Product Quantized Collaborative Filtering

Defu Lian, Xing Xie, Enhong Chen and Hui Xiong

Abstract—Because of strict response-time constraints, efficiency of top-k recommendation is crucial for real-world recommender systems. Locality sensitive hashing and index-based methods usually store both index data and item feature vectors in main memory, so they handle a limited number of items. Hashing-based recommendation methods enjoy low memory cost and fast retrieval of items, but suffer from large accuracy degradation. In this paper, we propose product Quantized Collaborative Filtering (pQCF) for better trade-off between efficiency and accuracy. pQCF decomposes a joint latent space of users and items into a Cartesian product of low-dimensional subspaces, and learns clustered representation within each subspace. A latent factor is then represented by a short code, which is composed of subspace cluster indexes. A user's preference for an item can be efficiently calculated via table lookup. We then develop block coordinate descent for efficient optimization and reveal the learning of latent factors is seamlessly integrated with quantization. We further investigate an asymmetric pQCF, dubbed as QCF, where user latent factors are not quantized and shared across different subspaces. The extensive experiments with 6 real-world datasets show that pQCF significantly outperforms the state-of-the-art hashing-based CF and QCF increases recommendation accuracy compared to pQCF.

Index Terms—Recommendation, Product Quantization, Collaborative Filtering, Maximum Inner Product Search

1 Introduction

Recommender systems aim at finding a short list of items to be consumed with the highest chance. They have been widely used in many online services for dealing with information overload. Matrix factorization (MF) is one of the most popular recommendation methods, even with the recent development of deep-learning-based recommendation. By using advanced loss functions, MF can show very competitive recommendation performance [1], [2]. Moreover, MF has been extended for incorporating side information by applying deep learning for processing the side information [3], [4], [5]. Their recommendation performance can be comparable to other deep-learning-based recommendation methods, such as DeepFM [6] and XDeepFM [7]. Therefore, in this paper, we only focus on the MF-based recommendation models. Another reason of this choice is that the MF models are more efficient to train and predict.

In MF modes, both users and items are represented by points in a joint latent space. The score of matching between a user i represented by $\boldsymbol{p}_i$ and an item j represented by $\boldsymbol{q}_j$ is estimated by the inner product $\langle \boldsymbol{p}_u, \boldsymbol{q}_j \rangle = \boldsymbol{p}_u^T \boldsymbol{q}_j$. A higher score indicates user's higher preference for the item. Given a user, the top-k recommendation task selects k items with the largest scores among N candidate items. The time complexity is $\mathcal{O}(k \log k + NK)$, where K is the dimension of latent space. The massive growth of users and items in online services gives rise to the challenge of efficient top-k recommendation. In most recommender systems, user interest evolves over time, so that it is not a good choice to precompute top-k items, and then store them into a database. This paper studies the scalability of the top-k recommendation based on matrix factorization.

To instantly generate recommendation, locality-sensitive hashing (LSH) and index-based methods have been widely used in practical scenarios such as news recommendation [8] and movie recommendation [9]. Note that the inner product generally violates the triangle inequality, so maximum inner product search should be transformed to nearest neighbor search [10], [11], [12]. Index-based methods, such as KD-tree and metric tree [13], are usually better than LSH for real data because of the usage of data distribution. However, both LSH and index-based methods are required to perform a fine re-ranking step based on exact distance, so that in addition to index structure, item feature vectors are also stored in main memory. This constraint restricts the number of items to handle. Recently, hashing-based recommendation methods are proposed for efficient collaborative filtering [14], [15], [16], [17], [18] with limited memory usage. These algorithms directly learn short binary codes from rating/preference data represented by a user-item matrix $\boldsymbol{R}$, and efficiently estimate preference scores via hamming distance. However, these algorithms suffer from challenge of optimization and large quantization errors.

Motivated by much lower approximation errors of product quantization (PQ) than hashing [19], [20], in this paper, we propose product Quantized Collaborative Filtering (pQCF) to construct short codes for users and items. pQCF decomposes the joint space of users and items formed by matrix factorization into a Cartesian product of lower-dimensional subspaces, and learns clustered representation within each subspace, as demonstrated the left (training) part of Fig. 1. As a result, a latent feature vector of either user or item is represented by a short code composed of subspace cluster indexes. The representation capacity is then much larger than binary Hamming space. For the sake of illustration, these cluster indexes are represented by one-hot indicators in Fig. 1. In fact, each cluster index can be more compactly encoded using $\log C$ bits if there are C clusters in a subspace. By constructing lookup tables between cluster

- *Defu Lian and Enhong Chen are with the School of Computer Science and Technology, University of Science and Technology of China. Email: {liandefu,cheneh}@ustc.edu.cn.*
- *Xing Xie is with Microsoft Research Asia. Email: xingx@microsoft.com.*
- *Hui Xiong is with the Department of Management Science and Information Systems, Rutgers University. Email:hxiong@rutgers.edu.*

centers within each subspace, the preference score can be efficiently computed with a few additions, as shown in the right (prediction) part of Fig. 1. Therefore, time consumption is comparable to hamming distance in hashing-based CF. The storage cost is also close to hashing-based CF, with a few extra spaces for storing lookup tables.

It is straightforward to apply PQ for quantizing user latent factors and item latent factors separately. However, as evidenced by empirical results, this quantizer can be significantly improved since PQ is based on the Euclidean distance rather than the inner product [19]. Though PQ is subsequently extended for maximum inner product search [21], quantization is only performed over item latent factors learned from matrix factorization. Different from this extension, pQCF only takes rating/preference data as input and unifies quantization and the learning of latent factors into a joint framework. This can be better tailored for collaborative filtering with only rating data[1] provided. In spite of this, pQCF can be still effectively optimized using block coordinate descent. The time complexity of optimization in each iteration is also linearly proportional to the number of ratings, not higher than the matrix factorization algorithms.

To summarize, the main contributions are four-fold:

- We propose product Quantized Collaborative Filtering (pQCF) to learn semi-structured latent factors for users and items from rating data. This essentially extends product quantization from the Euclidean space to the inner product space, and strikes a better balance between efficiency and accuracy of item recommendation.
- We propose an efficient block coordinate descent algorithm for parameter learning. The time complexity of each iteration is only linearly proportional to the number of ratings. We also reveal how pQCF integrates the learning of latent factors into quantization.
- We extend pQCF to an asymmetric version – QCF, where only item latent factors are quantized and user latent factors are shared across different subspaces. QCF is shown to dramatically increase recommendation accuracy due to approximating the inner product more precisely.
- We evaluate the proposed algorithms with six real-world explicit or implicit datasets. The empirical results demonstrate the significant superiority of the proposed algorithms to the state-of-the-art hashing-based CF methods with comparable retrieval time and a few extra memories. pQCF also achieves 30x speedup for top-k recommendation tasks with negligible accuracy degradation.

2 Related Work

This work targets for striking a better balance between efficiency and accuracy of item recommendation. We first review recent advance of non-sequential recommendation models in improving recommendation accuracy. Then, we turn to recommendation efficiency. Since the preference score is estimated by the inner product between latent factors, it is closely related with the research of maximum inner product search (MIPS) given latent feature vectors of users and items. It is also closely related with hashing-based collaborative filtering given users' rating data.

1. Below rating/preference data are collectively denoted rating data.

2.1 Recommendation Models

The recent advance of non-sequential recommender systems can be classified into three categories. The first taxonomy is to design new loss functions for recommendation models, such as the Bayesian Personalized Ranking (BPR) loss [22], [23], Gravity Regularizer [24], [25], [26], Weighted Approximate-Rank Pairwise (WARP) loss [27], Ranking-based implicit regualrizer [28], CliMF [29], Sampled Softmax [3]. The second taxonomy is to design preference functions, which are based on dot product, Euclidean distance [1], multilayer perception [30], or cosine similarity [3]. The final taxonomy is to model feature interaction, whose representative models include PNN [31], Deep&Cross [32], Wide&Deep [33], XDeepFM [7] and GCN based models [34], [35]. In spite of the recent advance of recommender systems, MF with the advanced loss functions plays an important part in collaborative filtering, due to its competitive recommendation accuracy and superior computational efficiency.

