Discrete Content-aware Matrix Factorization

Defu Lian
School of Computer Science and Engineering and Big Data Research Center, University of Electronic Science and Technology of China
dove.ustc@gmail.com

Rui Liu
Big Data Research Center, University of Electronic Science and Technology of China
ruiliu011@gmail.com

Yong Ge
Eller School of Management, University of Arizona
yongge@email.arizona.edu

Kai Zheng
University of Electronic Science and Technology of China
kevinzheng1983@gmail.com

Xing Xie
Microsoft Research
xingx@microsoft.com

Longbing Cao
University of Technology Sydney
longbing.cao@uts.edu.au

ABSTRACT

Precisely recommending relevant items from massive candidates to a large number of users is an indispensable yet computationally expensive task in many online platforms (e.g., Amazon.com and Netflix.com). A promising way is to project users and items into a Hamming space and then recommend items via Hamming distance. However, previous studies didn't address the cold-start challenges and couldn't make the best use of preference data like implicit feedback. To fill this gap, we propose a Discrete Content-aware Matrix Factorization (DCMF) model, 1) to derive compact yet informative binary codes at the presence of user/item content information; 2) to support the classification task based on a local upper bound of logit loss; 3) to introduce an interaction regularization for dealing with the sparsity issue. We further develop an efficient discrete optimization algorithm for parameter learning. Based on extensive experiments on three real-world datasets, we show that DCFM outperforms the state-of-the-arts on both regression and classification tasks.

CCS CONCEPTS

•Information systems →Collaborative filtering;

KEYWORDS

Recommendation, Discrete Hashing, Collaborative Filtering, Content-based Filtering

ACM Reference format:
Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of KDD'17, August 13–17, 2017, Halifax, NS, Canada.,* , 10 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098008

KDD'17, August 13–17, 2017, Halifax, NS, Canada.

DOI: http://dx.doi.org/10.1145/3097983.3098008

1 INTRODUCTION

Recommender systems aim to recommend relevant items (e.g., products and news) to users via mining and understanding their preferences for items. Thanks to the development of recommendation techniques over the past decades, they have been widely used in various web services such as Amazon.com and eBay.com for improving sale of products and increasing click-through rate of advertisements. However, within most of these web services, the number of customers and products is dramatically growing, making recommendation more challenging than ever before. For example, there are more than 300 million active Amazon customers and over 480 million products for sale till now. Consequently, it is challenging to generate immediate response to find out potentially-preferred products for customers via analyzing large-scale yet sparse browsing, purchasing and searching data.

In the past, a variety of recommendation approaches have been proposed, which include content-based methods, collaborative filtering algorithms, and the combination of both kinds [1]. Among these recommendation algorithms, dimension reduction techniques exemplified by matrix factorization demonstrate not only high effectiveness but also the best sole-model performance [1]. Matrix factorization algorithms factorize an $M \times N$ user-item rating matrix to map both users and items into a D-dimensional latent space, where one user's preference for an item is modeled by the inner product between their latent features. When content information of users and items is available, content-aware matrix factorization (CaMF) algorithms have been introduced [3, 16, 23], where extracted features from content information are also mapped into the same latent space. Such incorporation of content information usually leads to better recommendation performance [16]. In CaMF methods, users' preference for items could be still modeled as the inner product between the latent features of users and items [16]. The computational complexity for generating top-K preferred items for all users is $O(MND + MN \log K)$. Therefore, CaMF methods are often computationally expensive and lead to crucial low-efficiency issues when either M or N is large.

Upon the independence of generating top-K preferred items for different users, one way to solve the aforementioned challenge is to distribute the computation with parallel/distributed computing techniques [40]. Another promising solution for improving the efficiency is to encode real-valued latent factors with compact binary

codes because the inner product between binary codes could be computed much more efficiently via bit operations. Furthermore, by indexing all items with special data structures, approximately querying top-K preferred items with user binary codes has logarithmic or even constant time complexity [21, 29]. The extensive efficiency study has been presented in [37]. However, learning compact binary codes is generally NP-hard [8] due to the discretization constraints. To tackle this problem, people resort to a two-stage procedure [19, 37, 39]. This procedure first solves a relaxed optimization algorithm via discarding the discrete constraints, and then performs direct binary quantization. However, according to [35], such a two-stage procedure results in a large quantization loss due to oversimplification. Consequently, a learning based framework was proposed for direct discrete optimization [35]. In spite of the advantages of such a framework, there are two important limitations. First, the content information of users and items is not taken into account, thus cold-start problems could not be well addressed. Second, they cannot make good use of preference data like binary feedback or implicit feedback (e.g., click data), which is often more prevalent in many recommendation scenarios.

To address these limitations, we propose Discrete Content-aware Matrix Factorization (DCMF) to learn hash codes for users and items at the presence of content information (such as user's age and gender, item's category and textual content). Without imposing discretization constraints on latent factors of content information, our framework requires the minimal number of discretization constraints. By additionally imposing balanced and de-correlated constraints, DCMF could derive compact yet informative binary codes for both users and items. Besides supporting the regression task, this framework can also handle the classification task based on a local variational bound of logistic regression when taking binary feedback or implicit feedback as input. In order to make better use of implicit feedback, an interaction regularizer is further introduced to address the sparsity challenge in implicit feedback. To solve the tractable discrete optimization of DCMF with all the challenging constraints, we develop an efficient alternating optimization method which essentially solves the mixed-integer programming subproblems iteratively. With extensive experiments on three real-world datasets, we show that DCMF outperforms the state-of-the-arts for both classification and regression tasks and verify the effectiveness of item content information, the logit loss for classification tasks, and the interaction regularization for sparse preference data.

To summarize, our contributions include:

- We study how to hash users and items at the presence of their respective content information for fast recommendation in both regression and classification tasks.
- We develop an efficient discrete optimization algorithm for tackling discretization, balanced and de-correlated constraints as well as interaction regularization.
- Through extensive experiments on three public datasets, we show the superiority of the proposed algorithm to the state-of-the-arts.

2 PRELIMINARIES

Matrix factorization operates on a user-item rating/preference matrix $\boldsymbol{R}$ of size $M \times N$, where M and N is the number of users and items, respectively. Each entry r_{ij} indicates rating/preference of a user i for an item j (using preference for subsequent presentation). All observed entries are denoted by $\Omega = \{(i,j)|r_{ij} \text{ is known}\}$. The set of items for which a user i have preference is $\mathbb{I}_i$ of size $N_i = |\mathbb{I}_i|$ and the set of users having preference for an item j is $\mathbb{U}_j$ of size $M_j = |\mathbb{U}_j|$. $\text{sgn}(\cdot) : \mathbb{R} \to \{\pm 1\}$ is a sign function. Below, upper case bold letters denote matrices, lower case bold letters denote column vectors, and non-bold letters represent scalars.

2.1 Content-aware Matrix Factorization

Matrix factorization maps both users and items onto a joint D-dimensional latent space ($D \ll \min(M,N)$), where each user is represented by $\tilde{\boldsymbol{p}}_i \in \mathbb{R}^D$ and each item is represented by $\tilde{\boldsymbol{q}}_j \in \mathbb{R}^D$. Thus, each user's predicted preference for each item is estimated by inner product. At the presence of their respective content information, encoded as $\boldsymbol{x}_i \in \mathbb{R}^{M\times F}$ and $\boldsymbol{y}_j \in \mathbb{R}^{N\times L}$ respectively, content-aware matrix factorization additionally maps their respective features into the same latent space via a matrix $\boldsymbol{U} \in \mathbb{R}^{F\times D}$ and a matrix $\boldsymbol{V} \in \mathbb{R}^{L\times D}$ respectively. Then each user (item) is represented by $\boldsymbol{p}_i = \tilde{\boldsymbol{p}}_i + \boldsymbol{U}'\boldsymbol{x}_i$ ($\boldsymbol{q}_j = \tilde{\boldsymbol{q}}_j + \boldsymbol{V}'\boldsymbol{y}_j$) according to [3, 4]. In this case, the predicted preference of user i for item j is $\hat{r}_{ij} = \boldsymbol{p}_i'\boldsymbol{q}_j = (\tilde{\boldsymbol{p}}_i + \boldsymbol{U}'\boldsymbol{x}_i)'(\tilde{\boldsymbol{q}}_j + \boldsymbol{V}'\boldsymbol{y}_j)$. Thus different from factorization machines [23], we don't take interaction between user (item) id and user (item) feature, and interaction between different user (item) features into account. However, such a prediction formula makes it possible to more easily leverage hash techniques for speeding up recommendation. To learn $\{\boldsymbol{p}_i\}$, $\{\boldsymbol{q}_j\}$, $\boldsymbol{U}$ and $\boldsymbol{V}$, we can minimize the following objective function [16]:

$$\sum_{(i,j)\in\Omega} \ell(r_{ij}, \boldsymbol{p}_i'\boldsymbol{q}_j) + \lambda_1 \sum_i \|\boldsymbol{p}_i - \boldsymbol{U}'\boldsymbol{x}_i\|^2 + \gamma_1\|\boldsymbol{U}\|_F^2 + \lambda_2 \sum_j \|\boldsymbol{q}_j - \boldsymbol{V}'\boldsymbol{y}_j\|^2 + \gamma_2\|\boldsymbol{V}\|_F^2 \quad (1)$$

where $\ell(r_{ij}, \boldsymbol{p}_i'\boldsymbol{q}_j)$ is a convex loss, like square loss $\ell(r_{ij}, \boldsymbol{p}_i'\boldsymbol{q}_j) = (r_{ij} - \boldsymbol{p}_i'\boldsymbol{q}_j)^2$ for rating prediction, logistic loss $\ell(r_{ij}, \boldsymbol{p}_i'\boldsymbol{q}_j) = \log(1 + e^{-r_{ij}\boldsymbol{p}_i'\boldsymbol{q}_j})$ for the classification task from binary feedback $r \in \{-1, 1\}$. When taking implicit feedback as input, an interaction regularization $\sum_{i,j}(\boldsymbol{p}_i'\boldsymbol{q}_j - 0)^2$, penalizing non-zero predicted preference, should be imposed for better recommendation performance. This is because the state-of-the-art objective function [10, 15, 17, 34] for implicit feedback could be decomposed into a Ω-dependent part and an interaction regularization. In particular, assuming $w_{ij} = \alpha + 1$ if $(i,j) \in \Omega$ and $w_{ij} = 1$ otherwise, then we have

$$\begin{aligned}
&\sum_{i,j} w_{ij}(r_{ij} - \boldsymbol{p}_i\boldsymbol{q}_j)^2 \\
=&(\alpha+1)\sum_{(i,j)\in\Omega}(r_{ij} - \boldsymbol{p}_i\boldsymbol{q}_j)^2 + \sum_{(i,j)\notin\Omega}(0 - \boldsymbol{p}_i'\boldsymbol{q}_j)^2 \\
=&\alpha\sum_{(i,j)\in\Omega}(\frac{\alpha+1}{\alpha}r_{ij} - \boldsymbol{p}_i'\boldsymbol{q}_j)^2 + \sum_{i,j}(0 - \boldsymbol{p}_i'\boldsymbol{q}_j)^2 - \frac{\alpha+1}{\alpha}r_{ij}^2 \\
\approx&\alpha\sum_{(i,j)\in\Omega}(r_{ij} - \boldsymbol{p}_i'\boldsymbol{q}_j)^2 + \sum_{i,j}(0 - \boldsymbol{p}_i'\boldsymbol{q}_j)^2 - r_{ij}^2
\end{aligned}$$

where the last approximation is derived since α is usually significantly larger than 1.

3 DISCRETE CONTENT-AWARE MATRIX FACTORIZATION

After obtaining latent representation for users and items, generating top-K preferred items for each user is considered as a similarity-based retrieval problem. In particular, treating a user's latent factor as a query, one can compute "similarity" score between user and item via inner product, and then extract the top-K preferred items through the max-heap data structure. However, such a similarity-based retrieval scheme, costing $O(ND + N \log K)$, leads to crucial low-efficiency issues for real practice when N is large.

If representing them by binary codes, this similarity-based search could be accelerated by more efficiently computing the inner product via the Hamming distance. Denoting $\mathbf{\Phi} = [\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_M]' \in \{-1, 1\}^{M \times D}$ and $\mathbf{\Psi} = [\boldsymbol{\psi}_1, \cdots, \boldsymbol{\psi}_N]' \in \{-1, 1\}^{N \times D}$ as user and item binary code, respectively, the inner product is represented as $\boldsymbol{\phi}_i' \boldsymbol{\psi}_j = 2\mathrm{H}(\boldsymbol{\phi}_i, \boldsymbol{\psi}_j) - D$, where $\mathrm{H}(\boldsymbol{\phi}, \boldsymbol{\psi})$ denotes the Hamming distance between binary codes. Based on fast bit operations, Hamming distance computation is extremely efficient. If only conducting approximated similarity-based search, it has logarithmic or even constant time complexity based on advanced indexing techniques [21, 29]. For extensive efficiency study of hashing-based recommendation could be referred in [37]. Below, we investigate how to directly learn binary codes for users and items.

3.1 Loss Function

To derive the loss function, the first action is to convert continuous latent factors into binary ones. In order to assure that each bit carries as much information as possible, balanced constraints should be imposed on binary codes of users and items [39]. To make binary codes compact so that each bit should be as independent as possible, de-correlated constraints are also imposed on them. And they may lead to using shorter code for encoding more information [33]. Therefore, learning binary codes for users and items is to minimize the following objective function.

$$\begin{aligned} &\sum_{(i,j)\in\Omega} \ell(r_{ij}, \boldsymbol{\phi}_i' \boldsymbol{\psi}_j) + \lambda_1 \|\mathbf{\Phi} - \boldsymbol{X}\boldsymbol{U}\|_F^2 + \gamma_1 \|\boldsymbol{U}\|_F^2 \\ &+\lambda_2 \|\mathbf{\Psi} - \boldsymbol{Y}\boldsymbol{V}\|_F^2 + \gamma_2 \|\boldsymbol{V}\|_F^2 + \beta \sum_{ij} (\boldsymbol{\phi}_i' \boldsymbol{\psi}_j)^2 \\ &\text{s.t. } \mathbf{\Phi} \in \{-1, 1\}^{M \times D}, \mathbf{\Psi} \in \{-1, 1\}^{N \times D}, \\ &\underbrace{\mathbf{1}_M' \mathbf{\Phi} = 0, \mathbf{1}_N' \mathbf{\Psi} = 0}_{\text{balance}}, \underbrace{\mathbf{\Phi}'\mathbf{\Phi} = M\boldsymbol{I}_D, \mathbf{\Psi}'\mathbf{\Psi} = N\boldsymbol{I}_D}_{\text{de-correlation}} \end{aligned} \tag{2}$$

However, optimizing this objective function is a challenging task and it is generally NP-hard due to involving combinatorial optimization over space of size $(M + N)D$. Accordingly, we introduce an optimization framework that can minimize this objective function in a computationally tractable way by softening the balanced and de-correlated constraints. In particular, we can introduce a delegate continuous variable $\boldsymbol{P} \in \mathcal{P}$ for $\mathbf{\Phi}$ and a delegate continuous variable $\boldsymbol{Q} \in \mathcal{Q}$ for $\mathbf{\Psi}$, where $\mathcal{P} = \{\boldsymbol{P} \in \mathbb{R}^{M \times D} | \mathbf{1}_M' \boldsymbol{P} = 0, \boldsymbol{P}'\boldsymbol{P} = M\boldsymbol{I}_D\}$ and $\mathcal{Q} = \{\boldsymbol{Q} \in \mathbb{R}^{N \times D} | \mathbf{1}_N' \boldsymbol{Q} = 0, \boldsymbol{Q}'\boldsymbol{Q} = N\boldsymbol{I}_D\}$. Then the balanced and de-correlated constraints on users and items could be softened by $\min_{\boldsymbol{P} \in \mathcal{P}} \|\mathbf{\Phi} - \boldsymbol{P}\|_F$ and $\min_{\boldsymbol{Q} \in \mathcal{Q}} \|\mathbf{\Psi} - \boldsymbol{Q}\|_F$, respectively. It becomes a three-objective minimization problem. Applying scalarization techniques of multi-objective optimization problems, we formulate the tractable objective function of DCMF as follows:

$$\begin{aligned} &\sum_{(i,j)\in\Omega} \ell(r_{ij}, \boldsymbol{\phi}_i' \boldsymbol{\psi}_j) + \alpha_1 \|\mathbf{\Phi} - \boldsymbol{P}\|_F^2 + \lambda_1 \|\mathbf{\Phi} - \boldsymbol{X}\boldsymbol{U}\|_F^2 + \gamma_1 \|\boldsymbol{U}\|_F^2 \\ &+\alpha_2 \|\mathbf{\Psi} - \boldsymbol{Q}\|_F^2 + \lambda_2 \|\mathbf{\Psi} - \boldsymbol{Y}\boldsymbol{V}\|_F^2 + \gamma_2 \|\boldsymbol{V}\|_F^2 + \beta \sum_{ij} (\boldsymbol{\phi}_i' \boldsymbol{\psi}_j)^2 \\ &\text{s.t. } \mathbf{1}_M' \boldsymbol{P} = 0, \boldsymbol{P}'\boldsymbol{P} = M\boldsymbol{I}_D, \mathbf{1}_N' \boldsymbol{Q} = 0, \boldsymbol{Q}'\boldsymbol{Q} = N\boldsymbol{I}_D \\ &\mathbf{\Phi} \in \{-1, 1\}^{M \times D}, \mathbf{\Psi} \in \{-1, 1\}^{N \times D} \end{aligned} \tag{3}$$

where α_1 and α_2 are tuning parameters. If there are feasible solutions in Eq (2), using very large values of α_1 and α_2 will enforce $\mathbf{\Phi} = \boldsymbol{P}$ and $\mathbf{\Psi} = \boldsymbol{Q}$, turning Eq (3) to Eq (2). The comparative small values of α_1 and α_2 allow a certain discrepancy between $\mathbf{\Phi}$ and $\boldsymbol{P}$, between $\mathbf{\Psi}$ and $\boldsymbol{Q}$, making Eq (3) more flexible. Through jointly optimizing binary codes and delegate real variables, we can get nearly balanced and de-correlated hash codes for users and items.

