Discrete Ranking-based Matrix Factorization with Self-Paced Learning

Yan Zhang
School of Computer Science and Engineering, University of Electronic Science and Technology of China
yixianqianzy@gmail.com

Haoyu Wang
School of Computer Science and Engineering, University of Electronic Science and Technology of China
haoyu.uestc@gmail.com

Defu Lian*
School of Computer Science and Engineering, University of Electronic Science and Technology of China
dove.ustc@gmail.com

Ivor W. Tsang
Centre for Artificial Intelligence, University of Technology Sydney
Ivor.Tsang@uts.edu.au

Hongzhi Yin
School of Information Technology and Electrical Engineering, The University of Queensland
h.yin1@uq.edu.au

Guowu Yang
School of Computer Science and Engineering, University of Electronic Science and Technology of China
guowu@uestc.edu.cn

ABSTRACT

The efficiency of top-k recommendation is vital to large-scale recommender systems. Hashing is not only an efficient alternative but also complementary to distributed computing, and also a practical and effective option in a computing environment with limited resources. Hashing techniques improve the efficiency of online recommendation by representing users and items by binary codes. However, objective functions of existing methods are not consistent with ultimate goals of recommender systems, and are often optimized via discrete coordinate descent, easily getting stuck in a local optimum. To this end, we propose a Discrete Ranking-based Matrix Factorization (DRMF) algorithm based on each user's pairwise preferences, and formulate it into binary quadratic programming problems to learn binary codes. Due to non-convexity and binary constraints, we further propose self-paced learning for improving the optimization, to include pairwise preferences gradually from easy to complex. We finally evaluate the proposed algorithm on three public real-world datasets, and show that the proposed algorithm outperforms the state-of-the-art hashing-based recommendation algorithms, and even achieves comparable performance to matrix factorization methods.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**; **Personalization**; **Recommender systems**;

KEYWORDS

Ranking-based matrix factorization; hashing; binary quadratic programming; self-paced learning

*The corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '18, August 19–23, 2018, London, United Kingdom
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220116

ACM Reference Format:
Yan Zhang, Haoyu Wang, Defu Lian, Ivor W. Tsang, Hongzhi Yin, and Guowu Yang. 2018. Discrete Ranking-based Matrix Factorization with Self-Paced Learning. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220116

1 INTRODUCTION

With the development of E-commerce, recommender systems have been extensively applied in a growing number of e-commerce websites for helping their customers find desirable products. However, it is challenging to match products to their potential customers accurately and efficiently, particularly with the ever-growing scales of products and customers.

Matching products to potential customers is an essential goal of recommender systems, whose techniques have been widely investigated for improving the accuracy [1, 13]. As a critical class of recommendation methods, collaborative filtering (CF) techniques exemplified by latent factor models (e.g., matrix factorization) show not only high effectiveness but also the best role-model performance in real-world recommender systems. The basic method of latent factor models usually factorizes an $M \times N$ rating/preference matrix to map M users and N items into a D-dimensional latent space [13]. Each user's preference for each item is predicted as the inner product between their representation in the latent space. To align with the ultimate goal of recommender systems, matrix factorization (MF) with ranking based objectives, such as BPR [23], CofiRank [29], and ListCF [25], have been proposed, showing superior performance of recommendation to others.

However, it is challenging to generate an immediate response for the better user experience, particularly with growing scale of users and items. Specifically, recommending the top-k preferred items for each user from those N items costs $O(ND + N \log k)$ with real-valued latent factors. Furthermore, many recommender systems [4, 6, 26] update user interest frequently, and a good recommender system shall recommend items which are matched with users' latest interests. In this case, the top-k preferred items should be generated frequently. Although parallel/distributed computing is a practical solution, the computational burden remains unchanged.

Recent studies show that hashing based recommendation methods, by representing users and items by binary codes, are promising to tackle the efficiency challenge [31, 33, 35]. Specifically, the preference score in this case could be fast calculated via Hamming distance. One can even use multi-index hashing [22] for exact top-k preferred items with sub-linear time complexity. More advanced indexing techniques can also be leveraged to find approximated ones with logarithmic or constant time complexity [27]. Besides, hashing has advantages concerning storage cost. In particular, each dimension of binary codes is only stored by one bit instead of 32/64 bits float number for real-valued vectors, so the storage for representations of users and items can be dramatically reduced.

However, many previous hashing-based recommender systems improve the efficiency at the price of accuracy due to a large amount of quantization loss [19, 33, 35]. Although effort has been devoted to reducing quantization loss by discrete optimization [31], this method has two important limitations. First, the optimized objective is not consistent with ultimate goals of recommender systems, i.e., generating top-k preferred items for each user. Second, the loss function for optimization is discontinuous, so optimization via discrete coordinate descent may easily get stuck in a local optimum, even with a good initialization.

To address the above challenges, we propose a Discrete Ranking-based Matrix Factorization (DRMF) to optimize cross-entropy loss between the true pairwise preference ranking and predicted pairwise preference ranking for each user. Due to non-linearity of the cross-entropy loss, it is difficult to derive closed-form solutions for updating user/item binary codes, so that we seek its local quadratic upper bound [10]. We then formulate the upper bound into a series of binary quadratic programming problems and resort to semi-definite programming solvers followed by Gaussian randomization to obtain binary codes. Hence, binary codes of each user/item are optimized as a whole instead of bit-by-bit in discrete coordinate descent. This optimization strategy should be superior to discrete coordinate descent in achieving lower loss values. This is similar to the difference between blocked Gibbs sampling and collapsed Gibbs sampling [9]. To further improve optimization, we exploit self-paced learning (SPL) to include pairwise preferences gradually from easy to complex. SPL has been demonstrated to be useful to avoid bad local optima and achieve better generalization results in latent variable models [14–16, 34].

We finally evaluate the proposed algorithms on three real-world rating datasets. The experimental results show that it significantly outperforms the state-of-the-art hashing-based recommendation algorithms, and even achieve comparable recommendation performance to matrix factorization methods. The results also verify the effectiveness of self-paced learning for improving recommendation performance, and demonstrate efficiency improvements of the top-k recommendation via binary coding.

To summarize, we make the following contributions:

- We propose discrete ranking-based matrix factorization for directly optimizing pairwise ranking cross-entropy loss. Hence, the objective function is consistent with ultimate goals of real-world recommender systems.
- We transform the discrete objective function into a series of binary quadratic programming problems after seeking the local quadratic upper bound of the cross-entropy cost, and thus learn each binary code as a whole instead of bit-by-bit.
- We incorporate self-paced learning into the overall optimization procedure to avoid bad local optima and achieve better generalization results. To the best of our knowledge, self-paced learning is used for discrete and ranking-based matrix factorization for the first time.
- We test the proposed algorithm on three real-world datasets and demonstrate its consistent and significant superiority to the state-of-the-art hashing-based and ranking-based recommendation methods.

2 RELATED WORK

The recent advance of hashing-based recommendation algorithms will be discussed below. Please refer [28] for comprehensive reviews of hashing techniques and their applications in multimedia retrieval and computer vision.

2.1 Quantization-based Hashing

Due to binary constraints, learning binary codes is generally NP-hard. Quantization-based hashing or two-stage hashing consists of relaxed optimization and binary quantization. By relaxed optimization with some specific constraints, user/item continuous latent representation are obtained, and binary quantization then converts continuous latent representations into binary codes. Before 2012, there are a few two-stage frameworks, such as [5, 12, 35]. Recently, for deriving compact binary codes from user/item latent factors, the uncorrelated constraint of binary codes was imposed when learning user/item latent factors [19]. However, since user/item's latent factors' magnitudes are lost out of binary quantization [33], Zhang et al. imposed a Constant Feature Norm (CFN) constraint on user/item latent factors, and then separately quantized magnitudes of latent factors and cosine similarity between the user and item latent factors.