2.2 Maximum Inner Product Search

The MIPS problem has been studied for many years and attracts much renascent attention recently. The challenge of the MIPS problem is that the inner product violates the basic axioms of a metric, such as triangle inequality and minimality. Several works try to transform MIPS to nearest neighbor search (NNS) approximately [10], [36] or exactly [11], [12], [37]. Note that if the database vectors are of the same norm, MIPS is equivalent to NNS. Therefore, the key idea of the transformation lies in augmenting database vectors to ensure them an (nearly) identical norm. The differences of these works also include the transformation of query vectors, such as ensuring their norm identical to database vectors [12], [37] or keeping them unchanged [11], [36]. Though scaling query vectors does not affect the performance of retrieval, it leads to different distortion errors [12]. Following the transformation, a bulk of algorithms can be applied for ANN search, such as Euclidean Locality-Sensitive Hashing [38], Signed Random Projection [36], PCA-Tree [11], Hierarchical K-means trees [1], [39], [40]. Alternatively, MIPS can be accelerated by branch and bound algorithms with the metric tree [13] and clustering [41], by the threshold algorithms [42], or an attribute pruning-based algorithms [43].

Different from these works, we study the MIPS problem from the perspective of quantization, and unify quantization with the learning of latent factors. Therefore, there is no need of transforming MIPS to NNS any more. In fact, the proposed algorithm can be integrated with these advanced techniques for approximated MIPS. Several existing works also studied quantization-based MIPS by exploiting additive nature of inner product, such as additive quantization [44], composite quantization [45] and residual quantization [46]. However, they belong to alternative quantization to product quantization, without specific consideration of the problems of the inner product. Another work about quantization-based MIPS extended PQ from the Euclidean distance to the inner product [21]. However, they only took some query (user) latent factors as held-out samples for training, and did not take rating data into consideration. Therefore, it is totally different from our settings, where

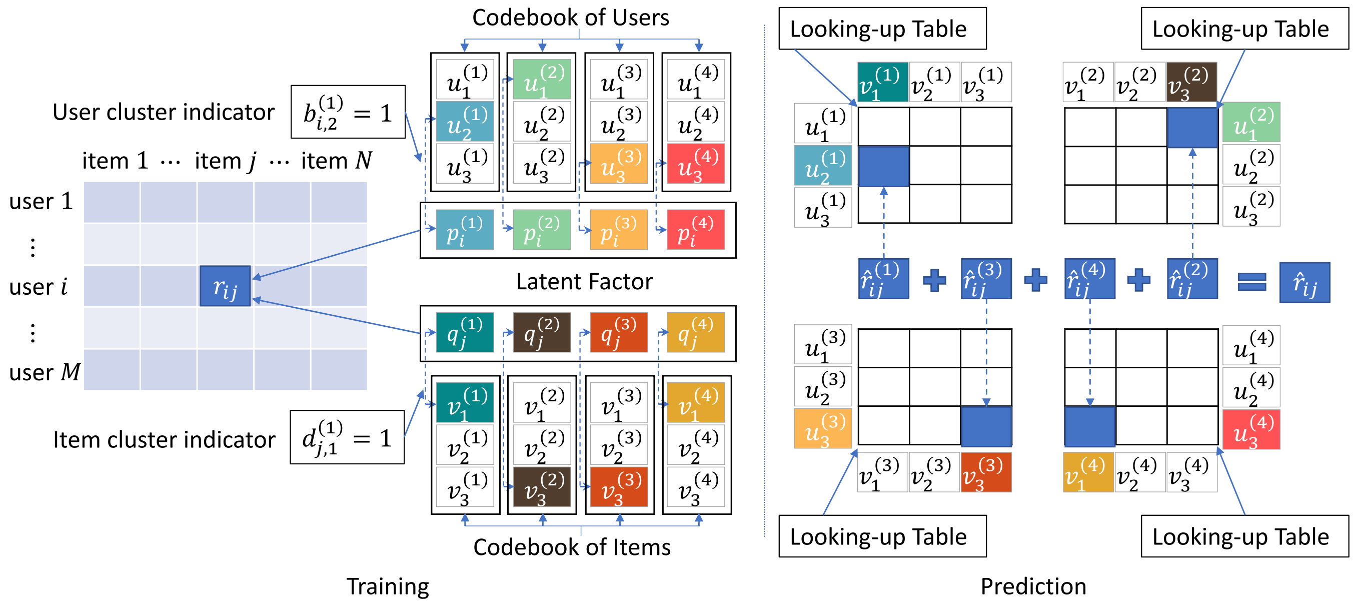

Fig. 1. The framework of product quantized collaborative filtering.

rating data is taken as input, and quantization is unified with the learning of latent factors of both users and items.

2.3 Hashing-based Collaborative Filtering

Hashing-based collaborative filtering learns hash codes from rating data and (or) side information. In other words, they are data-dependent, in contrast data-independent hashing such as L2LSH [14] and MinHash [8], [47], [48]. Hashing-based collaborative filtering can be organized into two categories: two-stage hashing and discrete hashing.

Two-stage hashing methods separate binarization from the learning of latent factors. For example, Zhou and Zha utilized iterative quantization (ITQ) to learn binary codes from user/item latent factors, which are learned from matrix factorization [15]. In order to ensure hash codes as compact as possible, a decorrelated constraint was imposed over latent factors before quantization [49]. As observed by [16], quantization results in the loss of magnitude information of latent factors, so existing methods only preserve similarity rather than the inner product between the user and the item. Therefore, they proposed to impose a constant feature-norm constraint on latent factors, and then quantized magnitude and similarity separately.

Discrete hashing methods aim to reduce large quantization errors from which the two-stage hashing methods suffer. For example, Zhang et al. proposed to directly learn binary codes for users and items in matrix factorization by cyclic coordinate descent [17]. To ensure hash codes informative and compact, the balanced and decorrelated constraints were imposed. In order to adapt from rating data (explicit feedback) to implicit feedback, Zhang et al. proposed to optimize pairwise ranking loss between each user's interacted items and other items [50]. Lian et al. proposed a unified framework for both explicit and implicit datasets by introducing an interaction regularization [18], [51] and further incorporated auxiliary information by a regression-based method.

Different from these works, the proposed algorithm learns compact codes for collaborative filtering from the perspective of quantization. As observed from experimental results, this leads to reduction of quantization errors and substantial improvements of recommendation accuracy.

3 Preliminary

Before introducing pQCF, we first introduce some notations and review some background. Denote M the number of users, N the number of items, K code length, D the dimension of each subspace, F the number of subspaces, C the number of clusters within each subspace. Note that the number of clusters across subspaces is assumed identical. The code length K is distinguished from k, the number of items to be recommended. Let i index a user, j index an item, c index a cluster, f index a subspace. Denote $\boldsymbol{R}$ a user-item rating matrix, and denote $\boldsymbol{p}_i, \boldsymbol{q}_j \in \mathbb{R}^K$ latent factor of a user i, an item j, respectively. Other notations are introduced in the context.

3.1 Matrix Factorization for Top-k Item Recommendation

Recommender system first concentrated on rating prediction based on rating data. Many algorithms have been developed particularly at the time of Netflix prize. Item recommendation has been started in implicit feedback [22], [24], since the algorithms tailored for rating prediction perform poorly in this case. One simple yet effective method is weighted regularized matrix factorization. In this model, missing values are considered zero-rated, but zero ratings are assigned a much lower confidence than observed ones. In fact, a regularizer is imposed to penalize non-zero estimation of preference scores [25]. Interestingly, a similar model has also been proposed for explicit feedback [52], where zero ratings are considered prior on missing values. Since this model is suitable for item recommendation based on both explicit and implicit feedback, we choose it as our base model and dub it *MF*. Formally, MF optimizes the following objective function

$$\min_{\boldsymbol{p}_i,\boldsymbol{q}_j\in\mathbb{R}^K} \sum_{i,j} w_{ij}(r_{ij} - \boldsymbol{p}_i^T\boldsymbol{q}_j)^2 + \lambda(\|\boldsymbol{P}\|^2 + \|\boldsymbol{Q}\|^2), \quad (1)$$

where w_{ij} equals to $\alpha + 1(\alpha \gg 0)$ if r_{ij} is observed and 1 otherwise. $\boldsymbol{P} \in \mathbb{R}^{K\times M}$ is a matrix stacking user latent factors by column and $\boldsymbol{Q} \in \mathbb{R}^{K\times N}$ is a matrix stacking item latent factors. The first part can be decomposed into a well-known rating prediction loss and a regularization term $\sum_{(i,j)\notin\Omega}(\boldsymbol{p}_i^T\boldsymbol{q}_j)^2$.