Making use of $\mathrm{tr}(\mathbf{\Phi}'\mathbf{\Phi}) = \mathrm{tr}(\boldsymbol{P}'\boldsymbol{P}) = MD$ and $\mathrm{tr}(\mathbf{\Psi}'\mathbf{\Psi}) = \mathrm{tr}(\boldsymbol{Q}'\boldsymbol{Q}) = ND$, Eq (3) is equivalent to

$$\begin{aligned} &\sum_{(i,j)\in\Omega} \ell(r_{ij}, \boldsymbol{\phi}_i' \boldsymbol{\psi}_j) + \beta \sum_{ij} (\boldsymbol{\phi}_i' \boldsymbol{\psi}_j)^2 - 2\mathrm{tr}\big(\mathbf{\Phi}'(\alpha_1 \boldsymbol{P} + \lambda_1 \boldsymbol{X}\boldsymbol{U})\big) \\ &+\lambda_1 \mathrm{tr}\big(\boldsymbol{U}'(\boldsymbol{X}'\boldsymbol{X} + \frac{\gamma_1}{\lambda_1}\boldsymbol{I}_F)\boldsymbol{U}\big) - 2\mathrm{tr}\big(\mathbf{\Psi}'(\alpha_2 \boldsymbol{Q} + \lambda_2 \boldsymbol{Y}\boldsymbol{V})\big) \\ &+\lambda_2 \mathrm{tr}\big(\boldsymbol{V}'(\boldsymbol{Y}'\boldsymbol{Y} + \frac{\gamma_2}{\lambda_2}\boldsymbol{I}_L)\boldsymbol{V}\big) \\ &\text{s.t. } \mathbf{1}_M' \boldsymbol{P} = 0, \boldsymbol{P}'\boldsymbol{P} = M\boldsymbol{I}_D, \mathbf{1}_N' \boldsymbol{Q} = 0, \boldsymbol{Q}'\boldsymbol{Q} = N\boldsymbol{I}_D \\ &\mathbf{\Phi} \in \{-1, 1\}^{M \times D}, \mathbf{\Psi} \in \{-1, 1\}^{N \times D} \end{aligned} \tag{4}$$

Note that we do not discard the discretization constraints but instead directly optimize discrete $\mathbf{\Phi}$ and $\mathbf{\Psi}$. It is worth mentioning that the norm of binary codes of both users and items are constant and don't take any effect of regularization, but the interaction regularization of binary codes between each user and each item is meaningful. Next, we develop an efficient learning solution for such a mixed-integer optimization problem.

3.2 Optimization

Generally speaking, alternating optimization is used, taking turns in updating each of $\mathbf{\Phi}, \mathbf{\Psi}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{U}$ and $\boldsymbol{V}$, given others fixed. Although the objective function in Eq (4) depends on the choice of loss function, it only affects the updating rules of $\mathbf{\Phi}$ and $\mathbf{\Psi}$ since their update seeks binary latent representation to preserve the user-item intrinsic preference.

3.2.1 Learning $\mathbf{\Phi}$ and $\mathbf{\Psi}$. We consider both regression and classification tasks for learning hash codes and thus elaborate the derivation for their updating rules, respectively.

- **Regression**: Ignoring the term irrelevant to $\mathbf{\Phi}$ and $\mathbf{\Psi}$, the loss function is

$$\begin{aligned} &\sum_{(i,j)\in\Omega} (r_{ij} - \boldsymbol{\phi}_i' \boldsymbol{\psi}_j)^2 - 2\mathrm{tr}\big(\mathbf{\Phi}'(\alpha_1 \boldsymbol{P} + \lambda_1 \boldsymbol{X}\boldsymbol{U})\big) \\ &- 2\mathrm{tr}\big(\mathbf{\Psi}'(\alpha_2 \boldsymbol{Q} + \lambda_2 \boldsymbol{Y}\boldsymbol{V})\big) + \beta \sum_{i,j} \boldsymbol{\phi}_i' \boldsymbol{\psi}_j \boldsymbol{\psi}_i' \boldsymbol{\phi}_j, \end{aligned} \tag{5}$$

where summation is conducted over users independently, so we can update hash code for each user in parallel. In particular, learning hash code for a user i is to solve the following optimization problem:

$$\min_{\boldsymbol{\phi}_i \in \{\pm 1\}^D} \boldsymbol{\phi}_i'(\sum_{j \in \mathbb{I}_i} \boldsymbol{\psi}_j \boldsymbol{\psi}_j' + \beta \boldsymbol{\Psi}'\boldsymbol{\Psi})\boldsymbol{\phi}_i - 2\boldsymbol{\phi}_i'(\sum_{j \in \mathbb{I}_i} r_{ij}\boldsymbol{\psi}_j + \alpha_1 \boldsymbol{p}_i + \lambda_1 \boldsymbol{U}'\boldsymbol{x}_i) \quad (6)$$

Due to the discrete constraints, such an optimization problem is generally NP-hard, we develop a coordinate-descent algorithm to take turns to update each bit of the hash code $\boldsymbol{\phi}_i$ when other bits fixed. In particular, assuming the d^{th} bit is represented by ϕ_{id} and the remaining codes by $\boldsymbol{\phi}_{i\bar{d}}$, the coordinate-descent algorithm solves the following objective function:

$$\min_{\phi_{id} \in \{\pm 1\}} \phi_{id}(\boldsymbol{\phi}_{i\bar{d}}' \sum_{j \in \mathbb{I}_i} \psi_{jd}\boldsymbol{\psi}_{j\bar{d}} + \beta \boldsymbol{\phi}_{i\bar{d}}' \boldsymbol{\Psi}_{\bar{d}}' \boldsymbol{\psi}_d - \sum_{j \in \mathbb{I}_i} r_{ij}\psi_{jd} - \alpha_1 p_{id} - \lambda_1 \boldsymbol{u}_d' \boldsymbol{x}_i),$$

where $\boldsymbol{\psi}_{j\bar{d}}$ is the remaining set of item codes excluding ψ_{jd}, $\boldsymbol{\psi}_d$ is the d^{th} column of the matrix $\boldsymbol{\Psi}$ while $\boldsymbol{\Psi}_{\bar{d}}$ excludes the d^{th} column from $\boldsymbol{\Psi}$, $\boldsymbol{u}_d$ is the d^{th} column of the matrix $\boldsymbol{U}$. Thus we update ϕ_{id} based on the following rule:

$$\phi_{id}^* = \mathrm{sgn}\big(K(\hat{\phi}_{id}, \phi_{id})\big), \quad (7)$$

where $\hat{\phi}_{id} = \sum_{j \in \mathbb{I}_i}(r_{ij} - \hat{r}_{ij} + \phi_{id}\psi_{jd})\psi_{jd} + \alpha_1 p_{id} + \lambda_1 \boldsymbol{u}_d' \boldsymbol{x}_i - \beta \boldsymbol{\phi}_i' \boldsymbol{\Psi}' \boldsymbol{\psi}_d + \beta N \phi_{id}$ while $\hat{r}_{ij} = \boldsymbol{\phi}_i' \boldsymbol{\psi}_j$ is prediction of preference, and $K(x, y)$ equals to x if x≠0 and y otherwise, meaning that we don't make an update if $\hat{\phi}_{id} = 0$. The update rule is applied among bits iteratively until convergence. Denoting the number of the bit-wise iteration as $\#iter$. Note that when the preference prediction of observed entries is cached, $\hat{r}_{ij}$ could be dynamically updated, i.e., $\hat{r}_{ij}^* = \hat{r}_{ij} + (\phi_{id}^* - \phi_{id})\psi_{jd}$. The third term except $\boldsymbol{\phi}_i$ for all bits could be pre-computed before the user-level iteration, costing $O(ND^2)$. Therefore, excluding overhead of pre-computation, the complexity of updating the hash code for the user i is $O\big(\#iter(D^2 + N_i D)\big)$, indicating that making update for all users in sequence costs $O\big(\#iter(MD^2 + |\Omega|D) + ND^2\big)$. If not considering interaction regularization, the complexity could be reduced to $O(\#iter|\Omega|D)$. When leveraging parallel computation, its complexity could decrease by a factor of the number of threads and (or) processes.