2.2 Optimization-based Hashing

These quantization-based hashing methods suffer from large information loss in the binary quantization stage according to [31], so matrix factorization with binary constraints was optimized. To derive compact hash codes for users and items, the balanced and de-correlated constraints were further imposed. For dealing with implicit feedback datasets, a ranking-based AUC objective function with the similar constraints was proposed in [32], and implicit/interaction regularization [2, 6] was proposed for penalizing non-zero predicted preference [17].

3 DISCRETE RANKING-BASED MATRIX FACTORIZATION WITH SELF-PACED LEARNING

Different from existing rating-based hashing recommender systems [17, 31, 33], we propose a ranking-based hashing framework, Discrete Ranking-based Matrix Factorization with Self-Paced Learning (DRMF-SPL). DRMF-SPL provides a more reliable efficient recommendation framework based on the following three points: first, the ranking-based objective is consistent with ultimate goals of recommender systems – recommending top-k preferred items to

users, which is vital to ascertain accurate recommendation; second, the block-wise rule of updating each user's/item's binary code as a whole is applied during the optimization of the proposed ranking-based objective. It should be superior to the bit-wise rule of discrete coordinate descent in achieving lower loss values; third, self-paced learning is applied to further improve optimization by including pairwise preferences gradually from easy to complex, which is helpful to avoid bad local optima and achieve better performance for recommender systems.

We first introduce the preference model and loss funcitons of discrete ranking-based matrix factorization (DRMF) and its self-paced learning variant DRMF-SPL in Section 3.1. We then present the model optimization algorithm of DRMF and DRMF-SPL in Section 3.2, and explore how to optimize the objective function by a strategy of solving binary quadratic programming (BQP). Specifically, we first seek its local quadratic upper bound, and then formulate the upper bound into binary quadratic programming problems, by which we update each binary code as a whole instead of bit-by-bit. We finally analyze the complexity of the proposed algorithm in Section 3.3.

3.1 Loss Function

3.1.1 Preference Model. Suppose we have M users, N items and integer rating values from 1 to K. The user set is denoted as U and the item set is denoted as I. Let r_{ui} in the matrix $\boldsymbol{R}$ be the rating of user u to item i. $\boldsymbol{P} \in \{\pm 1\}^{M\times D}$ and $\boldsymbol{Q} \in \{\pm 1\}^{N\times D}$ are binary matrices stacked by hash codes of users and items, respectively, whose the u-th row $\boldsymbol{p}_u$ of $\boldsymbol{P}$ and the i-th row $\boldsymbol{q}_i$ of $\boldsymbol{Q}$ represent hash codes of user u and item i, respectively. Let $\Omega = \{(u,i)|r_{ui} \text{ is a observed rating}\}$ be the index set of observed entries in $\boldsymbol{R}$, $U_i = \{u|(u,i) \in \Omega\}$ be users who have rated item i, and $I_u = \{i|(u,i) \in \Omega\}$ be items that user u has rated. Then the preference of user u for item i is defined as

$$\begin{aligned}\hat{r}_{ui} &= \frac{1}{D}\sum_{k=1}^{D}\mathbb{I}(p_{uk} = q_{ik}) = 1 - \frac{1}{D}H(\boldsymbol{p}_u, \boldsymbol{q}_i) \\ &= \frac{1}{2} + \frac{1}{2D}\boldsymbol{p}_u^T\boldsymbol{q}_i,\end{aligned} \tag{1}$$

where $\mathbb{I}(\cdot)$ is an indicator function that returns 1 if the input is true and returns 0 otherwise, and $H(\boldsymbol{p}_u, \boldsymbol{q}_i)$ denotes the Hamming distance between $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$. Note that this preference can be efficiently computed by Hamming distance. Then we denote $\hat{\boldsymbol{R}} = [\hat{r}_{ui}] \in [0,1]^{M\times N}$ the predicted preference matrix.

3.1.2 The DRMF Model. In order to align with ultimate goals of recommender systems, we define the loss function based on pairwise preferences instead of ratings themselves. In this paper, we follow RankNet [3] and use cross entropy for loss function. In particular, we first define P_{uij} the target probability that item i is preferred by a user u to item j and specify it as follows:

$$P_{uij} = \begin{cases} 1 & \text{if } r_{ui} > r_{uj} \\ 0.5 & \text{if } r_{ui} = r_{uj} \\ 0 & \text{if } r_{ui} < r_{uj} \end{cases}. \tag{2}$$

We then define $\hat{P}_{uij}$ the predicted probability that item i is preferred by a user u to item j, and model it based on a logistic function,

$$\hat{P}_{uij} = \frac{e^{\hat{o}_{uij}}}{1 + e^{\hat{o}_{uij}}}, \tag{3}$$

where $\hat{o}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$. If $\hat{r}_{ui} > \hat{r}_{uj}$, then $\hat{P}_{uij} > 0.5$, indicating there are over 50% chances that user u prefers item i to item j; otherwise, $\hat{P}_{uij} < 0.5$. Then cross entropy is used to penalize inconsistent pairwise ranking, i.e.,

$$\begin{aligned}\ell_{uij} &= -P_{uij}\log\hat{P}_{uij} - (1 - P_{uij})\log(1 - \hat{P}_{uij}) \\ &= -P_{uij}\hat{o}_{uij} + \log\left(1 + e^{\hat{o}_{uij}}\right)\end{aligned} \tag{4}$$

Let's denote $S = \{(u,i,j)|(u,i) \in \Omega, (u,j) \in \Omega, i < j\}$ a set of user-specific pairs of rated items. Then the objective function of DRMF is represented as

$$\begin{aligned}\min_{\boldsymbol{P},\boldsymbol{Q}} \mathcal{L}_1 = &\sum_{(u,i,j)\in S} -P_{uij}\hat{o}_{uij} + \log\left(1 + e^{\hat{o}_{uij}}\right) \\ &\text{s.t. } \boldsymbol{P} \in \{\pm 1\}^{M\times D}, \boldsymbol{Q} \in \{\pm 1\}^{N\times D},\end{aligned} \tag{5}$$

where $\hat{o}_{uij} = \hat{r}_{ui} - \hat{r}_{uj} = \frac{1}{2D}\boldsymbol{p}_u^T(\boldsymbol{q}_i - \boldsymbol{q}_j)$.

3.1.3 The DRMF-SPL Model. Owing to discontinuous and non-linearity of the loss function for optimization in DRMF, directly optimization may easily stuck in local optima, therefore self-paced learning is introduced for improving optimization. Self-paced learning has been shown significant improvements in generalization and effectiveness in alleviating the bad local optimum problem in non-convex optimization. When applied in matrix factorization, which is a non-convex problem with missing values, it sequentially includes elements of the targeted matrix $\boldsymbol{R}$ into learning algorithms from easy to challenging. In particular, it optimizes the following objective function [34]:

$$\begin{aligned}&\sum_{(u,i)\in\Omega} w_{ui}\ell(r_{ui}, [\boldsymbol{P}\boldsymbol{Q}^T]_{ui}) + \lambda\Gamma(\boldsymbol{P},\boldsymbol{Q}) + \sum_{(u,i)\in\Omega} f_k(w_{ui}) \\ &\text{s.t. } w_{ui} \in [0,1] \text{ if } (u,i) \in \Omega,\end{aligned} \tag{6}$$

where $\Gamma(\boldsymbol{P},\boldsymbol{Q})$ is the regularizer term of matrix factorization. $f_k(w_{ui})$ is a self-paced regularizer, which consists of two classes: the "hard" self-paced regularizers that determine the samples (observed ratings in $\boldsymbol{R}$) to be selected or not be selected in training, i.e., $\boldsymbol{w}_{ui}$ takes value 0 or 1, and the "soft" self-paced regularizers that assign weights to selected samples, so that their levels of simplicity can be distinguished from each other by the weights, i.e., $\boldsymbol{w}_{ui}$ takes value in the closed interval $[0,1]$. Thus self-paced regularizers are also regarded as weighting schemes of samples.