3.2 Product Quantization

Product quantization is proposed for nearest neighbor search. It decomposes feature space into the Cartesian product of subspaces and performs k-means clustering in each subspace. Each feature vector is represented by a short code composed of cluster indexes in each subspace. The efficient computation of distance between feature vectors based on lookup tables enables faster nearest neighbor search. Below we introduce product quantization for item feature vectors. Denote item feature vector $\boldsymbol{q} \in \mathbb{R}^K$ as the concatenation of F subvectors of equal length D, $\boldsymbol{q} = [\boldsymbol{q}^1, \cdots, \boldsymbol{q}^f, \cdots, \boldsymbol{q}^F]$. The subvectors are quantized separately using F distinct quantizers. Formally, the f-th quantizer is a function h_f mapping a D-dimensional vector $\boldsymbol{q}^f \in \mathbb{R}^D$ to a codeword in codebook $\mathcal{V}^f$, which is defined as $\mathcal{V}^f = \{\boldsymbol{v}_c^f | c \in \{1, \cdots, C\}\}$. The Cartesian product $\mathcal{V} = \mathcal{V}^1 \times \cdots \times \mathcal{V}^F$ is the set in which a codeword $\boldsymbol{v} \in \mathcal{V}$ is also formed by concatenating the F codewords : $\boldsymbol{v} = [\boldsymbol{v}^1, \cdots, \boldsymbol{v}^f, \cdots, \boldsymbol{v}^F]$. The objective function of PQ is formulated as follows:

$$\min_{\mathcal{V},\boldsymbol{d}} \sum_{\boldsymbol{q}} \sum_f \|\boldsymbol{q}^f - \boldsymbol{V}^f\boldsymbol{d}^f\|^2 \quad (2)$$

The c-th column of $\boldsymbol{V}^f \in \mathbb{R}^{D\times C}$ corresponds to the c-th codeword. $\boldsymbol{d}^f$ is an one-hot vector, indicating to which codeword $\boldsymbol{q}^f$ is assigned. $\boldsymbol{d} = [\boldsymbol{d}^1, \cdots, \boldsymbol{d}^f, \cdots, \boldsymbol{d}^F]$ is a long binary vector, comprising codeword assignments of all subspaces. This objective function can be split into F independent subproblems, each of which corresponds to a k-means problem.

User latent factors are similarly quantized into the Cartesian product codebook $\mathcal{U} = \mathcal{U}^1 \times\cdots\times \mathcal{U}^F$ with the same settings. After that, the simple baseline for product quantized collaborative filtering is ready. Notice that PQ was improved by optimizing space decomposition [20], [53] so it is intuitive to apply it for improving quantization. However, optimizing space decomposition separately is not reasonable, as axises are not aligned with each other after independent rotation. This may lead to meaningless inner product. The solution to this problem is introduced in next section.

4 Optimized Product Quantization for Collaborative Filtering

To improve the baseline that simply applies PQ for independently quantizing user latent factors and item latent factors, we can jointly optimize quantization and space decomposition. According to [20], [53], optimal space decomposition is important for ANN search, so it should be also important for MIPS. As discussed, optimizing space decomposition separately for users and items leads to unaligned axises, so that it is meaningless to operate inner product between them. To this end, we rotate the joint latent space with an orthogonal matrix $\boldsymbol{H}$. Note that the preference scores do not change with such a rotation. In particular,

$$\hat{r}_{ij} = (\boldsymbol{H}\boldsymbol{p}_i)^T(\boldsymbol{H}\boldsymbol{q}_j) = \boldsymbol{p}_i^T\boldsymbol{H}^T\boldsymbol{H}\boldsymbol{q}_j = \boldsymbol{p}_i^T\boldsymbol{q}_j \quad (3)$$

Since Frobenius norm is unitarily invariant, rotating joint latent space does not alter optimality. In other words, if $(\boldsymbol{P},\boldsymbol{Q})$ is an optimal solution of Eq (1), $(\boldsymbol{HP},\boldsymbol{HQ})$ is also optimal with the same loss. Therefore, OPQ should be adaptive to collaborative filtering based on the following optimization:

$$\min_{\Theta,\boldsymbol{H}} \|\boldsymbol{P} - \boldsymbol{H}^T\tilde{\boldsymbol{U}}\boldsymbol{B}\|_F^2 + \|\boldsymbol{Q} - \boldsymbol{H}^T\tilde{\boldsymbol{V}}\boldsymbol{D}\|_F^2 \quad (4)$$

where $\Theta = \{\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}}, \boldsymbol{B}, \boldsymbol{D}\}$. $\tilde{\boldsymbol{U}}$ and $\tilde{\boldsymbol{V}}$ are block diagonal, $\boldsymbol{B}$ comprises the codeword assignments of users and $\boldsymbol{D}$ comprises the codeword assignments of items. $\tilde{\boldsymbol{V}}$ and $\boldsymbol{D}$ are defined as follows:

$$\tilde{\boldsymbol{V}} = \begin{bmatrix} \boldsymbol{V}^1 & 0 & \cdots & 0 \\ 0 & \boldsymbol{V}^2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{V}^F \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} \boldsymbol{d}_1^1 & \boldsymbol{d}_2^1 & \cdots & \boldsymbol{d}_N^1 \\ \boldsymbol{d}_1^2 & \boldsymbol{d}_2^2 & \cdots & \boldsymbol{d}_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{d}_1^F & \boldsymbol{d}_2^F & \cdots & \boldsymbol{d}_N^F \end{bmatrix}, \quad (5)$$

$\tilde{\boldsymbol{U}}$ and $\boldsymbol{B}$ are similarly defined and are not shown here.

Given the orthogonal matrix $\boldsymbol{H}$, the optimization problem is the same as PQ. Given $\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{V}}, \boldsymbol{B}, \boldsymbol{D}$, the optimization with respect to $\boldsymbol{H}$ is equivalent to the following problem

$$\max_{\boldsymbol{H}} \text{trace}\Big(\boldsymbol{H}^T(\tilde{\boldsymbol{U}}\boldsymbol{B}\boldsymbol{P}^T + \tilde{\boldsymbol{V}}\boldsymbol{D}\boldsymbol{Q}^T)\Big), \\ s.t. \boldsymbol{H}^T\boldsymbol{H} = \boldsymbol{I} \text{ and } \boldsymbol{H}\boldsymbol{H}^T = \boldsymbol{I} \quad (6)$$

This is the Orthogonal Procrustes problem [54] and there is a closed-form solution: first apply SVD on $\tilde{\boldsymbol{U}}\boldsymbol{B}\boldsymbol{P}^T + \tilde{\boldsymbol{V}}\boldsymbol{D}\boldsymbol{Q}^T = \boldsymbol{X}\boldsymbol{\Sigma}\boldsymbol{Y}^T$ and then let $\boldsymbol{H} = \boldsymbol{X}\boldsymbol{Y}^T$. The overall optimization alternates between learning $\boldsymbol{H}$ and learning Θ until convergence. This algorithm is dubbed as OPQ_CF. If only item latent factors are quantized via OPQ, this turns to an asymmetric OPQ, dubbed as OPQ_CF+. The optimal rotate matrix from OPQ will multiply user latent factors, to align them with items in the same space and enable valid inner product between them.

5 Product Quantized Collaborative Filtering

Both OPQ_CF and PQ postprocess latent factors learned from MF. Since PQ is based on Euclidean distance, it is not consistent with the inner product used for estimating the preference scores. This causes that two items similarly preferred by some user may be distant from each other in the Euclidean space. Formally, the triangle inequality is violated. For example, denote $\boldsymbol{p} = [1, 0]$ user latent factor, and $\boldsymbol{q}_1 = [x, y_1], \boldsymbol{q}_2 = [x, y_2]$ latent factors of two items. The preference scores with these two items are the same, but distance between two items, $|y_1 - y_2|$, can be arbitrarily large. It is intuitively feasible to exploit the Euclidean distance [1], [9] rather than the inner product to estimate the preference scores. However, they deviate from the familiar MF framework and can not benefit the existing MF based recommender systems.