Similarly, we can learn hash code for an item j by solving

$$\min_{\boldsymbol{\psi}_j \in \{\pm 1\}^D} \boldsymbol{\psi}_j'(\sum_{i \in \mathbb{U}_j} \boldsymbol{\phi}_i \boldsymbol{\phi}_i' + \beta \boldsymbol{\Phi}'\boldsymbol{\Phi})\boldsymbol{\psi}_j - 2\boldsymbol{\psi}_j'(\sum_{i \in \mathbb{U}_j} r_{ij}\boldsymbol{\phi}_i + \alpha_2 \boldsymbol{q}_j + \lambda_2 \boldsymbol{V}'\boldsymbol{y}_j)$$

Based on the coordinate-descent algorithm, we update $\boldsymbol{\psi}_j$ according to:

$$\psi_{jd}^* = \mathrm{sgn}(K(\hat{\psi}_{jd}, \psi_{jd})), \quad (8)$$

where $\hat{\psi}_{jd} = \sum_{i \in \mathbb{U}_j}(r_{ij} - \hat{r}_{ij} + \phi_{id}\psi_{jd})\phi_{id} + \alpha_2 q_{jd} + \lambda_2 \boldsymbol{v}_d' \boldsymbol{y}_j - \beta \boldsymbol{\psi}_j' \boldsymbol{\Phi}' \boldsymbol{\phi}_d + \beta M \psi_{jd}$. Following the same analysis, the complexity is $O\big(\#iter(MD^2 + |\Omega|D) + ND^2\big)$ when updating all items in sequence.

- **Classification** We only consider the logistic loss $\ell(r_{ij}, \boldsymbol{\phi}_i'\boldsymbol{\psi}_j) = \log(1 + e^{-r_{ij}\boldsymbol{\phi}_i'\boldsymbol{\psi}_j})$ due to its wide use in practice [3, 13]. However, due to the non-linearity of this loss function, it is impossible to directly obtain the close form of updating rule for hash codes of users and items even based on the coordinate descent algorithm. Therefore, we seek its upper variational yet quadratic bound [11] and optimize the variational variable. In particular,

$$\begin{aligned} &\log(1 + e^{-r_{ij}\boldsymbol{\phi}_i'\boldsymbol{\psi}_j}) \\ =&\log(1 + e^{\boldsymbol{\phi}_i'\boldsymbol{\psi}_j}) - \frac{1 + r_{ij}}{2}\boldsymbol{\phi}_i'\boldsymbol{\psi}_j \\ \le&\lambda(\hat{r}_{ij})\big((\boldsymbol{\phi}_i'\boldsymbol{\psi}_j)^2 - \hat{r}_{ij}^2\big) - \frac{1}{2}(r_{ij}\boldsymbol{\phi}_i'\boldsymbol{\psi}_j + \hat{r}_{ij}) + \log(1 + e^{\hat{r}_{ij}}) \end{aligned} \quad (9)$$

where $\lambda(x) = \frac{1}{4x}\tanh(x/2) = \frac{1}{2x}(\sigma(x) - \frac{1}{2})$ and the equality holds only if $\hat{r}_{ij} = \boldsymbol{\phi}_i'\boldsymbol{\psi}_j$. Using this upper bound, learning hash code for user i is to solve

$$\min_{\boldsymbol{\phi}_i \in \{\pm 1\}^D} \boldsymbol{\phi}_i'(\sum_{j \in \mathbb{I}_i} \lambda(\hat{r}_{ij})\boldsymbol{\psi}_j\boldsymbol{\psi}_j' + \beta\boldsymbol{\Psi}'\boldsymbol{\Psi})\boldsymbol{\phi}_i - \frac{1}{2}\boldsymbol{\phi}_i'\sum_{j \in \mathbb{I}_i} r_{ij}\boldsymbol{\psi}_j - 2\boldsymbol{\phi}_i'(\alpha_1 \boldsymbol{p}_i + \lambda_1 \boldsymbol{U}'\boldsymbol{x}_i), \quad (10)$$

where $\hat{r}_{ij}$ is the preference prediction based on the current value of $\boldsymbol{\phi}_i$ and $\boldsymbol{\psi}_j$. Through deviation, we can still apply Eq (7) for updating the d^{th} bit ϕ_{id} but the first term of $\hat{\phi}_{id}$ should be $\sum_{j \in \mathbb{I}_i}\big(\frac{1}{4}r_{ij} - \lambda(\hat{r}_{ij})\hat{r}_{ij} + \lambda(\hat{r}_{ij})\phi_{id}\psi_{jd}\big)\psi_{jd}$. Similarly, we can update hash codes for items according to Eq (8) after adjusting the first term of $\hat{\psi}_{id}$. The complexity of the updating rule remains the same as before.

3.2.2 Learning $\boldsymbol{P}$ and $\boldsymbol{Q}$. When fixing $\boldsymbol{\Phi}$, learning $\boldsymbol{P}$ could be solved via optimizing the following objective function:

$$\max_{\boldsymbol{P}} \mathrm{tr}(\boldsymbol{P}'\boldsymbol{\Phi}), \text{ s.t. } \boldsymbol{1}_M'\boldsymbol{P} = 0 \text{ and } \boldsymbol{P}'\boldsymbol{P} = M\boldsymbol{I}_D. \quad (11)$$

An analytical solution can be obtained with the aid of a centering matrix $\boldsymbol{J}_n = \boldsymbol{I}_n - \frac{1}{n}\boldsymbol{1}_n\boldsymbol{1}_n'$. In particular, assume $\boldsymbol{J}_M\boldsymbol{\Phi} = \boldsymbol{S}_P\boldsymbol{\Sigma}_P\boldsymbol{T}_P'$ as its Singular Value Decomposition, where each column of $\boldsymbol{S}_P \in \mathbb{R}^{M \times \tilde{D}}$ and $\boldsymbol{T}_P \in \mathbb{R}^{D \times \tilde{D}}$ contains the left- and right-singular vectors respectively, corresponding to $\tilde{D}$ non-zero singular values in the diagonal matrix $\boldsymbol{\Sigma}_P$. Here, for generic purpose, the $\boldsymbol{\Phi}$ is not assumed full column rank, i.e., $\tilde{D} \le D$. Note that $\boldsymbol{1}_M'\boldsymbol{J}_M = 0$, so $\boldsymbol{1}_M'\boldsymbol{J}_M\boldsymbol{\Phi} = 0$, implying $\boldsymbol{1}_M'\boldsymbol{S}_P = 0$. Then we construct matrices $\hat{\boldsymbol{S}}_P$ of size $M \times (D - \tilde{D})$ and $\hat{\boldsymbol{T}}_P$ of size $D \times (D - \tilde{D})$ by using a Gram-Schmidt orthogonalization process such that $\hat{\boldsymbol{S}}_P'[\boldsymbol{S}_P, \hat{\boldsymbol{S}}_P, \boldsymbol{1}_M] = 0$ and $\hat{\boldsymbol{T}}_P'[\boldsymbol{T}_P, \hat{\boldsymbol{T}}_P] = 0$. Then the analytical solution for updating $\boldsymbol{P}$ is determined as follows according to [18].

$$\boldsymbol{P} = \sqrt{M}[\boldsymbol{S}_P, \hat{\boldsymbol{S}}_P][\boldsymbol{T}_P, \hat{\boldsymbol{T}}_P]'. \quad (12)$$

In practice, to compute such an analytical solution, we could first conduct eigendecomposition on the matrix $\boldsymbol{\Phi}'\boldsymbol{J}_M\boldsymbol{\Phi}$ of size $D \times D$ to obtain $\boldsymbol{T}_P$ and $\hat{\boldsymbol{T}}_P$, where each column of $\hat{\boldsymbol{T}}_P$ corresponds to eigenvectors of zero eigenvalues. This only costs $O(D^3)$. Then, we can obtain $\boldsymbol{S}_P = \boldsymbol{J}_M\boldsymbol{\Phi}\boldsymbol{T}_P\boldsymbol{\Sigma}_P^{-1}$ based on matrix multiplication of $O(MD\tilde{D})$ complexity. And $\hat{\boldsymbol{S}}_P$ could be initialized to a random

matrix followed by the aforementioned Gram-Schmidt orthogonalization process, costing $O(M(D^2 - \tilde{D}^2))$. Therefore, the overall complexity of updating $\boldsymbol{P}$ is $O(MD^2)$.

