information from users and items, content-aware matrix factorization and factorization machine with binary constraints was further proposed [12, 16]. For dealing with social information, a discrete social recommendation model was proposed in [15].

Recently, the idea of compositional codes has been explored in the compression of feature embedding [1, 23, 24], which has become more and more important in order to deploy large models to small mobile devices. In general, they composed the embedding vectors using a small set of basis vectors. The selection of basis vectors was governed by the hash code of the original symbols. In this way, compositional coding approaches could maximize the storage efficiency by eliminating the redundancy inherent in representing similar symbols with independent embeddings. In contrast, this work employs compositional codes to address the inner product similarity search problem in recommender systems.

6 CONCLUSION AND FUTURE WORK

This work contributes a novel and much more effective framework called Compositional Coding for Collaborative Filtering (CCCF). The idea is to represent each user/item by multiple components of binary codes together with a sparse weight vector, where each element of the weight vector encodes the importance of the corresponding component of binary codes to the user/item. In contrast to standard binary codes, compositional codes significantly enriches the representation capability without sacrificing retrieval efficiency. To this end, CCCF can enjoy both the merits of effectiveness and efficiency in recommendation. Extensive experiments demonstrate that CCCF not only outperforms existing hashing-based binary code learning algorithms in terms of recommendation accuracy, but also achieves considerable speedup of retrieval efficiency over the state-of-the-art binary coding approaches. In future, we will apply compositional coding framework to other recommendation models, especially for the more generic feature-based models like Factorization Machines. In addition, we are interested in employing CCCF on the recently developed neural CF models to further advance the performance of item recommendation.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation Singapore under its AI Singapore Programme [AISG-RP-2018-001]. Xin Wang is supported by China Postdoctoral Science Foundation No. BX201700136.

REFERENCES

[1] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. *arXiv preprint arXiv:1806.09464* (2018).

[2] Zhiyong Cheng, Ying Ding, Lei Zhu, and Mohan Kankanhalli. 2018. Aspect-aware latent factor model: Rating prediction with ratings and reviews. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 639–648.

[3] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan S Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation.. In *IJCAI*, Vol. 17. 3654–3660.

[4] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 271–280.

[5] Johan Håstad. 2001. Some optimal inapproximability results. *Journal of the ACM (JACM)* 48, 4 (2001), 798–859.

[6] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 41–48.

[7] Alexandros Karatzoglou, Alex Smola, and Markus Weimer. 2010. Collaborative filtering on a budget. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 389–396.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[9] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2014. Local collaborative ranking. In *Proceedings of the 23rd international conference on World wide web*. ACM, 85–96.

[10] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2013. Local low-rank matrix approximation. In *International Conference on Machine Learning*. 82–90.

[11] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1419–1428.

[12] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 325–334.

[13] Chenghao Liu, Steven CH Hoi, Peilin Zhao, Jianling Sun, and Ee-Peng Lim. 2016. Online adaptive passive-aggressive methods for non-negative matrix factorization and its applications. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 1161–1170.

[14] Chenghao Liu, Tao Jin, Steven CH Hoi, Peilin Zhao, and Jianling Sun. 2017. Collaborative topic regression for online recommender systems: an online and Bayesian approach. *Machine Learning* 106, 5 (2017), 651–670.

[15] Chenghao Liu, Xin Wang, Tao Lu, Wenwu Zhu, Jianling Sun, and Steven CH Hoi. 2019. Discrete Social Recommendation. In *Thirty-Third AAAI Conference on Artificial Intelligence*.

[16] Han Liu, Xiangnan He, Fuli Feng, Liqiang Nie, Rui Liu, and Hanwang Zhang. 2018. Discrete Factorization Machines for Fast Feature-based Recommendation. *arXiv preprint arXiv:1805.02232* (2018).

[17] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *Advances in Neural Information Processing Systems*. 3419–3427.

[18] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. 2014. Collaborative hashing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2139–2146.

[19] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[20] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[21] Fumin Shen, Yadong Mu, Yang Yang, Wei Liu, Li Liu, Jingkuan Song, and Heng Tao Shen. 2017. Classification by Retrieval: Binarizing Data and Classifier. (2017).

[22] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 37–45.

[23] Raphael Shu and Hideki Nakayama. 2017. Compressing Word Embeddings via Deep Compositional Code Learning. *arXiv preprint arXiv:1711.01068* (2017).

[24] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. 2017. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems*. 4928–4936.

[25] Matt P Wand and M Chris Jones. 1994. *Kernel smoothing*. Chapman and Hall/CRC.

[26] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2012. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 12 (2012), 2393–2406.

[27] Xin Wang, Steven CH Hoi, Chenghao Liu, and Martin Ester. 2017. Interactive social recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 357–366.

[28] Xin Wang, Wei Lu, Martin Ester, Can Wang, and Chun Chen. 2016. Social recommendation with strong and weak ties. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 5–14.

[29] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. 2003. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*. 521–528.

[30] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 325–334.

[31] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback.. In *AAAI*. 1669–1675.

[32] Yongfeng Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. 2013. Improve collaborative filtering through bordered block diagonal form matrices. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 313–322.

[33] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 183–192.

[34] Huan Zhao, Quanming Yao, James T Kwok, and Dik Lun Lee. 2017. Collaborative Filtering with Social Local Models. In *Proceedings of the 16th International Conference on Data Mining*. 645–654.

[35] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 498–506.