5.1 Loss Function

In this part, we will directly learn codebooks and assignments from rating data instead of latent factors. To be specific, we propose product Quantized Collaborative Filtering (pQCF) to integrate the learning of latent factors into quantization. The partition of either users or items is not based on the Euclidean distance but the inner product preference. Therefore, product quantization is extended from the Euclidean space to the inner product space. According to Eq (4), quantizers are defined as minimizing distortion error between $\boldsymbol{P}$ and $\boldsymbol{H}^T\tilde{\boldsymbol{U}}\boldsymbol{B}$, between $\boldsymbol{Q}$ and $\boldsymbol{H}^T\tilde{\boldsymbol{V}}\boldsymbol{D}$. From another perspective, $\boldsymbol{H}^T\tilde{\boldsymbol{V}}\boldsymbol{d}_j$ is a quantized representation of $\boldsymbol{q}_j$, where $\boldsymbol{d}_j$ is the j-th column of $\boldsymbol{D}$. Then we can use quantized latent vectors to estimate the preference score:

$$\begin{aligned}\hat{r}_{ij} &= \Big(\boldsymbol{H}^T\tilde{\boldsymbol{U}}\boldsymbol{b}_i\Big)^T(\boldsymbol{H}^T\tilde{\boldsymbol{V}}\boldsymbol{d}_j) = \boldsymbol{b}_i^T\tilde{\boldsymbol{U}}^T\tilde{\boldsymbol{V}}\boldsymbol{d}_j \\ &= \sum_f (\boldsymbol{U}^f\boldsymbol{b}_i^f)^T(\boldsymbol{V}^f\boldsymbol{d}_j^f)\end{aligned} \tag{7}$$

The objective function of pQCF is then formulated by:

$$\begin{aligned}\min_{\mathcal{U},\mathcal{V},\boldsymbol{B},\boldsymbol{D}} \sum_{i,j} w_{ij}\Big(r_{ij} - \sum_f (\boldsymbol{U}^f\boldsymbol{b}_i^f)^T(\boldsymbol{V}^f\boldsymbol{d}_j^f)\Big)^2 \\ + \lambda\Big(\sum_f \|\boldsymbol{U}^f\|_F^2 + \sum_f \|\boldsymbol{V}^f\|_F^2\Big)\end{aligned} \tag{8}$$

where the terms in the second line act a regularizer to prevent overfitting. $\mathcal{U}$ and $\mathcal{V}$ are the Cardesian product codebooks. In this objective function, it is worth noting that rotating the joint latent space is implicitly achieved, and explicit rotation does not take effect. Moreover, only inner product is used so that there is no violation of triangle inequality. Next we elaborate how to learn parameters.

5.2 Optimization

Codebooks and codeword assignments among different subspaces are not independent, so that the learning of codebooks and codeword assignments can not be completed separately for each subspace. This is also different from PQ and OPQ. However, it is possible to iteratively learn the codebooks and codeword assignments for each subspace. Below we focus on the learning of parameters in the f-th subspace. Denote $\tilde{\boldsymbol{V}}^{-f}$ block diagonal with $\boldsymbol{V}^f$ excluded, $\tilde{\boldsymbol{U}}^{-f}$ block diagonal with $\boldsymbol{U}^f$ excluded, and let $\boldsymbol{B}^{-f}$ and $\boldsymbol{D}^{-f}$ comprise the codeword assignments of all subspaces but the f-th. For simplifying notations, we introduce two new variables: $\boldsymbol{s}_i = \tilde{\boldsymbol{U}}^{-f}\boldsymbol{b}_i^{-f}$ and $\boldsymbol{t}_j = \tilde{\boldsymbol{V}}^{-f}\boldsymbol{d}_j^{-f}$. The preference score $\hat{r}_{ij}$ is simplified as $\hat{r}_{ij} = \boldsymbol{s}_i^T\boldsymbol{t}_j + (\boldsymbol{U}^f\boldsymbol{b}_i^f)^T(\boldsymbol{V}^f\boldsymbol{d}_j^f)$. Without confusion, the superscript f is dropped for further simplification. The objective function is then rewritten as

$$\min_{\boldsymbol{U},\boldsymbol{V},\boldsymbol{B},\boldsymbol{D}} \sum_{i,j} w_{ij}\Big(r_{ij} - \boldsymbol{s}_i^T\boldsymbol{t}_j - \boldsymbol{b}_i^T\boldsymbol{U}^T\boldsymbol{V}\boldsymbol{d}_j\Big)^2 + \lambda\Big(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2\Big) \tag{9}$$

Here $\boldsymbol{B} \in \{0,1\}^{C\times M}$ and $\boldsymbol{D} \in \{0,1\}^{C\times N}$ are slightly abused for referring codeword assignments in the f-th subspace. The columns of $\boldsymbol{U},\boldsymbol{V} \in \mathbb{R}^{D\times C}$ corresponds to codewords in the f-th subspace.

Due to the symmetric between $\boldsymbol{U}$ and $\boldsymbol{V}$, between $\boldsymbol{B}$ and $\boldsymbol{D}$, we only derive the update rule for $\boldsymbol{U}$ and $\boldsymbol{B}$. Let's consider how to update $\boldsymbol{b}_i$ for a user i first. Denoting $\tilde{\boldsymbol{q}}_j = \boldsymbol{V}\boldsymbol{d}_j$, expand the loss function and discard the terms irrelevant to $\boldsymbol{b}_i$

$$\begin{aligned}\mathcal{L}(\boldsymbol{b}_i) = \boldsymbol{b}_i^T\boldsymbol{U}^T\Big(\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\tilde{\boldsymbol{q}}_j^T\Big)\boldsymbol{U}\boldsymbol{b}_i - 2\boldsymbol{b}_i^T\boldsymbol{U}^T\sum_j w_{ij}\tilde{\boldsymbol{q}}_j r_{ij} \\ + 2\boldsymbol{b}_i^T\boldsymbol{U}^T\Big(\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\boldsymbol{t}_j^T\Big)\boldsymbol{s}_i.\end{aligned} \tag{10}$$

Since $\boldsymbol{b}_i$ is an one-hot vector, we can efficiently enumerate all choices of $\boldsymbol{b}_i$ and choose the one with the minimal loss. If the user i is assigned to the c-th codeword, i.e., $b_{ic} = 1$, the loss $\mathcal{L}_{ic}$ is defined as

$$\begin{aligned}\mathcal{L}_{ic} = \boldsymbol{u}_c^T\Big(\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\tilde{\boldsymbol{q}}_j^T\Big)\boldsymbol{u}_c - 2\boldsymbol{u}_c^T\sum_j w_{ij}\tilde{\boldsymbol{q}}_j r_{ij} \\ + 2\boldsymbol{u}_c^T\Big(\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\boldsymbol{t}_j^T\Big)\boldsymbol{s}_i.\end{aligned} \tag{11}$$

For concise, it can be rewritten in a matrix form,

$$\mathcal{L}_{ic} = \boldsymbol{u}_c^T\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T\boldsymbol{u}_c - 2\boldsymbol{u}_c^T\Big(\tilde{\boldsymbol{Q}}(\boldsymbol{w}_i\circ\boldsymbol{r}_i) - \tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T\boldsymbol{s}_i\Big) \tag{12}$$

where $\boldsymbol{W}^i$ is a diagonal matrix with $\boldsymbol{w}_i$ on the diagonal and $\boldsymbol{T} \in \mathbb{R}^{(K-D)\times N}$ is a matrix stacking $\boldsymbol{t}_j$ by column. $\circ$ operates element-wise product between vectors. Based on the setting of w_{ij}, $\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T$ and $\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T$ can be efficiently computed with a simple trick [24]. For example, $\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T = \alpha\tilde{\boldsymbol{Q}}_i\boldsymbol{T}_i^T + \tilde{\boldsymbol{Q}}\boldsymbol{T}^T$, where $\tilde{\boldsymbol{Q}}_i$ is a submatrix of $\tilde{\boldsymbol{Q}}$ selected by rated items of the user i. Be independent of the user i, $\tilde{\boldsymbol{Q}}\boldsymbol{T}^T$ can be precomputed. Similarly, $\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T = \alpha\tilde{\boldsymbol{Q}}_i\tilde{\boldsymbol{Q}}_i^T + \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{Q}}^T$. Without considering precomputing overhead, the time complexity of choosing the assignment with the minimal loss for the user i is $\mathcal{O}(\Omega_i DK + CD^2)$, where Ω_i is the number of her ratings. With overhead, updating assignment for all users costs $\mathcal{O}(\Omega DK + MCD^2)$, where $\Omega = \sum_j \Omega_j$ equals the number of ratings.