When fixing $\boldsymbol{\Psi}$, learning $\boldsymbol{Q}$ could be solved in a similar way:

$$\max_{\boldsymbol{Q}} \operatorname{tr}(\boldsymbol{Q}'\boldsymbol{\Psi}), \text{ s.t. } \mathbf{1}_N'\boldsymbol{Q} = 0 \text{ and } \boldsymbol{Q}'\boldsymbol{Q} = N\boldsymbol{I}_D \tag{13}$$

Its analytical solution is

$$\boldsymbol{Q} = \sqrt{N}[\boldsymbol{S}_Q, \hat{\boldsymbol{S}}_Q][\boldsymbol{T}_Q, \hat{\boldsymbol{T}}_Q]', \tag{14}$$

where each column of $\boldsymbol{S}_Q$ and $\boldsymbol{T}_Q$ contains the left- and right-singular vectors of $\boldsymbol{J}_N\boldsymbol{\Psi}$ respectively. And the construction of $\hat{\boldsymbol{S}}_Q$ and $\hat{\boldsymbol{T}}_Q$ is the same as that of $\hat{\boldsymbol{S}}_P$ and $\hat{\boldsymbol{T}}_P$ via the Gram-Schmidt process. Its overall complexity is $O(ND^2)$.

3.2.3 Learning $\boldsymbol{U}$ and $\boldsymbol{V}$. When fixing $\boldsymbol{P}$ and $\boldsymbol{Q}$, taking all terms related to $\boldsymbol{U}$ and $\boldsymbol{V}$, the optimization problem with respect to $\boldsymbol{U}$ and $\boldsymbol{V}$ is then respectively formulated as:

$$\begin{aligned} &\min_{\boldsymbol{U}} \operatorname{tr}\big(\boldsymbol{U}'(\boldsymbol{X}'\boldsymbol{X} + \frac{\gamma_1}{\lambda_1}\boldsymbol{I}_F)\boldsymbol{U}\big) - 2\operatorname{tr}(\boldsymbol{\Phi}'\boldsymbol{X}\boldsymbol{U}) \\ &\min_{\boldsymbol{V}} \operatorname{tr}\big(\boldsymbol{V}'(\boldsymbol{Y}'\boldsymbol{Y} + \frac{\gamma_2}{\lambda_2}\boldsymbol{I}_L)\boldsymbol{U}\big) - 2\operatorname{tr}(\boldsymbol{\Psi}'\boldsymbol{Y}\boldsymbol{V}), \end{aligned} \tag{15}$$

The optimal solution for them is derived as:

$$\begin{aligned} \boldsymbol{U} &= (\boldsymbol{X}'\boldsymbol{X} + \frac{\gamma_1}{\lambda_1}\boldsymbol{I}_F)^{-1}\boldsymbol{X}'\boldsymbol{\Phi} \\ \boldsymbol{V} &= (\boldsymbol{Y}'\boldsymbol{Y} + \frac{\gamma_2}{\lambda_2}\boldsymbol{I}_L)^{-1}\boldsymbol{Y}'\boldsymbol{\Psi}. \end{aligned} \tag{16}$$

When the number of features is large, conjugate gradient descent could be applied. The time complexity only depends on the multiplication between matrices, costing $O\big((\|\boldsymbol{X}\|_0 + \|\boldsymbol{Y}\|_0)D\#iter\big)$, where $\|\cdot\|_0$ is the ℓ_0 norm of matrices, equaling to the number of non-zeros entries in the matrices, and #iter is the number of iterations of conjugate gradient descent to reach a given threshold of approximation error.

3.3 Learning Hashing Codes in Cold-Start Case

When items have no rating history in the training set but are associated with content information, it remains unknown to derive their hash codes. Accordingly, denoting feature representation, hash codes and delegate continuous variables of these N_1 items as $\boldsymbol{X}_1$, $\boldsymbol{\Psi}_1$ and $\boldsymbol{Q}_1$, we can derive $\boldsymbol{\Psi}_1$ by solving:

$$\begin{aligned} &\min_{\boldsymbol{\Psi}_1, \boldsymbol{Q}_1} -\operatorname{tr}\big(\boldsymbol{\Psi}_1'(\lambda_2\boldsymbol{X}_1\boldsymbol{U} + \alpha_2\boldsymbol{Q}_1)\big) \\ &\text{s.t. } \boldsymbol{Q}_1'\boldsymbol{Q}_1 = N_1\boldsymbol{I}_D \text{ and } \mathbf{1}_{N_1}'\boldsymbol{Q} = 0. \end{aligned} \tag{17}$$

Resorting to the similar optimization techniques, we can easily obtain the hash codes of the cold-start items. The hash codes of cold-start users can be derived similarly.

3.4 Initialization

Note that the optimization problem involves a mixed-integer non-convex optimization, a better initialization strategy could be important for faster convergence and finding better local optimum. The solutions of two-stage learning schemes are promising. In particular, we first solve a relaxed optimization problem in Eq (3) by discarding the discretization constraints and apply binary quantization for obtaining hash codes for users and items. Note that such an objective function has imposed balanced and de-correlated constraints on latent factors of users and items, leading to a small quantization error according to [35].

To solve the relaxed optimization problem, we can also leverage alternating optimization for parameter learning. The update rules for all parameters are the same except $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ because of discarding the discretization constraints. In particular, learning latent factors for a user i in case of the regression task is achieved by solving

$$\begin{aligned} \min_{\boldsymbol{\phi}_i \in \mathbb{R}^D} \boldsymbol{\phi}_i'(\sum_{j \in \mathbb{I}_i} \boldsymbol{\psi}_j\boldsymbol{\psi}_j' + \beta\boldsymbol{\Psi}'\boldsymbol{\Psi} + \alpha_1\boldsymbol{I}_D)\boldsymbol{\phi}_i \\ - 2\boldsymbol{\phi}_i'(\sum_{j \in \mathbb{I}_i} r_{ij}\boldsymbol{\psi}_j + \alpha_1\boldsymbol{p}_i + \lambda_1\boldsymbol{U}'\boldsymbol{x}_i) \end{aligned}$$

Thanks to its quadratics with respect to $\boldsymbol{\phi}_i$, there is a closed form of the updating rule. However, such a closed form costs $O(|\Omega|D^2 + MD^3)$ for updating latent factors of all users in sequence due to its existence of interaction regularization. Therefore, coordinate descent algorithm could be more appealing. In this case, the closed form for updating ϕ_{id} is

$$\begin{aligned} \phi_{id}^* = \frac{\sum_{j \in \mathbb{I}_i}(r_{ij} - \hat{r}_{ij} + \phi_{id}\psi_{jd})\psi_{jd} - \beta\boldsymbol{\phi}_i'\boldsymbol{\Psi}'\boldsymbol{\psi}_d}{\sum_{j \in \mathbb{I}_i}\psi_{jd}^2 + \beta\boldsymbol{\psi}_d'\boldsymbol{\psi}_d + \alpha_1} \\ + \frac{\beta\phi_{id}\boldsymbol{\psi}_d'\boldsymbol{\psi}_d + \alpha_1 p_{id} + \lambda_1\boldsymbol{u}_d'\boldsymbol{x}_i}{\sum_{j \in \mathbb{I}_i}\psi_{jd}^2 + \beta\boldsymbol{\psi}_d'\boldsymbol{\psi}_d + \alpha_1} \end{aligned} \tag{18}$$

where $\hat{r}_{ij}$ is an aforementioned preference prediction and will be cached and updated dynamically. In the case of classification, resorting to the variational upper bound of logistic loss, the closed form of updating ϕ_{id} is very similar to Eq (18), except that the first term of numerator in the first part is replaced by $\sum_{j \in \mathbb{I}_i}\big(\frac{1}{4}r_{ij} - \lambda(\hat{r}_{ij})(\hat{r}_{ij} - \phi_{id}\psi_{jd})\big)\psi_{jd}$ and the first part of denominator is replaced by $\sum_{j \in \mathbb{I}_i}\lambda(\hat{r}_{ij})\psi_{jd}^2$.