Similarly, taking the user-specific pairwise item preferences as samples, easy pairs are considered first for optimization, and with progressing of training procedure, difficult pairs are gradually taken into account. DRMF-SPL solves the following optimization problem:

$$\begin{aligned}\min_{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{W}} \mathcal{L}_2 = &\sum_{(u,i,j)\in S} \left(w_{uij}\ell_{uij} + f_k(w_{uij})\right) \\ &\text{s.t. } \boldsymbol{P} \in \{\pm 1\}^{M\times D}, \boldsymbol{Q} \in \{\pm 1\}^{N\times D}, \boldsymbol{W} \in [0,1]^{|S|}.\end{aligned} \tag{7}$$

According to the previous study, no single weighting scheme can always work best, and their performance is often comparable with

respect to each other. The choice of weighting schemes is problem-specific, so we compare different weighting schemes and select the best one. In this paper, we apply two self-paced regularizers: a "hard" self-paced regularizer $f_k(w_{uij}) = -\frac{1}{k}w_{uij}$ [14] and a "soft" linear weighting $f_k(w_{uij}) = \frac{1}{k}(\frac{1}{2}w_{uij}^2 - w_{uij})$ [11], to test their performances. The experiments show that the "soft" linear weighting can achieve better performance.

Since hashing based recommendation frameworks generally suffer from low recommendation accuracy, it's necessary to incorporate self-paced learning into the overall optimization for avoiding bad local optima and achieving better recommendation accuracy. To the best of our knowledge, self-paced learning is used for discrete and ranking-based matrix factorization for the first time.

3.2 Model Optimization

Due to non-linearity of Eq (5), it is difficult to derive closed formulas for updating binary codes $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$. However, ℓ_{uij} is convex with respect to $\hat{o}_{uij}$, so we can seek its quadratic upper bound. In particular, based on Jaakkola-Jordan bound [10], $\log\left(1+\exp^{\hat{o}_{uij}}\right)$ is locally bounded from above:

$$\log\left(1+e^{\hat{o}_{uij}}\right) \le \lambda(\varphi_{uij})(\hat{o}_{uij}^2 - \varphi_{uij}^2) + \frac{1}{2}(\hat{o}_{uij} - \varphi_{uij}) + \log\left(1+e^{\varphi_{uij}}\right), \quad (8)$$

where $\lambda(\varphi_{uij}) = \frac{1}{4\varphi_{uij}}\tanh(\frac{\varphi_{uij}}{2}) = \frac{1}{2\varphi_{uij}}\left(\sigma(\varphi_{uij}) - \frac{1}{2}\right)$, $\sigma(x) = \frac{1}{1+e^{-x}}$ is a sigmoid function. Note that the optimal $\varphi_{uij} = \hat{o}_{uij}$, as we will show later, so this upper bound is tight. Hence, ℓ_{uij} is then bounded by a upper bound $\tilde{\ell}_{uij}$,

$$\begin{aligned}\tilde{\ell}_{uij} &= -P_{uij}\hat{o}_{uij} + \lambda(\varphi_{uij})(\hat{o}_{uij}^2 - \varphi_{uij}^2) \\ &\quad + \frac{1}{2}(\hat{o}_{uij} - \varphi_{uij}) + \log\left(1+e^{\varphi_{uij}}\right),\end{aligned} \quad (9)$$

After introducing a variational parameter φ_{uij}, the objective function of DRMF-SPL, Eq (7) is re-formulated as:

$$\begin{aligned}&\min_{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{\Phi},\boldsymbol{W}} \mathcal{L}_3 = \sum_{(u,i,j)\in S}\left(w_{uij}\tilde{\ell}_{uij} + f_k(w_{uij})\right) \\ &\text{s.t. } \boldsymbol{P}\in\{\pm1\}^{M\times D}, \boldsymbol{Q}\in\{\pm1\}^{N\times D}, \boldsymbol{\Phi}\in\mathbb{R}^{|S|}, \boldsymbol{W}\in[0,1]^{|S|}.\end{aligned} \quad (10)$$

This method is called Majorize-Minimization [8]. According to [14], alternative search strategy is used for optimization in self-paced learning, by alternatively optimizing the parameters $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{\Phi}$ given the weight of pairwise preferences $\boldsymbol{W}$, and optimizing $\boldsymbol{W}$ given $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{\Phi}$. We denote the optimization of DRMF-SPL given $\boldsymbol{W}$ as $\text{DRMF}_{\boldsymbol{W}}$:

$$\min_{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{\Phi}} \mathcal{L}_{\boldsymbol{W}} = \sum_{(u,i,j)\in S} w_{uij}\tilde{\ell}_{uij}. \quad (11)$$

In fact, for the "hard" self-paced regularizer, optimizing DRMF-SPL given $\boldsymbol{W}$ is equivalent to solving the DRMF problem. The optimization of $\mathcal{L}_{\boldsymbol{W}}$ is also based on alternating optimization, which is an important technique for matrix factorization in addition to stochastic gradient descent [13]. Below, we first introduce how to update $\boldsymbol{p}_u$ ($\boldsymbol{q}_i$) as a whole vector instead of bit-by-bit. To distinguish this strategy from discrete coordinate descent, we also briefly introduce discrete coordinate descent in Section 3.2.1. We then introduce how to update the variational parameter $\boldsymbol{\Phi}$ and the weight $\boldsymbol{W}$. The overall procedure of minimizing $\mathcal{L}_{\boldsymbol{W}}$ is shown in Algorithm 1 and the procedure of DRMF-SPL is shown in Algorithm 2.

3.2.1 *Updating User Hash Codes P.*

Given $\boldsymbol{Q}$, $\boldsymbol{\Phi}$ and $\boldsymbol{W}$, we update $\boldsymbol{p}_u$ for each user u separately. Ignoring terms irrelevant to $\boldsymbol{p}_u$, the sub-objective function is quadratic with respect to $\boldsymbol{p}_u$,

$$\min_{\boldsymbol{p}_u\in\{\pm1\}^D} \boldsymbol{p}_u^T C_u \boldsymbol{p}_u + \boldsymbol{d}_u^T \boldsymbol{p}_u, \quad (12)$$

where

$$C_u = \sum_{i\in I_u}\boldsymbol{q}_i\boldsymbol{q}_i^T\Big(\sum_{j\in I_u}\lambda(\varphi_{uij})w_{uij}\Big) - \sum_{i\in I_u}\boldsymbol{q}_i\Big(\sum_{j\in I_u}\lambda(\varphi_{uij})w_{uij}\boldsymbol{q}_j^T\Big), \quad (13)$$

and

$$\boldsymbol{d}_u = \sum_{i\in I_u}\boldsymbol{q}_i\Big(\sum_{j\in I_u, i<j}\frac{1}{2}w_{uij} - w_{uij}P_{uij}\Big) - \sum_{i\in I_u}\sum_{j\in I_u, i<j}\Big(\frac{1}{2}w_{uij} - w_{uij}P_{uij}\Big)\boldsymbol{q}_j. \quad (14)$$