Regarding codebook generation, we also expand the loss function and discard the terms irrelevant to $\boldsymbol{U}$. The objective function with respect to $\boldsymbol{U}$ is written as follows:

$$\mathcal{L} = \sum_i \boldsymbol{b}_i^T\boldsymbol{U}^T\sum_j (w_{ij}\tilde{\boldsymbol{q}}_j\tilde{\boldsymbol{q}}_j^T)\boldsymbol{U}\boldsymbol{b}_i - 2\sum_{i,j} w_{ij}(r_{ij} - \boldsymbol{s}_i^T\boldsymbol{t}_j)\boldsymbol{b}_i^T\boldsymbol{U}^T\tilde{\boldsymbol{q}}_j$$

We can observe that it is difficult to derive the closed form for updating the whole codebook once a time. It is intuitive to derive the gradient of $\mathcal{L}$ with respect to $\boldsymbol{U}$ and to apply gradient descent for parameter learning. However, we are more interested in deriving the closed-form solutions for updating codebooks since there is no need of parameter tuning. Let's consider the loss function with respect to a codeword $\boldsymbol{u}_c$, i.e., the c-th column of $\boldsymbol{U}$, which is defined as

$$\begin{aligned}\mathcal{L}(\boldsymbol{u}_c) = \boldsymbol{u}_c^T\Big(\sum_{i\in E_c}\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\tilde{\boldsymbol{q}}_j^T + \lambda\boldsymbol{I}\Big)\boldsymbol{u}_c \\ - 2\boldsymbol{u}_c^T\sum_{i\in E_c}\sum_j w_{ij}r_{ij}\tilde{\boldsymbol{q}}_j + 2\boldsymbol{u}_c^T\sum_{i\in E_c}\sum_j w_{ij}\tilde{\boldsymbol{q}}_j\boldsymbol{t}_j^T\boldsymbol{s}_i\end{aligned} \tag{13}$$

Algorithm 1: pQCF

```
   Input: Rating matrix $\boldsymbol{R}$, code length $K$, confidence
          of observations $\alpha$.
   Output: $\mathcal{U}, \mathcal{V}, \boldsymbol{B}, \boldsymbol{D}$
1  Initialize $\mathcal{U}, \mathcal{V}, \boldsymbol{B}, \boldsymbol{D}$ ;
2  repeat
3      for $f \in \{1, \cdots, F\}$ do
4          $\boldsymbol{S} \leftarrow \tilde{\boldsymbol{U}}^{-f}\boldsymbol{B}^{-f}$ ;          // $\mathcal{O}((K-D)M)$
5          $\boldsymbol{T} \leftarrow \tilde{\boldsymbol{V}}^{-f}\boldsymbol{D}^{-f}$ ;          // $\mathcal{O}((K-D)N)$
6          $\tilde{\boldsymbol{Q}} \leftarrow \boldsymbol{V}^f\boldsymbol{D}^f$ ;                 // $\mathcal{O}(DN)$
7          Cache $\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{Q}}^T$ and $\tilde{\boldsymbol{Q}}\boldsymbol{T}^T$ ;          // $\mathcal{O}(DKN)$
8          for $i \in \{1, \cdots, M\}$ do
9              Compute $\mathcal{L}_{ic}, \forall c \in \mathcal{C}$ ;
                 // $\mathcal{O}(\Omega_i DK + CD^2)$
10             $c^\star \leftarrow \arg\min_c \mathcal{L}_{ic}$ ;
11             Update $\boldsymbol{b}_i^f$ such that $b_{ic^\star}^f = 1$;
12         for $c \in \mathcal{C}$ do // $\mathcal{C} = \{1, \cdots, C\}$
13             Update $\boldsymbol{u}_c^f$ with Eq (15) ;
                 // $\mathcal{O}(\Omega_c^I DK + D^3)$
14         $\tilde{\boldsymbol{P}} \leftarrow \boldsymbol{U}^f\boldsymbol{B}^f$ ;                 // $\mathcal{O}(DM)$
15         Cache $\tilde{\boldsymbol{P}}\tilde{\boldsymbol{P}}^T$ and $\tilde{\boldsymbol{P}}\boldsymbol{S}^T$ ;          // $\mathcal{O}(DKM)$
16         for $j \in \{1, \cdots, N\}$ do
17             Compute $\mathcal{L}_{jc}, \forall c \in \mathcal{C}$;
                 // $\mathcal{O}(\Omega_j DK + CD^2)$
18             $c^\star \leftarrow \arg\min_{c \in \mathcal{C}} \mathcal{L}_{jc}$ ;
19             Update $\boldsymbol{d}_j^f$ such that $d_{jc^\star}^f = 1$;
20         for $c \in \mathcal{C}$ do // $\mathcal{C} = \{1, \cdots, C\}$
21             Update $\boldsymbol{v}_c^f$ based on Eq (15);
                 // $\mathcal{O}(\Omega_c^J DK + D^3)$
22 until Convergent;
```

where E_c is the user set assigned to the c-th codeword. For concise, it is rewritten in a matrix form

$$\begin{aligned}\mathcal{L}(\boldsymbol{u}_c) = \boldsymbol{u}_c^T\Big(\sum_{i\in E_c}\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T + \lambda\boldsymbol{I}\Big)\boldsymbol{u}_c \\ - 2\boldsymbol{u}_c^T\sum_{i\in E_c}\Big(\tilde{\boldsymbol{Q}}(\boldsymbol{w}_i\circ\boldsymbol{r}_i) - \tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T\boldsymbol{s}_i\Big). \quad (14)\end{aligned}$$

Computing the gradient of $\mathcal{L}(\boldsymbol{u}_c)$ with respect to $\boldsymbol{u}_c$ and setting it to zero, we can get the optimal solution of $\boldsymbol{u}_c$ by solving the following system of linear equations,

$$\Big(\sum_{i\in E_c}\tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T + \lambda\boldsymbol{I}\Big)\boldsymbol{u}_c = \sum_{i\in E_c}\tilde{\boldsymbol{Q}}(\boldsymbol{w}_i\circ\boldsymbol{r}_i) - \tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T\boldsymbol{s}_i. \quad (15)$$

Based on similar analysis, making use of precomputing user-independent terms, the time complexity of updating the codeword $\boldsymbol{u}_c$ is $\mathcal{O}(\Omega_c^I DK + D^3)$, where $\Omega_c^I = \sum_{i\in E_c}\Omega_i$. Since $\sum_c \Omega_c^I = \sum_i \Omega_i$, the time complexity of updating all codewords is $\mathcal{O}(\Omega DK + CD^3)$.

The overall algorithm is shown in Algorithm 1, where the superscript f is put back for clear demonstration. In practice, $\boldsymbol{B}$ and $\boldsymbol{D}$ are not stored as binary matrices but index matrices, i.e., $\boldsymbol{B} \in \mathcal{C}^{F\times M}$ and $\boldsymbol{D} \in \mathcal{C}^{F\times N}$, where $\mathcal{C} = \{1, \cdots, C\}$. Then line 4-7 and line 14 can be efficiently computed based on array access via assignments. Algorithm 1 is based on block coordinate descent, where subspaces correspond to blocks of coordinates, so that the convergence can be theoretically guaranteed [55]. Based on previous analysis, the overall time complexity of updating codebooks and codeword assignments in each iteration is $\mathcal{O}\Big(\Omega K^2 + (M+N)CKD\Big)$. Furthermore, since the updating rules are independent between users, between items, and between codewords, parallel update can be applied to speed up the training procedure. Regarding memory cost for item recommendation, only $\boldsymbol{B}$ and $\boldsymbol{D}$ as well as F lookup tables of size $C \times C$ are needed. $4FC^2$ bytes are used for storing lookup tables. Each one-hot vector in $\boldsymbol{B}$ and $\boldsymbol{D}$ is converted into an integer of $\log C$ bits, and then each user or each item has F integers in total. Then $\frac{1}{8}(N+M)F\log C$ will be used to compactly encode $\boldsymbol{B}$ and $\boldsymbol{D}$. Since $F = K/D$, $\frac{1}{8}(M+N)K\frac{\log C}{D} + 4FC^2$ bytes are needed.