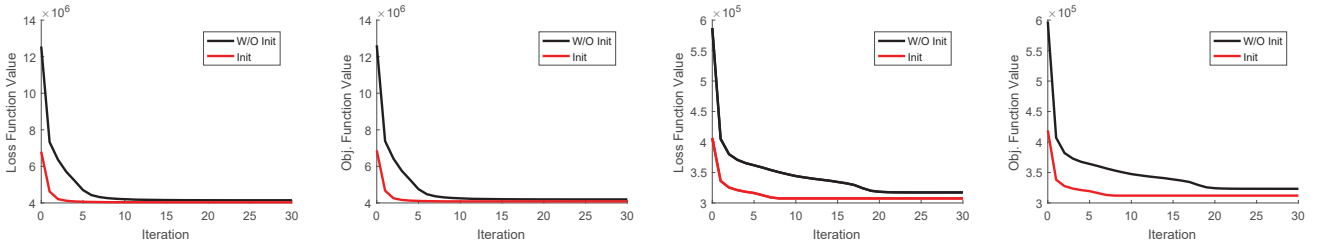
In both cases, the updating rule for ψ_{jd} could be derived similarly. Following the same analysis as before, its complexity of each iteration could be reduced by a factor of D compared to a method which directly performs optimization with respect to $\boldsymbol{\phi}_i$. But the former one may require a larger number of iterations until convergence. After convergence, assuming the solution of the relaxed objective function is $(\boldsymbol{\Phi}^*, \boldsymbol{\Psi}^*, \boldsymbol{P}^*, \boldsymbol{Q}^*, \boldsymbol{U}^*, \boldsymbol{V}^*)$, the parameters $(\boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{U}, \boldsymbol{V})$ in Eq (3) could be initialized as a feasible solution $(\operatorname{sgn}(\boldsymbol{\Phi}^*), \operatorname{sgn}(\boldsymbol{\Psi}^*), \boldsymbol{P}^*, \boldsymbol{Q}^*, \boldsymbol{U}^*, \boldsymbol{V}^*)$. The effectiveness of the proposed initialization algorithm is illustrated in Fig 1.

4 EXPERIMENTS

4.1 Datasets

We evaluate the proposed algorithm on three public datasets of explicit feedback from different real-world online websites. In these three datasets, each user is assumed to have only one rating for an item, otherwise, the average value of multiple rating scores is assigned to this item.

The first dataset, denoted as **Yelp**, is the latest Yelp Challenge dataset, which originally includes 2,685,066 ratings from 409,117 users and 85,539 items (points of interest, such as restaurants, hotels and shopping malls). The rating scores are integers from 1 to 5. Most items are usually associated to a set of textual reviews. For



(a) Regression-Loss Function Value (b) Regression-Objective Function Value (c) Classification-Loss Function Value (d) Classification-Objective Function Value

Figure 1: Convergence curve of the overall objective function and loss (square loss and logit loss) function with/without initialization on the Yelp dataset. We see that it indeed helps to achieve faster convergence and lower objective/loss values.

each item, we aggregate all of its textual reviews and represent them by bag of words, after filtering stopping words and using tf-idf to choose the top 8,000 distinct words as the vocabulary according to [28].

The second dataset, denoted as **Amazon**, is a subset of 8,898,041 user ratings for Amazon books [20], where all users and all items originally have at least 5 ratings. All its rating scores are integers from 1 to 5. Similarly, most books are provided a set of textual reviews. A similar preprocessing procedure is applied for each book's reviews to obtain the content representation of each book.

The third one, denoted as **MovieLens**, is from the classic MovieLens 10M dataset, originally including 10,000,054 rating from 71,567 users for 10,681 items (movies). The rating scores are from 0.5 to 5 with 0.5 granularity. Most movies in this dataset are associated with 3-5 labels from a dictionary of 18 genre labels.

Due to the extreme sparsity of the Yelp and Amazon original datasets, we remove users who have less than 20 ratings and remove items which are rated by less than 20 users. Such a filtering strategy is also applied for the MovieLens dataset. Table 1 summaries the filtered datasets for evaluation.

Table 1: Data statistics of three datasets

Datasets	#users	#items	#ratings	Density
MovieLens	69,838	8,940	9,983,758	1.60%
Yelp	13,679	12,922	640,143	0.36%
Amazon	35,151	33,195	1,732,060	0.15%

4.2 Evaluation Framework

We will investigate the capability of the proposed algorithm for incorporating content information, the effectiveness of both logit loss and interaction regularization for the classification task. According to [28], there are two types of recommendation in practice: in-matrix recommendation and out-of-matrix recommendation, where the former task could be addressed by collaborative filtering and the latter task corresponds to the well-known cold-start problem in recommendation and cannot resort to collaborative filtering. Thus, it is sufficient that we evaluate the proposed algorithm from the following three perspectives for our investigation. Note that efficiency study of the hashing-based recommendation is not presented any more, since it has been extensively evaluated in previous work [37].

In-matrix regression. In this evaluation, we first randomly sample 50% ratings for each user as training and the rest 50% as testing. However, this task considers the case where each user has a set of items that she has not rated before, but that at least one other user has rated. Therefore, we carefully check this condition and always move items in the test set, which are not rated by any user in the training set, to the training set. We fit a model to the training set and evaluate it on the test set. We repeat five random splits and report the averaged results.

Out-matrix regression. This evaluation, corresponding to the cold-start problems, considers the case where a new collection of items appear but no one has rated them. In this case, we randomly sample 50% items and put all ratings for them into the training set, and then put the ratings for the rest items into the test set. This corresponds to randomly shuffling item column in the user-item matrix and then cutting matrix into two parts vertically. Hence this guarantees that none of these items in the test set are rated by any user in the training set.

In-matrix classification. This evaluation is similar to in-matrix regression, but converts the aforementioned training sets of ratings into binary like/dislike datasets. We follow the method proposed in [13, 26] for constructing the binary datasets from the rating datasets as follows. First, we treat only ratings of 4 stars or higher as "like" preference. And then for each user we add the same number of pseudo negative ("dislike") items to "like" ones by sampling in proportion to their popularity [6, 9, 25]. The popularity-based sampling is chosen to discourage trivial solutions. For each test set, only users' "like" preferences for items are kept.

4.3 Evaluation Measures

We use two measures for these evaluation tasks, respectively suitable for the regression and classification tasks.

For the regression task, error-based metrics such as root mean square error (RMSE) and mean absolute error (MAE) are diverged from the ultimate goal of practical recommender systems.

Thus we measure the recommendation performance with a widely-used NDCG (normalized Discounted Cumulative Gain) for evaluating recommendation algorithms (including most of hashing-based collaborative filtering algorithms) and information retrieval algorithms [32]. This metric has taken into account both ranking precision and the position of ratings. The average NDCG at cut off from 1 to 10 over all users is the final metric of the regression-based recommendation accuracy. The larger value of NDCG indicates higher accuracy of recommendation performance.

For the classification task, according to [13], MPR (mean percentile rank) is utilized for measuring the performance, since it is commonly used in the recommendations based on implicit feedback datasets [10, 27]. For a user i in the test set, we first rank all items not rated in the training set, and then compute the percentile rank PR_{ij} of each "like" item j in the test set with regard to this ranking:

$$PR_{ij} \triangleq \frac{1}{N - |\mathbb{I}_i(train)|} \sum_{j' \notin \mathbb{I}_i(train)} \mathbf{1}[p_{ij} < p_{ij'}],$$

where $\mathbf{1}[x] = 1$ if x is true and $\mathbf{1}[x] = 0$ otherwise, $\mathbb{I}_i(train)$ is a set of the rated items of the user i in the training set, and p_{ij} is the probability that the user i likes the item j. After that, we compute the average percentile rank over all "like" items of the user i in the test set and denoted it as PR_i. The final metric for the classification-based recommendation accuracy is computed by averaging PR_i over all users. Accordingly, the smaller MPR value indicates better rankings.

4.4 Baselines of Comparison

For hashing-based collaborative filtering, we only compare DCMF with the state-of-the-art method: **DCF** [35], which outperforms almost all two-stage binary code learning methods for collaborative filtering, include BCCF [39], PPH [37], CH [19]. Note that DCF also directly tackles a discrete optimization problem, subject to the de-correlated and balanced constraints, for seeking informative and compact binary codes for users and items. But it has not been designed for classification-based collaborative filtering, and for incorporating content information.

For feature-based recommendation system, we compare a very popular method: **libFM** [23], which has achieved the best sole-model for the track I challenge–link prediction–of KDDCup 2012 [24]. It supports both classification and regression tasks of recommendation, and provides several algorithms, including SGD, ALS and MCMC, for optimization. With the advantage of learning hyperparameters in MCMC, we choose MCMC for optimization.

4.5 Results

In three tasks, the algorithm is sensitive to the tuning parameters α_1 and α_2, so that they should not be set to large values. And we tune them on a validation set, which are randomly selected from the training data, by grid search over {1e-4, 1e-3, 1e-2, 1e-1, 1} and set both of them to 0.01. Our previous empirical studies showed that recommendation performance is insensitive to γ_1 and γ_2 [16], so we simply set $\frac{\gamma_1}{\lambda_1} = 1$ and $\frac{\gamma_2}{\lambda_2} = 1$. The most important parameter among three tasks is λ_2 since only item contents are considered. This parameter of the initialization algorithm, denoted as DCMFi, is different from DCMF. In the DCMFi, we tune it on the validation set by grid search over {100,1000,10000,100000,1000000} and respectively set them as 1000, 100000 and 1000000 on the Yelp, Amazon, and MovieLens dataset. In DCMF, it is simply set to 1. The interaction regularization, designed for implicit feedback datasets, is only put into use in the classification task. Its coefficient β is set to 0.01, 0.01 and 1e-6 on the Yelp, Amazon and MovieLens dataset after tuning on the validation set by grid search over {1e-6,1e-4,1e-2,1e-1}. β is much smaller on the MovieLens dataset since this dataset is denser than others.