It is easy to verify that C_u is a real symmetric matrix since $\lambda(\varphi_{uij}) = \lambda(\varphi_{uji})$ and $w_{uij} = w_{uji}$. And the constraint $i \neq j$ is dropped in C_u by noting that $\lambda(\varphi_{uij}) = 0$ if $i = j$. However, the problem (12) is generally NP-hard, so several greedy algorithms are proposed. The commonly-used algorithm is discrete coordinate descent [24, 31], which updates $\boldsymbol{p}_u$ by a bit-wise way. In particular, denoting p_{uk} as the k-th bit of $\boldsymbol{p}_u$ and $\boldsymbol{p}_{u\bar{k}}$ as the rest codes excluding p_{uk}, the bit-wise method with discrete coordinate descent updates p_{uk} when $\boldsymbol{p}_{u\bar{k}}$ fixed. Ignoring the terms independent of p_{uk}, the Eq (12) could be rewritten as a sub-objective with respect to p_{uk}. However, as a greedy algorithm, discrete coordinate descent easily falls into local optima and is very sensitive to the initialization of parameters. In contrast to discrete coordinate descent, we will update $\boldsymbol{p}_u$ all together in a block-wise way instead of bit-wise.

When treating $\boldsymbol{p}_u$ as a vector variable, Eq (12) is a binary quadratic programming (BQP) problem, but inhomogeneous. Semi-definite relaxation (SDR) technique is a powerful computationally efficient approximation technique for BQP problems [20], but is based on homogeneous problems. To this end, we first transform Eq (12) to a homogeneous BQP. Let's introduce a scalar $v\in\mathbb{R}$, and then Eq (12) can be rewritten as

$$\begin{aligned}&\min_{\boldsymbol{x}\in\mathbb{R}^{D+1}} \boldsymbol{x}^T\boldsymbol{H}_u\boldsymbol{x} \\ &\text{s.t. } x_d^2 = 1, d = 1\cdots, D+1,\end{aligned} \quad (15)$$

where $\boldsymbol{H}_u = \begin{bmatrix} C_u, & \frac{\boldsymbol{d}_u}{2} \\ \frac{\boldsymbol{d}_u^T}{2}, & 0 \end{bmatrix}$ is a real symmetric matrix and $\boldsymbol{x} = [\boldsymbol{p}_u^T, v]^T$. The details about how SDR solves BPQ problems are elaborated in Appendix. In some cases, $\boldsymbol{H}_u$ is singular, and SDR may fail. Therefore, $\tilde{C}_u = \boldsymbol{C}_u + \alpha\boldsymbol{I}_D$ is used in $\boldsymbol{H}_u$.

3.2.2 *Updating Item Hash Codes Q.*

Similarly, given $\boldsymbol{P}$, $\boldsymbol{\Phi}$, and $\boldsymbol{W}$, we can also derive a quadratic sub-objective with respect to $\boldsymbol{q}_i$,

$$\min_{\boldsymbol{q}_i\in\{\pm1\}^D} \boldsymbol{q}_i^T C_i \boldsymbol{q}_i + \boldsymbol{d}_i^T\boldsymbol{q}_i \quad (16)$$

where

$$C_i = \sum_{u \in U_i} \sum_{j \in I_u} \lambda(\varphi_{uij}) w_{uij} \boldsymbol{p}_u \boldsymbol{p}_u^T \tag{17}$$

and

$$\boldsymbol{d}_i = \frac{1}{2} \sum_{u \in U_i} \sum_{j \in I_u} w_{uij} \boldsymbol{p}_u - 2 \sum_{u \in U_i} \boldsymbol{p}_u \boldsymbol{p}_u^T \sum_{j \in I_u} \lambda(\varphi_{uij}) w_{uij} \boldsymbol{q}_j - \sum_{u \in U_i} \Big(\sum_{j \in I_u} w_{uij} P_{uij} \Big) \boldsymbol{p}_u \tag{18}$$

Similar to updating $\boldsymbol{p}_u$ in Section 3.2.1, the constraint $i \neq j$ is also dropped, and $\boldsymbol{q}_i$ can be also updated by solving a similar BQP problem. To avoid the failure of SDR in some cases, $\tilde{C}_i = C_i + \beta \boldsymbol{I}_D$ is also used in this paper.

3.2.3 Learning Variational Parameter Φ. Fixing $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{W}$, the sub-objective function with respect to φ_{uij} is derived as follows Eq (19) by substituting $\tilde{\ell}_{uij}$ into Eq (10):

$$\min_{\boldsymbol{\varphi}_{uij} \in \mathbb{R}} w_{uij} \big(\lambda(\varphi_{uij})(\hat{o}_{uij}^2 - \varphi_{uij}^2) - \frac{1}{2} \varphi_{uij} + \log(1 + e^{\varphi_{uij}}) \big). \tag{19}$$

The sub-objective is convex since its second derivative is greater than zero. Hence, the optimal value is achieved when the derivation of Eq (19) equals to zero. For each user, the derivation of Eq (19) equals to zero at

$$\varphi_{uij} = \hat{o}_{uij} = \hat{r}_{ui} - \hat{r}_{uj} \tag{20}$$

Actually, we don't need store $\varphi_{uij}, \forall (u, i, j) \in S$, since we only need to store the all predicted ratings $\hat{r}_{ui}$. We do also not need to store $\lambda(\varphi_{uij})$, since it can be computed on-the-fly in constant time when we cache these $\hat{r}_{ui}$.

3.2.4 Updating Weight of Pairwise Preferences. Given $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{\Phi}$, the sub-objective with respect to $\boldsymbol{w}_{uij}$ is

$$\min_{\boldsymbol{w}_{uij} \in [0,1]} w_{uij} \tilde{\ell}_{uij} + f_k(w_{uij}), \tag{21}$$

the optimal weight w_{uij} is obtained by setting the derivative of Eq (21) to zero. Experiments show that the linear "soft" weighting can achieve better performance. Therefore, we choose the linear "soft" self-paced regularizer $f_k(w_{uij}) = \frac{1}{k}(\frac{1}{2} w_{uij}^2 - w_{uij})$ in this paper, which indicates that the optimal weight for each user u is

$$w_{uij} = \begin{cases} -k\tilde{\ell}_{uij} + 1 & \text{if } \tilde{\ell}_{uij} \leq \frac{1}{k}; \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

3.3 Complexity Analysis

In this section, we analyze the complexity of updating binary codes $\boldsymbol{P}$ and $\boldsymbol{Q}$ in one round of iteration in Algorithm 2. Take the iteration of $\boldsymbol{P}$ as an example, updating $\boldsymbol{p}_u$ for each user u contains three steps: computing $\boldsymbol{C}_u$ and $\boldsymbol{d}_u$, solving an SDR problem, and conducting a Gaussian randomization procedure.