5.3 Relations with Product Quantization

Although pQCF is only based on the inner product, the subspace quantizer is still related to the k-mean quantizer. For better discussion, we further denote $\boldsymbol{A}_i = \tilde{\boldsymbol{Q}}\boldsymbol{W}^i\tilde{\boldsymbol{Q}}^T$ and $\boldsymbol{y}_i = \tilde{\boldsymbol{Q}}(\boldsymbol{w}_i\circ\boldsymbol{r}_i) - \tilde{\boldsymbol{Q}}\boldsymbol{W}^i\boldsymbol{T}^T\boldsymbol{s}_i$. Then $\mathcal{L}_{ic}$ can be reformulated as follows

$$\begin{aligned}\mathcal{L}_{ic} &= \boldsymbol{u}_c^T\boldsymbol{A}_i\boldsymbol{u}_c - 2\boldsymbol{u}_c^T\boldsymbol{y}_i \\ &= (\boldsymbol{u}_c - \boldsymbol{A}_i^{-1}\boldsymbol{y}_i)^T\boldsymbol{A}_i(\boldsymbol{u}_c - \boldsymbol{A}_i^{-1}\boldsymbol{y}_i) - \boldsymbol{y}_i^T\boldsymbol{A}_i^{-1}\boldsymbol{y}_i.\end{aligned} \quad (16)$$

Since the last term is independent of $\boldsymbol{u}_c$ and can be discarded when choosing the assignment with the minimal loss. If we substitute $\boldsymbol{U}\boldsymbol{b}_i$ with $\tilde{\boldsymbol{p}}_i$ in Eq (9), and optimize Eq (9) by alternating least square, then $\tilde{\boldsymbol{p}}_i = \boldsymbol{A}_i^{-1}\boldsymbol{y}_i$. Therefore, the assignment algorithm first learns user latent factor and then assigns it to the nearest codeword in terms of Mahalanobis distance. Therefore, pQCF unifies quantization and the learning of latent factors in a seamless way.

Regarding codewords, pQCF defines a novel "average" of group members,

$$\boldsymbol{u}_c = \Big(\sum_{i\in E_c}\boldsymbol{A}_i + \lambda\boldsymbol{I}\Big)^{-1}\Big(\sum_{i\in E_c}\boldsymbol{y}_i\Big). \quad (17)$$

This is different from $\boldsymbol{u}_c = \frac{1}{|E_c|}\sum_{i\in E_c}\boldsymbol{A}_i^{-1}\boldsymbol{y}_i$ in the k-means quantizer except the case $\boldsymbol{A}_i = \boldsymbol{A}_j, \forall i,j \in E_c$. Note that the exception is impossible due to diversity of behavior among users.

6 Quantized Collaborative Filtering

If user latent factors are not quantized, inner product can be also fast computed by table lookup, as shown in Fig. 2. Such an asymmetric pQCF is dubbed as QCF. If we directly follow the asymmetric PQ [19], the rating is estimated as

$$\hat{r}_{ij} = \langle[\boldsymbol{p}_i^1, \cdots, \boldsymbol{p}_i^F], [\boldsymbol{V}^1\boldsymbol{d}_j^1, \cdots, \boldsymbol{V}^F\boldsymbol{d}_j^F]\rangle.$$

To improve generalization ability of this model, we let user latent factors share across different subspaces. This also helps to dramatically reduce the parameters in this model. The estimated rating is then formulated by

$$\hat{r}_{ij} = \langle\boldsymbol{p}_i, \sum_f \boldsymbol{V}^f\boldsymbol{d}_j^f\rangle. \quad (18)$$

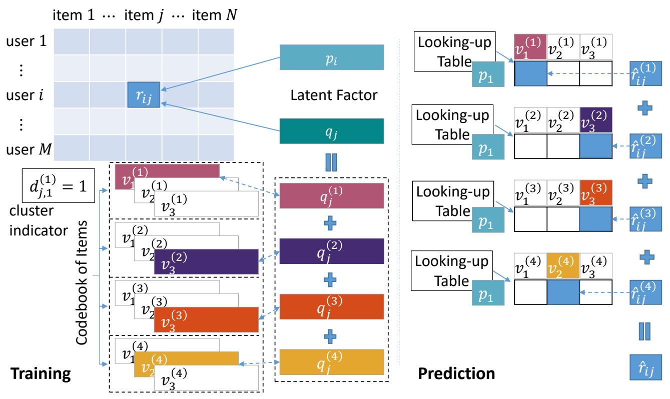
item 1 ⋯ item j ⋯ item N
user 1
user i
user M
r_{ij}
p_i
Latent Factor
q_j
$d_{j,1}^{(1)} = 1$
cluster indicator
Codebook of Items
$v_1^{(1)}$ $v_2^{(1)}$ $v_3^{(1)}$
$v_1^{(2)}$ $v_2^{(2)}$ $v_3^{(2)}$
$v_1^{(3)}$ $v_2^{(3)}$ $v_3^{(3)}$
$v_1^{(4)}$ $v_2^{(4)}$ $v_3^{(4)}$
$q_j^{(1)}$ $q_j^{(2)}$ $q_j^{(3)}$ $q_j^{(4)}$
Training
Looking-up Table
p_1
$\hat{r}_{ij}^{(1)}$ $\hat{r}_{ij}^{(2)}$ $\hat{r}_{ij}^{(3)}$ $\hat{r}_{ij}^{(4)}$
$\hat{r}_{ij}$
Prediction

Fig. 2. Framework of Quantized Collaborative Filtering.

Interestingly, QCF now turns to additive quantization [44], [45], but it is based on the inner product instead of the Euclidean distance. Moreover, different from these existing works but similar to pQCF, codebooks are directly learned from rating data via optimizing the following objective function,

$$\min_{\boldsymbol{P},\mathcal{V},\boldsymbol{D}} \sum_{i,j} w_{ij}\Big(r_{ij} - \boldsymbol{p}_i^T \sum_f \boldsymbol{V}^f \boldsymbol{d}_j^f\Big)^2 + \lambda\Big(\sum_i \|\boldsymbol{p}_i\|^2 + \sum_f \|\boldsymbol{V}^f\|_F^2\Big). \quad (19)$$

Given $\boldsymbol{P}$ fixed, the optimization with respect to $\mathcal{V}$ and $\boldsymbol{D}$ is similarly achieved by iterating each codebook and its codeword assignments. Regarding the f-th codebook, its closed-form update equation is very similar to pQCF as long as setting $\boldsymbol{t}_j = \sum_{f'\neq f} \boldsymbol{V}^{f'} \boldsymbol{d}_j^{f'}$ and $\boldsymbol{s}_i = \boldsymbol{p}_i$. In other words, $\boldsymbol{v}_c$ is updated by solving the following system of linear equations,

$$\Big(\sum_{j\in H_c} \boldsymbol{P}\boldsymbol{W}^j\boldsymbol{P}^T + \lambda \boldsymbol{I}\Big)\boldsymbol{v}_c = \sum_{j\in H_c} \boldsymbol{P}(\boldsymbol{w}_j \circ \boldsymbol{r}_j) - \boldsymbol{P}\boldsymbol{W}^j\boldsymbol{P}^T\boldsymbol{t}_j, \quad (20)$$

where H_c denotes the item set assigned to the c-th codeword of the current codebook. The loss of assigning item j to the c-th codeword is represented by

$$\mathcal{L}_{jc} = \boldsymbol{v}_c^T\boldsymbol{P}\boldsymbol{W}^j\boldsymbol{P}^T\boldsymbol{v}_c - 2\boldsymbol{v}_c^T\Big(\boldsymbol{P}(\boldsymbol{w}_j \circ \boldsymbol{r}_j) - \boldsymbol{P}\boldsymbol{W}^j\boldsymbol{P}^T\boldsymbol{t}_j\Big). \quad (21)$$

The codebook with the minimal loss is then assigned to the item j. Following similar analysis, the time complexity of updating F codebooks and its codeword assignment is $\mathcal{O}(\Omega F K^2 + NFCK^2)$, after precomputing $\boldsymbol{P}\boldsymbol{P}^T$.

Given $\mathcal{V}$ and $\boldsymbol{D}$ fixed, $\boldsymbol{P}$ is easily updated by cycling through each user's latent factor. In particular, denoting $\boldsymbol{q}_j = \sum_f \boldsymbol{V}^f \boldsymbol{d}_j^f$, the optimal $\boldsymbol{p}_i$ is obtained by solving the following system of linear equations,

$$(\boldsymbol{Q}\boldsymbol{W}^i\boldsymbol{Q}^T + \lambda\boldsymbol{I})\boldsymbol{p}_i = \boldsymbol{Q}(\boldsymbol{w}_i \circ \boldsymbol{r}_i), \quad (22)$$

where $\boldsymbol{Q}$ is obtained by stacking $\boldsymbol{q}_j$ by column. The time complexity of updating $\boldsymbol{P}$ is $\mathcal{O}(\Omega K^2 + MK^3)$.