4.5.1 Regression. In this task, in addition to comparing DCMF with DCF and libFM, we add DCMFi, representing the initialization algorithm with sign function applied, into the baselines. The comparison results with varying length of hash codes are shown in Fig 2, where DCF fails in the task of the out-matrix regression. From this figure, we observe that DCF works well for in-matrix regression, but adding item content through DCMF could improve the performance. Discrete constraints indeed lead to quantization loss by comparing DCMF with libFM, but their gap may also result from the adaptive learning of hyper-parameters using the MCMC inference. However, it is worth mentioning that in the task of the out-matrix regression, DCMF could outperform libFM, particularly on the MovieLens dataset. The reason still lies in the higher density of the dataset, so that libFM may overfit on the training set and put more emphasis on ratings than item content information. This could be further verified that with the increasing dimension of latent factor (or hash codes), the performance of DCMF could approach and even surpass that of libFM on three datasets. Finally, the superiority of DCMF to DCMFi demonstrates the effectiveness of the developed discrete optimization algorithm.

4.5.2 Classification. In this task, in addition to libFM and DCF, we also compare DCMF with several variants of the initialization algorithm – DCMFi, and show the results in Fig 4. We can observe that for ranking all candidate items, DCMFi(c), the initialization of DCMF with continuous latent factor, is better than DCMFi(c)/F, which removes item content, and DCMFi(c)/IF, which removes both interaction regularization and item content. Combining it with the superiority of DCMFi(c)/F to DCMFi(c)/IF, we can demonstrate the effectiveness of interaction regularization and item content for improving recommendation performance. The comparison of DCMFi(c)/IF with DCFi(c), the initialization of DCF with continuous latent factors, could reveal the benefit of using logit loss for modeling the classification task. The overall benefit of incorporating these three factors into the classification task could be further observed by the superiority of DCMF to DCF. The better performance of DCMF than DCMFi(d) shows the validity of the proposed discrete optimization algorithm. Finally, an interesting observation arises from the comparison of libFM with DCMF and DCMFi(c). On both Amazon and Yelp datasets, both DCMF and DCMFi(c) outperform libFM. This mainly depends on the consideration of interaction regularization. Due to the extremely small value of β, interaction regularization almost couldn't take effect on the MovieLens dataset. In other words, the high density of the MovieLens data is an important factor that leads libFM to demonstrating surprising recommendation performance.

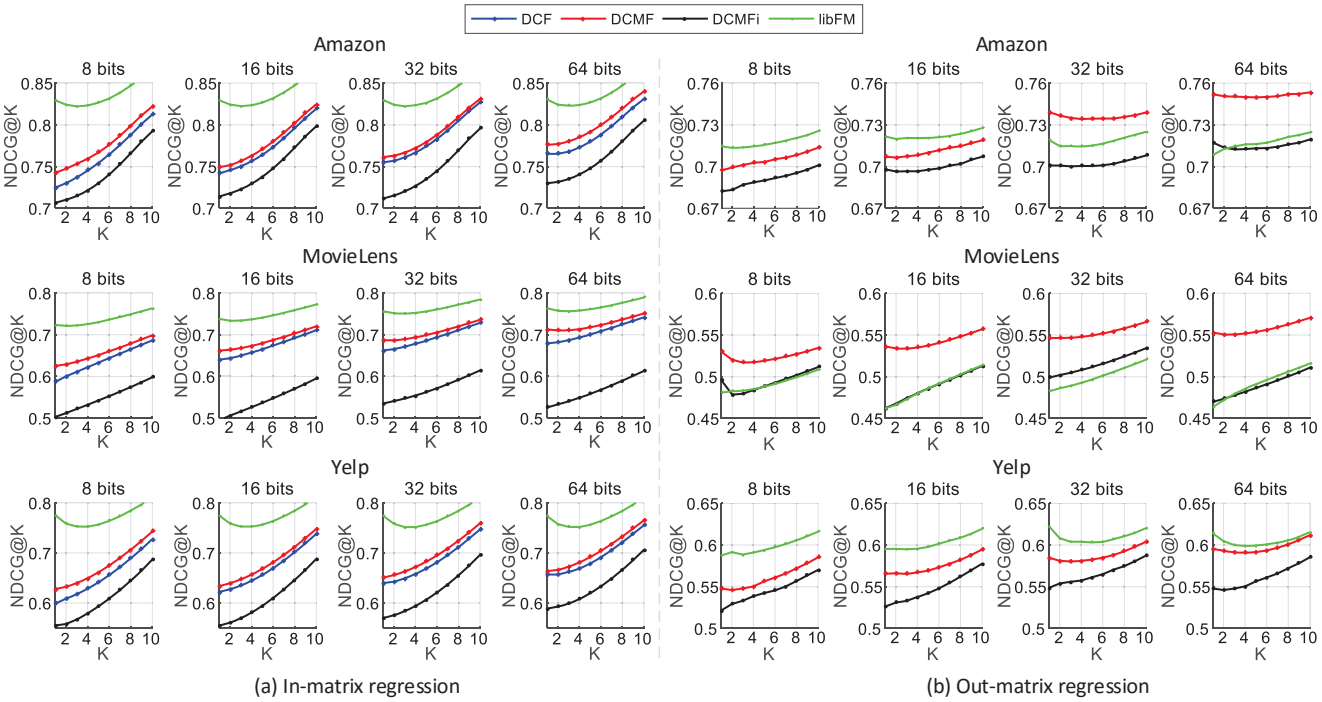


Figure 2: Item recommendation performance of hashing-based CF methods in the regression task given different code lengths

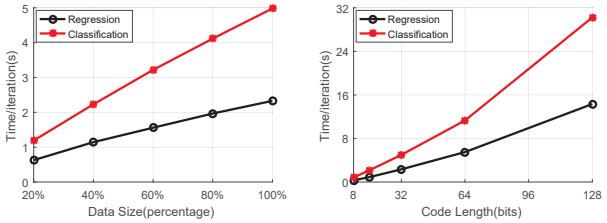


Figure 3: Efficiency v.s. data size and code length

4.6 Efficiency Study

After demonstrating the superior recommendation performance of DCMF, we further study its efficiency with the increase of data size and code length on the largest MovieLens dataset, and show them in Fig 3. When fitting DCMF of 32-D, each round of iteration costs several seconds and scales linearly with the increase of data size. When fitting DCMF on 100% training data, each round of iteration in case of classification scales quadratically with the increase of code length due to interaction regularization while the regression case is more efficient and scales linearly with code length.

5 RELATED WORK

This paper investigates hashing-based collaborative filtering at the presence of content information for regression and classification, so we mainly review recent advance of hashing-based recommendation algorithms. For comprehensive reviews of hashing techniques, please refer to [30, 31]. We also review some distributed/parallel recommender systems, since they share a similar spirit of improving recommendation efficiency.

5.1 Hashing-based Recommendation

As a pioneer work, Locality-Sensitive Hashing has been utilized for generating hash codes for Google News readers based on their click history [5]. Following this work, random projection is applied for mapping user/item learned latent representations from regularized matrix factorization into the Hamming space to obtain hash codes for users and items [12]. Similar to the idea of projection, Zhou et al. applied Iterative Quantization for generating binary code from rotated continuous user/item latent representation [39]. For the sake of deriving more compact binary codes from user/item continuous latent factors, the de-correlated constraint of different binary codes was imposed on user/item continuous latent factors [19]. However, since user/item's latent factors' magnitudes are lost due to binary discretization, hashing only preserves similarity between user and item rather than inner product based preference [37]. Thus they imposed a Constant Feature Norm (CFN) constraint on user/item continuous latent factors, and then separately quantized magnitudes and similarity. The relevant work could be summarized as two independent stages: relaxed learning of user/item latent factors with some specific constraints and subsequent binary discretization. However, such two-stage methods suffer from a large quantization loss according to [35], so direct optimization of matrix factorization with discrete constraints was proposed. To derive compact hash codes, the balanced and de-correlated constraints were further