For speeding up the computing of $\boldsymbol{C}_u$ and $\boldsymbol{d}_u$, we first cache the all preference prediction $\hat{r}_{ui}$, we then calculate terms a_{ui}, b_{ui}, $\bar{\boldsymbol{q}}_{ui}$ and $\tilde{\boldsymbol{q}}_{ui}$ (listed in Table 1) of $\boldsymbol{C}_u$ and $\boldsymbol{d}_u$ outside the user loop in Algorithm 1. The complexity of computing these quantities is $O(|I_u|D)$ as long as we cache the all $\hat{r}_{ui}$ first. After that, the computation of $\boldsymbol{C}_u$ and $\boldsymbol{d}_u$ costs $O(|I_u|D^2)$ and $O(|I_u|D)$, respectively. In terms of solving an SDR problem, the complexity is $O(D^{3.5})$ as introduced in Appendix. Gaussian randomization procedure costs $O(D^3 + LD^2)$ since it can be achieved by eigenvalue decomposition of $\boldsymbol{X}^*$ with complexity of $O(D^3)$, i.e., $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$, and drawing L samples $\{\boldsymbol{\xi}_l\}_{l \in \{1, \cdots, L\}}$ with complexity of $O(LD^2)$ by $\boldsymbol{\xi}_l = \boldsymbol{V}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{\epsilon}_l$, where $\boldsymbol{\epsilon}_l$ is generated from $N(\boldsymbol{0}, \boldsymbol{I}_n)$, which is introduced in Appendix in detail. Therefore, solving the optimal $\boldsymbol{P}$ costs $O\big(\|\boldsymbol{R}\|_0 D^2 + M(D^{3.5} + LD^2)\big)$ in one round of iteration, where $\|\boldsymbol{R}\|_0$ denotes the number of observed entries in the matrix $\boldsymbol{R}$.

Similarly, for speeding up the computing of $\boldsymbol{C}_i$ and $\boldsymbol{d}_i$, we first calculate terms a_{ui}, d_{ui}, $\bar{\boldsymbol{q}}_{ui}$ and $\bar{\boldsymbol{p}}_{ui}$ (listed in Table 1) of $\boldsymbol{C}_i$ and $\boldsymbol{d}_i$ outside the item loop. The time complexity of computing $\boldsymbol{C}_i$ and $\boldsymbol{d}_i$ is $O(|U_i|D^2)$ for each item i. Therefore, solving the optimal $\boldsymbol{Q}$ costs $O\big(\|\boldsymbol{R}\|_0 D^2 + N(D^{3.5} + LD^2)\big)$ in one round of iteration.

In summary, the complexity of Algorithm 2 in one round of iteration is $O\big(\|\boldsymbol{R}\|_0 D^2 + (M+N)(D^{3.5} + LD^2)\big)$, and the overall DRMF-SPL generally ends in several rounds.

Algorithm 1: DRMF$_{\boldsymbol{W}}$

Input: User-item rating matrix $\boldsymbol{R}$, dimension D of Hamming space, weight $\boldsymbol{W}$
Output: User binary codes $\boldsymbol{P}$ and item binary codes $\boldsymbol{Q}$

```
Initialize P and Q using the standard normal distribution (μ = 0, σ = 0.1);
repeat
    for u ∈ {1, ··· , M} do
        for i ∈ I_u do
            Compute r̂_{u,i} ;
        Update p_u based on the BQP solver;
    for i ∈ {1, ··· , N} do
        for u ∈ U_i do
            Compute r̂_{u,i} ;
        Update q_i based on the BQP solver;
until L_W is convergent;
```

Algorithm 2: DRMF-SPL

Input: User-item rating matrix $\boldsymbol{R}$, k_0, k_{end}, $\tau \geq 1$
Output: User binary codes P and item binary codes Q

```
Initialize P^0, Q^0 ← DRMF_W given W = 1^{|S|}, and initialize
  k ← k_0, t ← 0 by calculating the all ℓ̃_uij ;
repeat
    W^{t+1} ← Eq (22);
    P^{t+1}, Q^{t+1} ← DRMF_W ;
    Evaluate the all loss ℓ̃_uij ;
    k ← k/τ, t ← t + 1 ;
until k < k_end;
```

4 EXPERIMENT

In this section, we evaluate our proposed hashing framework with the aim of answering the following questions.

(1) Does the recommendation performance of the proposed DRMF-SPL outperform the state-of-the-art hashing-based

Table 1: Pre-computation

$a_{ui} = \sum_j \lambda(\varphi_{uij}) w_{uij}$	$b_{ui} = \sum_{j,i<j} \frac{1}{2} w_{uij} - w_{uij} P_{uij}$
$\bar{\boldsymbol{p}}_{ui} = \sum_j w_{uij} \boldsymbol{p}_u$	$\tilde{\boldsymbol{q}}_{ui} = \sum_{j,i<j} \left(\frac{1}{2} w_{uij} - w_{uij} P_{uij}\right) \boldsymbol{q}_j$
$\bar{\boldsymbol{q}}_{ui} = \sum_j \lambda(\varphi_{uij}) w_{uij} \boldsymbol{q}_j$	$d_{ui} = \sum_j w_{uij} P_{uij}$

Table 2: Statistics of datasets.

Dataset	#User	#Item	#Rating	Density
MovieLens-100K	917	939	94,481	10.97%
MovieLens-10M	69,838	8,939	9,983,758	1.60%
Amazon-Books	35,151	33,195	1,732,060	0.15%

recommender systems, ranking-based recommender systems, or rating-based recommender systems?

(2) Is the self-paced learning strategy helpful to improve the recommendation performance?

(3) Whether or not the semidefinite relaxation and Gaussian randomization approximation techniques are effective in the proposed discrete optimization algorithm?

(4) What's the gap between the cross-entropy loss and its quadratic upper bound approximation used in this paper? How about the convergence of Algorithm 1?

(5) What's the advantage of hashing-based methods for online recommendation over real-valued frameworks?

In the following, we first introduce the experimental settings, followed by answering the above questions.

4.1 Experiment Settings

In this section, we first introduce datasets used in our experiments. Then we display six important baselines, including ranking-based, rating-based, and hashing-based recommendation frameworks, followed by the introduction of the evaluation metric, Normalized Discounted Cumulative Gain(NDCG) used in this paper.

4.1.1 Datasets. We do experiment with three public datasets: MovieLens-100K, MovieLens-10M, and Amazon Book dataset, to evaluate the proposed framework. In the three datasets, it is supposed that each user had only one rating for each item.

MovieLens datasets[1] are collected and made available rating data sets from the MovieLens web site[2] by GroupLens Research. MovieLens-100K originally includes 100000 ratings from 1 to 5 from 943 users on 1682 items. MovieLens-10M contains 10 million ratings and 100,000 tag applications applied to 10,681 movies by 71,567 users, and the rating scores are from 0.5 to 5 with 0.5 interval.

Amazon dataset[3] contains user ratings and reviews on Amazon of 24 product categories, such as Books, Electronics, Movies and TV, etc. We evaluate our method on one of the largest product category, Books dataset, which initially contains 8,898,041 integer ratings from 1 to 5 rated by 603,668 users to 367,982 movies.

As we evaluate our algorithm on top-10 recommendation, for all datasets, users having less than 20 ratings are not considered. The statistics of the filtered datasets are summarized in Table 2. For each user, we randomly sampled 50% ratings as training and the rest 50% ratings as testing. We repeated for ten random splits and reported the average results.

4.1.2 Comparison Methods. As introduced in Section 1 and Section 2, we choose three types of recommendation methods that consist of two ranking-based recommender systems: **BPR** and **PAIR-MF**, four real-valued recommender systems: **BPR**, **PAIR-MF**, **MF-L2**, and **PMF**, and two latest competing hashing-based recommender systems consists of a quantization-based hashing **PPH** and an optimization-based hashing **DCF**. Specifically, **PAIR-MF** is a real-valued pairwise ranking framework based on matrix factorization, which has the same objective Eq (5) with our proposed method. The difference is: **PAIR-MF** learned real latent factors from the objective Eq (5) by Newton method, while **DRMF-SPL** attain hash codes by Algorithm 2. To verify the effectiveness of the proposed hashing framework, we set the **PAIR-MF** method as a baseline.