The overall algorithm is shown in Algorithm 2, where only $\boldsymbol{P}\boldsymbol{P}^T$ and $\boldsymbol{Q}\boldsymbol{Q}^T$ are cached. The time complexity is $\mathcal{O}(\Omega F K^2 + NFCK^2)$, being dominated by updating codebooks and codeword assignments. This is around F times higher than pQCF. Regarding memory cost for item recommendation, only $\boldsymbol{P}$, $\mathcal{V}$ and $\boldsymbol{D}$ are required to store so that $\frac{1}{8}NF\log C + 4FCK + 4MK$ bytes will be allocated. Note that it may be not worth storing user preference scores of codewords in each codebook, particularly when $FC \gg K$, so it will consume slightly more time than pQCF for item recommendation.

Algorithm 2: QCF

```
Input: Rating matrix $\boldsymbol{R}$, code length $K$, confidence
       of observations $\alpha$.
Output: $\boldsymbol{P}, \mathcal{V}, \boldsymbol{D}$
1  Initialize $\boldsymbol{P}, \mathcal{V}, \boldsymbol{D}$ ;
2  $\boldsymbol{Q} \leftarrow$ zeros_like($\boldsymbol{P}$);
3  for $f \in \{1, \cdots, F\}$ do
4      $\boldsymbol{Q} \leftarrow \boldsymbol{Q} + \boldsymbol{V}^f\boldsymbol{D}^f$;                // $\mathcal{O}(NK)$
5  repeat
6      Cache $\boldsymbol{P}\boldsymbol{P}^T$ ;                         // $\mathcal{O}(MK^2)$
7      for $f \in \{1, \cdots, F\}$ do
8          $\boldsymbol{T} \leftarrow \boldsymbol{Q} - \boldsymbol{V}^f\boldsymbol{D}^f$ ;                // $\mathcal{O}(NK)$
9          for $c \in \mathcal{C}$ do // $\mathcal{C} = \{1, \cdots, C\}$
10             Update $\boldsymbol{v}_c^f$ with Eq (20);
                 // $\mathcal{O}(\Omega_c^J K^2 + K^3)$
11         for $j \in \{1, \cdots, N\}$ do
12             Compute $\mathcal{L}_{jc}, \forall c \in \mathcal{C}$;
                 // $\mathcal{O}(\Omega_j K^2 + CK^2)$
13             $c^\star \leftarrow \arg\min_{c\in\mathcal{C}} \mathcal{L}_{jc}$ ;
14             Update $\boldsymbol{d}_j^f$ such that $d_{jc^\star}^f = 1$;
15         $\boldsymbol{Q} \leftarrow \boldsymbol{T} + \boldsymbol{V}^f\boldsymbol{D}^f$;
16     Cache $\boldsymbol{Q}\boldsymbol{Q}^T$ ;                         // $\mathcal{O}(NK^2)$
17     for $i \in \{1, \cdots, M\}$ do
18         Update $\boldsymbol{p}_i$ with Eq(20);     // $\mathcal{O}(\Omega_i K^2 + K^3)$
19 until Convergent;
```

7 Experiments

We evaluate the proposed algorithms (pQCF) from the aspects of recommendation accuracy, sensitivity of parameters, efficiency of item recommendation and visualization of item latent factors.

7.1 Datasets

Since pQCF is suitable for both explicit feedback and implicit feedback, we use 4 explicit datasets and 2 implicit datasets for evaluation. The 4 explicit datasets are also converted into implicit datasets when evaluating pQCF against implicit feedback according to suggestions of prior works [4], [22]. Table 1 summarizes statistics of these datasets. The datasets vary in the numbers of items and ratings, the density and concentration. The Yelp dataset includes users' ratings for points of interest. The Amazon dataset is a subset of customers' ratings for Amazon books [56]. The Netflix dataset is from the well-known Netflix Prize. The rating scores of these three datasets are integers from 1 to 5. The MovieLens dataset is from the classic MovieLens10M dataset. The rating scores are from

0.5 to 5 with 0.5 granularity. Following convention of evaluating CF algorithms, we filter these four datasets such that users rated at least 20 items that were rated by at least 20 users. The implicit datasets include Gowalla and LastFM. The Gowalla dataset includes users' check-ins at locations. Because of low density, it is less strictly filtered such that users check-in at least 10 locations, which were checked in at least by 10 users. The LastFM dataset is based on users' play count of songs. Following [1], we include songs a user listened to at least 5 times as positive feedback.

TABLE 1
Data Statistics. Concentration (Concn.) indicates rating percentage on the top 5% most popular items [1].

Datasets	Yelp	Amazon	Netflix	MovieLens	Gowalla	LastFM
#users	18,454	35,736	429,584	69,838	29,858	357,847
#items	14,670	38,121	17,764	8,939	40,988	156,122
#ratings	869,126	1,960,674	99,884,887	9,983,739	1,027,464	16,893,651
Density	3.21e-03	1.44e-03	1.31e-02	1.60e-02	8.40e-04	3.02e-04
Concn.	23.25%	22.56%	59.43%	47.54%	29.15%	81.47%

7.2 Settings

For each user, we randomly sample his 80% ratings as training set and the rest 20% as testing test. We fit a model to the training set and evaluate it in the test set. In case of explicit feedback, the model is fit to the explicit training set, but evaluated against a converted implicit test set from the explicit test set, which enables better computation of ranking-based metrics. We repeat 5 random splits and report averaged accuracy metrics. The hyperparameters of pQCF and baselines are tuned on a validation set, which consists of 5% ratings of the training set. QCF shares the hyperparameters with pQCF. Note that pQCF and QCF can handle both explicit and implicit feedback without any modification except that the hyperparameters may be different.

7.2.1 Metrics

The accuracy of recommendation is based on how well positively preferred items in the test set are ranked[2]. We use three widely-used metrics of ranking evaluation: Area under ROC curve (AUC), Recall and Normalized Discounted Cumulative Gain (NDCG). The cutoff k in Recall and NDCG is set 50 by default.

7.2.2 Baselines

Regarding baselines, we mainly focus on hashing-based collaborative filtering and quantization-based methods. The baselines of hashing-based collaborative filtering include:

- **DMF** [18], the state-of-the-art discrete hashing for item recommendation, which can take both explicit feedback and implicit feedback as inputs. The parameter ρ for interaction regularization is tuned within $\{10^{-6}, 10^{-5}, \cdots, 10^{-1}, 1\}$, both α and β for the decorrelated and balanced constraints are tuned within $\{10^{-4}, 10^{-3}, \cdots, 10^{1}, 10^{2}\}$.
- **DCF** [17], the first discrete hashing for collaborative filtering, which directly tackles a discrete optimization problem, subject to the decorrelated and balanced constraints. The parameters α and β for the decorrelated and balanced constraints are tuned within $\{10^{-4}, 10^{-3}, \cdots, 10^{1}, 10^{2}\}$.
- **BCCF** [15], is a two-stage hashing-based collaborative filtering. It first solves matrix factorization with a balanced regularization. It then uses ITQ [57] to derive the binary codes. Following their suggestions, the coefficient for the balanced regularization is tuned within $\{0.01, 0.03, 0.05, 0.07, 0.09\}$.
- **PPH** [16], is a preference preserving hashing for collaborative filtering. PPH first solves matrix factorization with constant feature norm, where preferences can be well approximated by similarities. It then binarizes latent factor into K-bit phrase codes and quantizes 2-bit magnitude codes. The coefficient for the constant feature norm is tuned within $\{0.01, 0.5, 1, 2, 4, 8, 16\}$.
- **C**ollaborative **H**ashing, dubbed as **CH**, is a two-stage method for learning binary codes [49]. CH first solves matrix factorization on the full-matrix, by treating all unrated items as zero-rated. Following [17], we implement CH as $\arg\min_{\boldsymbol{U},\boldsymbol{V}} \|\boldsymbol{R} - \boldsymbol{U}\boldsymbol{V}^T\|_F^2$, s.t. $\boldsymbol{U}^T\boldsymbol{U} = m\boldsymbol{I}_k, \boldsymbol{V}^T\boldsymbol{V} = n\boldsymbol{I}_k$. CH then binarizes $\boldsymbol{U}$ and $\boldsymbol{V}$ based on the sign function.

The quantization-based methods include:

- **PQ** [19], directly quantizes latent factors of both users and items which are learned from matrix factorization according to Eq (1).
- **OPQ_CF**, is the proposed variant of OPQ [20] for collaborative filtering, introduced in Section 4.1.
- **OPQ_CF+**, is an asymmetric OPQ_CF, which only quantizes item latent factors, but rotates user latent factors with the optimal rotation matrix returned by OPQ.