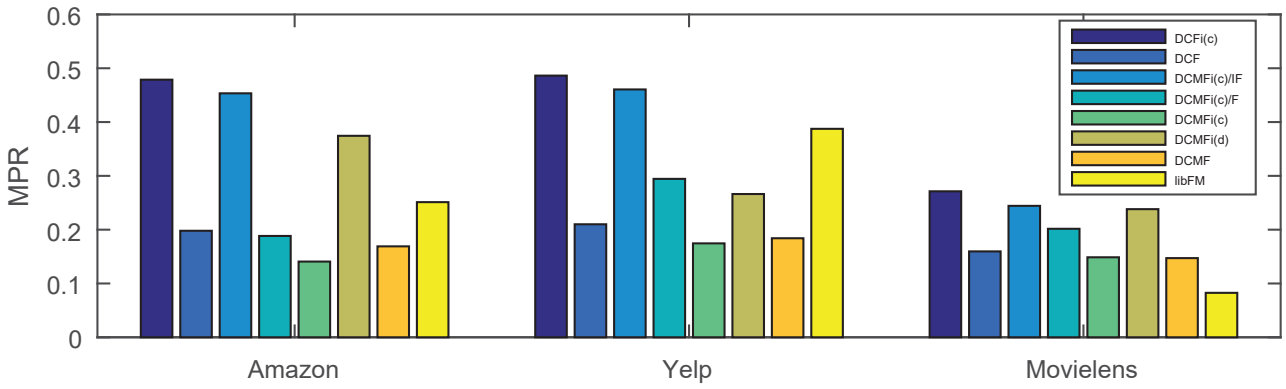


Figure 4: Item recommendation performance of hashing-based CF methods in the classification task, setting the length of hashing code to 32. DCMFi(c) and DCMFi(d) represent the initialization algorithm of DCMF using continuous latent factors and hash codes, DCFi(c) is the initialization algorithm of DCF using continuous latent factors. DCMFi(c)/F does not take content information into account. DCMFi(c)/IF does not take content information and interaction regularization into account.

imposed. For the sake of dealing with implicit feedback datasets, ranking-based loss function with binary constraints was proposed in [36].

5.2 Distributed/Parallel Recommender System

Due to superiority of matrix factorization algorithms for recommendation, their scalability has been extensively investigated recently. Based on the usage of optimization algorithms, there are mainly two lines of research. The first line is based on (block) coordinate descent. For example, user-wise (item-wise) block coordinate descent could be separated into many independent subproblems due to the independence of updating rules for different users (items) [40]. However, it is hard to be scaled up to very large-scale recommender systems since its parallelization in a distributed system requires a lot of communication cost. Instead, coordinate descent based parallel/distributed algorithm was proposed for updating rank-one factors one by one. By storing a part of residual $(r_{ij} - \boldsymbol{\phi}_i' \boldsymbol{\psi}_j)$ related to users and items in each machine, no communication of residual is required. The other line is based on stochastic gradient descent. For example, "delayed update" strategies were proposed in [2] and a lock-free approach called HogWild was investigated in [22]. In addition to proposing approximated but parallelized updating rules, exact distributed/parallel update among independent matrix blocks was proposed in [7]. When features of users and (or) items are available, feature-based matrix factorization such as Factorization Machine should be put into use. Its distributed optimization was also studied in [14, 38] based on parameter servers. However, the relevant work use continuous latent factors rather than hash codes. As an alternative way for improving the recommendation efficiency, the efficiency of hashing-based collaborative filtering in learning binary codes and online recommendation could be significantly improved with parallel/distributed techniques.

6 CONCLUSIONS

In this paper, we propose Discrete Content-aware Matrix Factorization for investigating how to learn informative and compact hash codes for users and items at the presence of content information, and extend the recommendation task from regression to classification. Simultaneously, we suggest an interaction regularization, which penalizes non-zero predicted preference, for dealing with the sparsity challenge. We then develop an efficient discrete optimization algorithm for learning hash codes for users and items. The evaluation results of the proposed algorithm on three public datasets not only demonstrates the capability of the proposed algorithm for incorporating content information, but also outperforms the state-of-the-art hashing-based collaborative filtering algorithms on both regression and classification tasks. And it is interestingly observed that the interaction regularization could greatly improve the recommendation performance when the user-item matrix is sparse, verifying the effect of the interaction regularization at addressing the sparsity issue.

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China (61502077,61631005,61602234,61572032,61502324,61532018) and the Fundamental Research Funds for the Central Universities (ZYGX2014Z012, ZYGX2016J087).

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Know. Data. Eng.* 17, 6 (2005), 734–749.
[2] Alekh Agarwal and John C Duchi. 2011. Distributed delayed stochastic optimization. In *Proceedings of NIPS'11*. 873–881.
[3] Deepak Agarwal and Bee-Chung Chen. 2009. Regression-based latent factor models. In *Proceedings of KDD'09*. ACM, 19–28.
[4] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. 2012. SVDFeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research* 13, 1 (2012), 3619–3622.
[5] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of WWW'07*. ACM, 271–280.
[6] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2012. The Yahoo! Music Dataset and KDD-Cup'11.. In *KDD Cup*. 8–18.
[7] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of KDD'11*. ACM, 69–77.

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

[9] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of SIGIR'16*, Vol. 16.

[10] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of ICDM'08*. IEEE, 263–272.

[11] T Jaakkola and M Jordan. 1997. A variational approach to Bayesian logistic regression models and their extensions. In *Sixth International Workshop on Artificial Intelligence and Statistics*, Vol. 82.

[12] Alexandros Karatzoglou, Alexander J Smola, and Markus Weimer. 2010. Collaborative Filtering on a Budget.. In *AISTATS*. 389–396.

[13] Noam Koenigstein and Ulrich Paquet. 2013. Xbox movies recommendations: Variational Bayes matrix factorization with embedded feature selection. In *Proceedings of RecSys'13*. ACM, 129–136.

[14] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. 2016. DiFacto: Distributed factorization machines. In *Proceedings of WSDM'16*. ACM, 377–386.

[15] Defu Lian, Yong Ge, Nicholas Jing Yuan, Xing Xie, and Hui Xiong. 2016. Sparse Bayesian Content-Aware Collaborative Filtering for Implicit Feedback. In *Proceedings of IJCAI'16*. AAAI.

[16] Defu Lian, Yong Ge, Fuzheng Zhang, Nicholas Jing Yuan, Xing Xie, Tao Zhou, and Yong Rui. 2015. Content-Aware Collaborative Filtering for Location Recommendation based on Human Mobility Data. In *Proceedings of ICDM'15*. IEEE, 261–270.

[17] Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. 2014. GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *Proceedings of KDD'14*. ACM, 831–840.

[18] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *Proceedings of NIPS'14*. 3419–3427.

[19] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. 2014. Collaborative hashing. In *Proceedings of CVPR'14*. 2139–2146.

[20] Julian McAuley, Rahul Pandey, and Jure Leskovec. 2015. Inferring networks of substitutable and complementary products. In *Proceedings of KDD'15*. ACM, 785–794.

[21] Marius Muja and David G Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of VISAPP'09*. 331–340.

[22] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of NIPS'11*. 693–701.

[23] Steffen Rendle. 2012. Factorization machines with libFM. *ACM Trans. Intell. Syst. Tech.* 3, 3 (2012), 57.

[24] Steffen Rendle. 2012. Social network and click-through prediction with factorization machines. In *KDD-Cup Workshop*.

[25] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of WSDM'14*. ACM, 273–282.

[26] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of UAI'09*. AUAI Press, 452–461.

[27] Harald Steck. 2010. Training and testing of recommender systems on data missing not at random. In *Proceedings of KDD'10*. ACM, 713–722.

[28] Chong Wang and David M Blei. 2011. Collaborative topic modeling for recommending scientific articles. In *Proceedings of KDD'11*. ACM, 448–456.

[29] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2012. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Mach. Intell.* 34, 12 (2012), 2393–2406.

[30] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2016. Learning to hash for indexing big data – A survey. *Proc. IEEE* 104, 1 (2016), 34–57.

[31] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2016. A survey on learning to hash. *arXiv preprint arXiv:1606.00185* (2016).

[32] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. 2007. Maximum margin matrix factorization for collaborative ranking. *Proceedings of NIPS'07* (2007), 1–8.

[33] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Proceedings of NIPS'09*. 1753–1760.

[34] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of KDD'16*. ACM, 353–362.

[35] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of SIGIR'16*, Vol. 16.

[36] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback. In *Proceedings of AAAI'17*. 1669–1675.

[37] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *Proceedings of SIGIR'14*. ACM, 183–192.

[38] Erheng Zhong, Yue Shi, Nathan Liu, and Suju Rajan. 2016. Scaling Factorization Machines with Parameter Server. In *Proceedings of CIKM'16*. ACM, 1583–1592.

[39] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proceedings of KDD'12*. ACM, 498–506.

[40] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*. Springer, 337–348.