4.1.3 Evaluation Metric. We evaluate the recommendation performance with a widely used ranking based metric, Normalized Discounted Cumulative Gain (NDCG) [18]. NDCG is the normalization of DCG(Discounted Cumulative Gain). DCG is a measure which can weigh the quality of ranking. Recommendation can be regarded as a task of ranking items. In our experiments, we predicted a top-k preference items ranking list for each user from testing datasets.

4.1.4 Parameter Settings. In our experiments, α and β are treated as two hyper-parameters of our models, because they take effect on the recommendation performance to some extent. We use 5-fold cross validation method on randomly splits of training data, to tune the optimal hyper-parameters by the grid search for the proposed DRMF-SPL and other compared algorithms, respectively.

Specifically, for DRMF-SPL, we set $\alpha = 100$ for all datasets, the optimal β depends on datasets and code lengths, we analyze the impact of β on recommendation accuracy in Figure 3.

Besides, for all baselines, we also tune the optimal parameters under the same settings. As a result, we set $\alpha = 100$ for PAIR-MF, similar to DRMF-SPL, β depends on datasets and code lengths. For BPR, we set $\alpha = 0.03$, $\lambda_i = 0.01$, and $\lambda_j = 0.01$. For MF-L2, the optimal hyper-parameter of the regularization term $\lambda = 0.009$. For PMF, it can achieve good performance when we set $momentum = 0.9$, $\lambda_U = \lambda_V = 0.9$ for Amazon-Books and $\lambda_U = \lambda_V = 0.7$ for MovieLens-100K, MovieLens-10M. For DCF, we set the optimal parameters $\alpha = 1e^{-3}$ and $\beta = 1e^{-3}$, respectively. For PPH, we set $\lambda = 10$.

4.2 Accuracy Comparison with Baselines

As introduced in Section 1, hashing recommendation has the significant advantage over real-valued recommendation for online recommendation, however, hashing usually suffers low recommendation accuracy due to a binary value instinctively carries less information than a real value. How to balance the recommendation accuracy and efficiency is challenging for a recommendation.

[1] https://grouplens.org/datasets/movielens/
[2] http://movielens.org
[3] http://jmcauley.ucsd.edu/data/amazon/

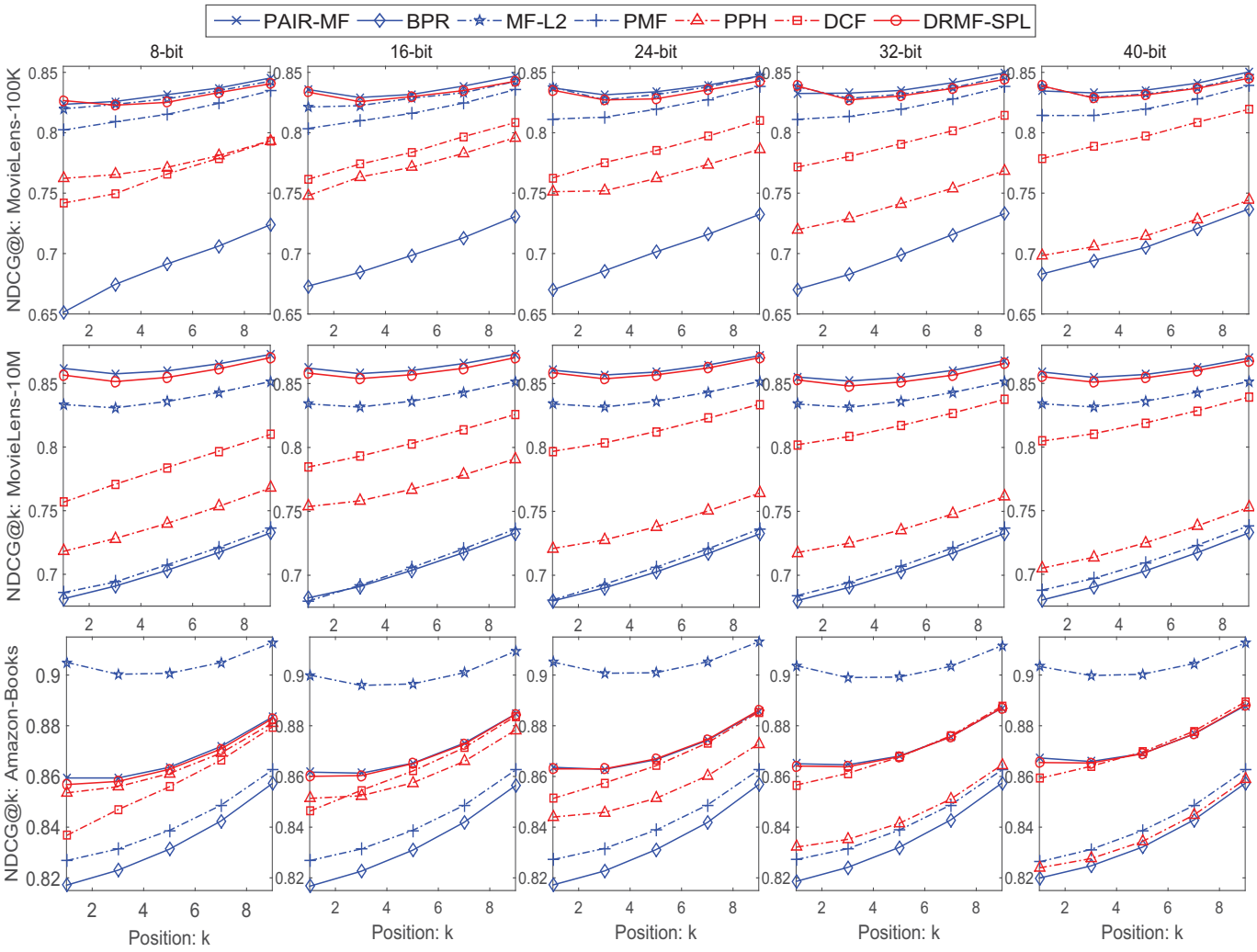

Figure 1: Recommendation performances on MovieLens-100K, MovieLens-10M, Amazon-Books.

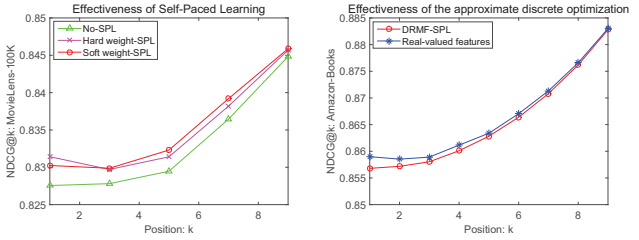

Figure 2: Left. Effectiveness of self-paced learning on the MovieLens-100K; Right. Effectiveness of the approximate discrete optimization on the Amazon-Books

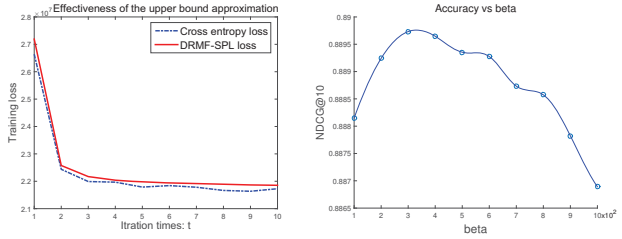

Figure 3: Left. The training loss comparison of DRMF-SPL and cross-entropy. Right. The effect of β on recommendation accuracy on the Amazon-Books ($\alpha = 100$, $d = 8$).

DRMF-SPL is proposed to search a better trade-off between efficiency and accuracy with hashing technology.

In this part, we will answer the first question at the beginning of Section 4. The recommendation accuracy (NDCG@k) comparisons on three datasets are shown in Figure 1 respectively. The performances of real-valued and hashing based recommender systems are plotted as blue and red lines, respectively. The performances of

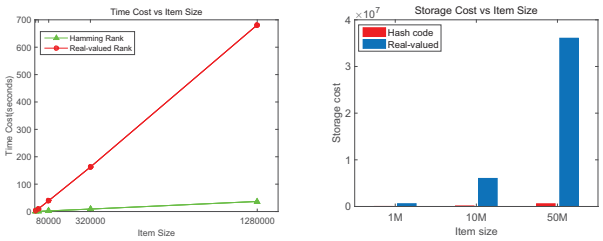

Figure 4: Efficiency results on artificial data. Left. The comparison of time cost. Right. The comparison of storage cost

ranking-based and rating-based recommender systems are plotted as solid and dotted lines, respectively.

Compared with hashing recommendation, the performance of DRMF-SPL far surpasses PPH on all datasets since DRMF-SPL extract binary codes by directly solving a discrete optimization problem instead of quantification method. DRMF-SPL is superior to DCF on MovieLens 100K and MovieLens 10M, since DRMF-SPL is proposed to optimize a ranking-based loss instead of a rating-based loss considered in DCF. The performance of DCF approaches to DRMF-SPL on Amazon-Books as the code length increases, which accounts for the high sparsity of Amazon-Books. Because DCF is adapted to sparse settings on account of the balanced and un-correlation constraints on hash codes.

Compared with ranking-based real-valued recommender systems, DRMF-SPL has evident advantages over BPR. Because BPR is based on implicit feedback other than explicit feedback such as ratings exploited in the proposed method. DRMF-SPL has very close performance with PAIR-MF since they have the same ranking-based objective. In spite of existing difference between DRMF-SPL and PAIR-MF, but its not significant. Due to real latent vectors intuitively carried more information than hash codes. Thus it is acceptable and reasonable to have small gaps between real-valued PAIR-MF and hashing based DRMF-SPL. Furthermore, DRMF-SPL has superiority in the efficiency of online recommendation over real-valued recommender systems, which will be demonstrated in Section 4.6.

Compared with other state-of-the-art methods, apart from the efficiency advantage for online recommendation, DRMF-SPL has the advantage of recommendation accuracy over traditional methods in some cases. From Figure 1, the accuracy of DRMF-SPL is superior to the traditional MF-L2, PMF, BPR, and approaches to PAIR-MF. The good performance of DRMF-SPL attributes to a ranking based objective, and a series effective learning strategies. We will explore the effectiveness of these strategies in the following experiments.

4.3 Effectiveness of Self-paced Learning

For answering the second question proposed at the beginning of Section 4, self-paced learning is also helpful to our algorithm. The left of Figure 2 shows the performance of three different learning strategies. It says that the linear "soft" weight SPL [30] can achieve a better performance than the "hard" weight SPL [11] in our algorithm. We can also obtain the answer of question (2) that self-paced learning can improve the recommendation performance.

4.4 Effectiveness of the Block-wise Hashing Framework

In Section 3, we learn hash codes in a block-wise way. Specifically, we first solve an SDR problem, which is the relaxation of the BQP problem, we then attain hash codes by the Gauss randomization method. To verify these approximation strategies are reliable, we relax the BQP problem to a real-valued QP problem, and then real-valued closed-form solutions can be obtained straightly. By comparing the performance of the real-valued features and the derived hash codes, we can validate whether or not these approximation techniques are effective in our algorithm.

The right of Figure 2 shows the performance between hash codes and the real-valued features on Amazon-Books dataset, which demonstrates the effectiveness of these approximation techniques, and thus answers the question (3). It tells us the performance difference is small in total, and the most significant difference is about 0.002. As the number of recommendation items increases, the difference becomes smaller. So our block-wise hashing optimization framework is valid.

4.5 Gap Between the Cross-entropy Loss and the DRMF-SPL Loss

In this section, we will answer the question (4) proposed at the beginning of Section 4. Due to the objective of DRMF-SPL does not straightly optimize the cross-entropy loss function, but turn to minimize a variational quadratic upper bound. So it is necessary to test the gap between the upper bound and the cross-entropy in the training procedure. In Figure 3, we train the proposed model by Algorithm 2 on Amazon-Books dataset, and record both the loss of DRMF-SPL and the cross entropy loss. We find the gap between the two loss functions is small, which indicates the upper bound approximation is a good approximation of the cross entropy loss in the proposed model. Therefore, it guarantees a good recommendation accuracy. Besides, we can conclude that the proposed optimization method can be converged within serval iterations, which furtherly ascertain the correctness of the optimization algorithm.

4.6 Efficiency Comparison

As analyzed in Section 1, the significant advantage of hashing recommendation over real-valued recommendation is the efficiency for online recommendation. In this section, we answer the question (5) and display the effectiveness of hashing-based recommendation compared with real-valued frameworks. We separately evaluate the efficiency in terms of time and storage on a synthetic dataset.

4.6.1 Time Complexity. We investigate the time cost of preference ranking when the item number varies. We use standard Gaussian distribution to generate real-valued features of items randomly. Items hash codes are obtained from real-valued vectors by sign function. We set different item numbers in our experiment: 5000, 20000, 80000, 320000, 128000, to test the ranking time variation. The time cost variation over item number is shown in the left of Figure 4. We can conclude that the time cost of real-valued features grows fast with the items number, in comparison, the time cost of hash codes increases much slower than real-valued features.

4.6.2 Storage Complexity. We test the storage costs of hash codes and real-valued features on three different sizes of item sets: 1 million, 10 million, 50 million. From the right of Figure 4, hash codes cost much less memory to store the same number of items than real-valued features, which is consistent with the analysis in Section 1.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose discrete ranking-based matrix factorization, to match the ultimate goal of the recommender system. For the sake of learning each binary code as a whole instead of bit-by-bit, we formulate it into a series of binary quadratic programming problems. Leveraging semidefinite programming and Gaussian randomization, we can obtain binary codes of higher quality, since the semidefinite relaxation of binary quadratic programming incurs smaller loss. The experiments on three real-world rating datasets reveal that the proposed algorithm can be significantly better than the state-of-the-art hashing based recommendation algorithms, particular on the denser datasets. However, the drawback of the proposed algorithm is the reliance of BQP solvers, but current BQP solvers are low efficiency with high dimensional BQP problems due to the time complexity of $O(n^{3.5})$. For improving the efficiency of updating high dimensional hash codes, in the future, we will try to apply an effective hashing technique, Hamming Subspace Learning (HSL) [21], to learn hash codes efficiently on massive social datasets.

A BINARY QUADRATIC PROGRAMMING

BQP is a classical combinatorial optimization problem [20], which minimizes a quadratic function, with respect to binary variables, i.e.,

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \\ &\text{s.t. } x_d^2 = 1, d = 1\cdots, n \end{aligned} \tag{23}$$

where $\boldsymbol{C}$ is a real symmetric matrix. The BQP is well known to be a computationally difficult problem, particularly belonging to the class of NP-hard problems. Hence, computing good solutions is a quite difficult task. Semidefinite relaxation (SDR) technique is a powerful computationally efficient approximation technique for a host of very difficult optimization problem including binary quadratic programming problems. A crucial first step in deriving a SDR of BQP in Eq (23) is to observe that $\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} = \text{tr}(\boldsymbol{C}\boldsymbol{x}\boldsymbol{x}^T)$. Introducing a new variable $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^T$ and noting that $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^T$ is equivalent to $\boldsymbol{X}$ being a rank one symmetric positive semidefinite (PSD) matrix, we can obtain the following equivalent formulation

$$\begin{aligned} &\min_{\boldsymbol{X}\in\mathbb{S}^n} tr(\boldsymbol{C}\boldsymbol{X}) \\ &\text{s.t. } X_{d,d} = 1, \boldsymbol{X} \succcurlyeq 0, \text{rank}(\boldsymbol{X}) = 1. \end{aligned} \tag{24}$$

where $\mathbb{S}^n$ denotes the set of symmetric $n \times n$ matrices, $\boldsymbol{X} \succcurlyeq 0$ indicates that $\boldsymbol{X}$ is PSD. The re-formulation allows us to identify the difficulty of solving BQP, that is, the rank constraint $\text{rank}(\boldsymbol{X}) = 1$. Thus, we may drop it to obtain the following semidefinite relaxation (SDR) of BQP:

$$\begin{aligned} &\min_{\boldsymbol{X}\in\mathbb{S}_+^n} \text{tr}(\boldsymbol{C}\boldsymbol{X}) \\ &\text{s.t. } X_{d,d} = 1. \end{aligned} \tag{25}$$

Algorithm 3: Gaussian randomization: $\boldsymbol{x}^* = \text{Round}(\boldsymbol{X}^*)$

Input: The SDR solution $\boldsymbol{X}^*$, the number of randomization L

Output: $\boldsymbol{x}^*$ the best approximated feasible solution

```
for ℓ = 1, 2, ··· , L do
    generate ξ_ℓ ~ N(0, X*);
    construct a feasible point x_ℓ ← sign(ξ_ℓ);
determine ℓ* ← arg min_{ℓ∈{1,···,L}} x_ℓ^T C x_ℓ;
x* ← x_{ℓ*} as the best approximated solution of BQP;
```

where $\mathbb{S}_+^n$ denotes the set of symmetric positive semidefinite $n \times n$ matrices. In spite of dropping the rank one constraint, SDR still provides an approximation of BQP with a very high accuracy[20]. Moreover, SDR can then be solved, to any arbitrary accuracy in a numerically reliable and efficient fashion, using like interior-point algorithms [7] with time complexity of $O(n^{3.5})$. However, there remains an issue of using SDR, that is how to convert a globally optimal solution of Eq (25) into a feasible solution to Eq (23).

If the optimal $\boldsymbol{X}^*$ is of rank one, then $\boldsymbol{X}^* = \boldsymbol{x}^*\boldsymbol{x}^{*T}$, and $\boldsymbol{x}^*$ is the optimal solution of BQP. Otherwise, Gaussian randomization procedure can be applied as shown in Algorithm 3 due to its probabilistic interpretation. In other words, $\boldsymbol{X}^*$ is also the optimal solution to the stochastic BQP:

$$\begin{aligned} &\min_{\boldsymbol{X}\in\mathbb{S}_+^n} E_{\xi\sim N(\boldsymbol{0},\boldsymbol{X})}[\boldsymbol{\xi}^T\boldsymbol{C}\boldsymbol{\xi}] \\ &\text{s.t. } E_{\boldsymbol{\xi}\sim N(\boldsymbol{0},\boldsymbol{X})}[\xi_i^2] = 1 \end{aligned} \tag{26}$$

Here sampling from normal distribution with a specified variance can be achieved by the following two steps:

(1) eigenvalue decomposition of variance $\boldsymbol{X}$, i.e., $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$, with the complexity of $O(n^3)$.
(2) drawing L samples $\{\boldsymbol{\xi}_l\}_{l\in\{1,\cdots,L\}}$ with the complexity of $O(Ln^2)$ by $\boldsymbol{\xi}_l = \boldsymbol{V}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{\epsilon}_l$, where $\boldsymbol{\epsilon}_l$ is generated from $N(\boldsymbol{0}, \boldsymbol{I}_n)$.

Therefore, the complexity of Gaussian randomization is $O(n^3 + Ln^2)$.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (Grant No. 61572335, 61572109, 61502077, 61631005) and the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2016J087). Prof. Ivor W. Tsang was supported by ARC FT130100746, DP180100106 and LP150100671. Dr. Hongzhi Yin was supported by ARC DE160100308, DP170103954, and New Staff Research Grant of the University of Queensland (Grant No.613134).

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] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of WWW'17*. International World Wide Web Conferences Steering Committee, 1341–1350.
[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of ICML'05*. ACM, 89–96.
[4] Wei Chen, Wynne Hsu, and Mong Li Lee. 2013. Modeling user's receptiveness over time for recommendation. In *Proceedings of SIGIR'13*. ACM, 373–382.

[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] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *Proceedings of KDD'15*. ACM, 189–198.

[7] Christoph Helmberg, Franz Rendl, Robert J Vanderbei, and Henry Wolkowicz. 1996. An interior-point method for semidefinite programming. *SIAM Journal on Optimization* 6, 2 (1996), 342–361.

[8] David R Hunter and Kenneth Lange. 2004. A tutorial on MM algorithms. *The American Statistician* 58, 1 (2004), 30–37.

[9] Hemant Ishwaran and Lancelot F James. 2001. Gibbs sampling methods for stick-breaking priors. *J. Amer. Statist. Assoc.* 96, 453 (2001), 161–173.

[10] 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.

[11] Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G Hauptmann. 2014. Easy samples first: Self-paced reranking for zero-example multimedia search. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 547–556.

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

[13] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[14] M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Proceedings of NIPS'10*. 1189–1197.

[15] M Pawan Kumar, Haithem Turki, Dan Preston, and Daphne Koller. 2011. Learning specific-class segmentation from diverse data. In *Proceedings of ICCV'11*. IEEE, 1800–1807.

[16] Changsheng Li, Fan Wei, Junchi Yan, Xiaoyu Zhang, Qingshan Liu, and Hongyuan Zha. 2016. A Self-Paced Regularization Framework for Multilabel Learning. *IEEE Transactions on Neural Networks and Learning Systems* 99 (2016), 1–7.

[17] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of KDD'17*. ACM, 325–334.

[18] T.Y. Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

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

[20] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. 2010. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine* 27, 3 (2010), 20–34.

[21] Chao Ma, Ivor W Tsang, Furong Peng, and Chuancai Liu. 2017. Partial hash update via hamming subspace learning. *IEEE Transactions on Image Processing* 26, 4 (2017), 1939–1951.

[22] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 3108–3115.

[23] 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.

[24] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised Discrete Hashing. In *CVPR*. 37–45.

[25] Yue Shi, Martha Larson, and Alan Hanjalic. 2010. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of RecSys'10*. ACM, 269–272.

[26] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. 2004. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of WWW'04*. ACM, 675–684.

[27] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2012. Semi-supervised hashing for large-scale search. *IEEE TPAMI* 34, 12 (2012), 2393–2406.

[28] 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.

[29] 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.

[30] Chang Xu, Dacheng Tao, and Chao Xu. 2015. Multi-view Self-Paced Learning for Clustering.. In *IJCAI*. 3974–3980.

[31] 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.

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

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

[34] Qian Zhao, Deyu Meng, Lu Jiang, Qi Xie, Zongben Xu, and Alexander G Hauptmann. 2015. Self-Paced Learning for Matrix Factorization.. In *AAAI*. 3196–3202.

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