The source code of the proposed algorithm and baselines are released in the Github repository[3]. The code length of the baselines together with the proposed pQCF is set to 64 by default. Each subspace of these quantization-based methods is of 8 dimensions, and clustered into 256 clusters following Jegou's suggestion [19]. The confidence parameter α of ratings is only tuned for matrix factorization in the validation set within $\{5, 10, 25, 50, 100, 250, 500\}$. Although the prior work [21] also studied quantization-based MIPS, it is unfair to compare it with ours since their settings are different, as discussed in related work. Hence, it is not considered as one of quantization-based baselines.

7.3 Comparison with Baselines

Table 2 shows recommendation accuracy in terms of Recall@50, NDCG@50 and AUC for the 4 explicit datasets, where SL-Prior [52] bounds QCF and pQCF from the above. We have the following observations.

First, pQCF significantly outperforms the best hashing-based collaborative filtering by up to 78.7% in NDCG, 86.2% in Recall and 11.8% in AUC in the four explicit datasets. This implies large quantization errors of binarized latent factor models. Though directly learning hash codes from rating data could reduce quantization errors of binarization, this task is very challenging due to involving combinational optimization. Quantization-like algorithms can lead to significant reduction of quantization errors and substantial improvements of recommendation accuracy.

2. In explicit datasets, items with ratings greater than or equal to 4 are considered positively preferred

3. https://github.com/DefuLian/recsys

Second, pQCF shows up to 12.3%, 9.8% and 2% improvements of NDCG, Recall and AUC over PQ. This indicates direct application of product quantization is suboptimal, since PQ is based on the Euclidean distance while the preference score is estimated by the inner product. It is also observed that pQCF is superior to OPQ_CF and that pQCF can bring larger improvements in the denser datasets. Therefore, unifying quantization and the learning of latent factors benefits further reduction of quantization errors.

Third, OPQ_CF is consistently better than PQ. The improvements are up to 6.4% in NDCG, 5.5% in Recall and 1.45% in AUC. This shows that rotating joint latent space indeed helps reduce quantization errors, as concluded in [20].

Forth, QCF outperforms pQCF by up to 13.4% in NDCG, 10.9% in Recall and 0.92% in AUC, and OPQ_CF+ outperforms OPQ_CF by up to 13.9% in NDCG, 11.9% in Recall and 1.28% in AUC. This indicates only quantizing item latent factors further reduces the quantization errors and improves recommendation performance.

Finally, when re-ranking not applied, performance degradation due to quantizing item latent factors is 2.6% in NDCG, 2.2% in Recall and 0.23% in AUC on average. Such performance degradation is indeed negligible.

Table 3 shows recommendation accuracy for the 6 implicit datasets, 4 of which are converted from explicit datasets. Note that these results can not be compared against that of explicit feedback, since the data set is not of the same size. According to this table, we have similar observations, even in the 2 real implicit feedback datasets. Note that implicit feedback datasets are much sparser than explicit feedback, so recommendation based on implicit feedback is more challenging. Moreover, the LastFM dataset shows high concentration, indicating more than 81% plays concentrate on the top-5% most popular songs. The recommendation in the LastFM dataset is much more difficult. However, pQCF still works well, showing 60.9% improvements relative to the best hashing-based collaborative filtering and 15.6% improvements relative to PQ in terms of NDCG. Performance degradation of recommendation is 3.1% in NDCG and 2.5% in Recall on average. This demonstrates the power of the proposed algorithms.

7.4 Convergence

Given any initialization of codeword assignments, both pQCF and QCF can be convergent in theory according to [55]. However, similar to k-means, they are sensitive to initialization. The empirical results of convergence in the MovieLens dataset are shown in Fig. 3. It is clear that even with a random initialization, they can be convergent, but to a higher loss and lower accuracy of recommendation than a good initialization. Moreover, initialized by OPQ_CF, both algorithms further reduce loss and improve recommendation accuracy. This implies effectiveness of the proposed optimization algorithms.

7.5 Sensitivity Analysis

The confidence parameter α mainly depends on the density of datasets, and has been well studied in prior works [24], [52]. Here, we focus on the sensitivity to code length. The results of study in the Amazon dataset and the Yelp dataset

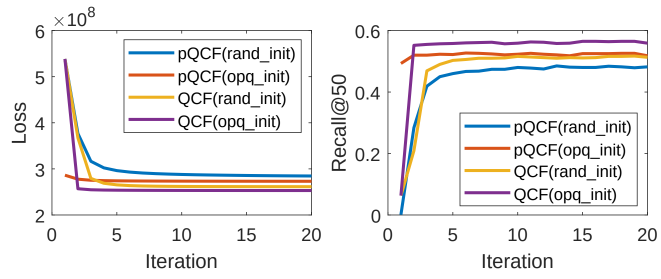

Fig. 3. Illustration of convergence. opq_init indicates the algorithms are initialized via OPQ_CF, and rand_init means random initialization.

are shown in Fig. 4. The accuracy of recommendation grows with the increase of code length in the Amazon dataset while it may show over-fitting in the Yelp dataset when the code length is larger than 128. The may be because far more factors are required to determine users' purchase of books than users' choice of restaurants, hotels and other points of interest. This is evidenced by larger singular values in the amazon dataset than the Yelp dataset at the same positions.

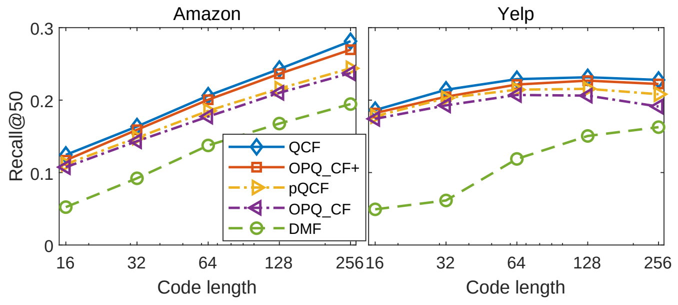

Fig. 4. Sensitivity Analysis of Code Length.

7.6 Top-k Recommendation

Although pQCF is significantly superior to the best hashing based collaborative filtering, it is unknown how much efficiency should be traded off. As discussed before, the preference score using quantized vectors can be efficiently computed by a few additions given precomputed lookup tables. The preference score using binarized vectors is also efficiently computed via hamming distance. We record their speedup and relative recall of top-k recommendation to real-valued vectors learned from MF. Following [19], [58], we re-rank the top-k items with accurate preference scores. The results in the LastMF dataset with the largest number of items are shown in Fig. 5, where k is varied from 200 to 2000. It is easy to observe that pQCF significantly outperforms DMF with comparable retrieval time. With re-ranking, pQCF achieves negligible accuracy degradation in both Recall@100 and Recall@50 with 30+x speedup.

We also report the results of ANNOY[4] in Fig. 5. ANNOY is one of the best industrial ANN libraries [39]. ANNOY performs better when using the Euclidean distance. MIPS is transformed into NNS according to [37]. The number of trees is fixed to 50 and the number of nodes to inspect is varied from 7% to 15% of items. The results show that pQCF is much better than ANNOY given the approximately same

4. https://github.com/spotify/annoy

TABLE 2
Comparison with the state-of-the-art on 4 explicit feedback datasets in terms of NDCG@50, Recall@50 and AUC (×100).

Dataset	PPH	BCCF	DCF	CH	DMF	PQ	OPQ_CF	pQCF	OPQ_CF+	QCF	SL-Prior
						NDCG@50					
Yelp	0.80±0.04	1.78±0.04	1.46±0.03	3.43±0.07	**6.03**±0.07	9.75±0.06	10.37±0.11	**10.77**±0.12	11.09±0.07	**11.65**±0.11	11.81±0.11
Amazon	0.62±0.02	3.60±0.03	1.56±0.03	5.00±0.05	**7.36**±0.05	8.94±0.06	9.42±0.05	**10.05**±0.11	10.73±0.08	**11.38**±0.06	12.45±0.06
MovieLens	2.75±0.03	5.67±0.02	7.54±0.12	7.23±0.12	**26.41**±0.11	34.42±0.11	36.11±0.09	**38.52**±0.06	39.79±0.15	**42.41**±0.05	42.57±0.08
Netflix	1.81±0.01	4.57±0.01	5.24±0.06	6.50±0.12	**19.59**±0.05	28.77±0.06	29.99±0.06	**32.28**±0.02	34.11±0.11	**36.59**±0.02	36.63±0.01
						Recall@50					
Yelp	1.84±0.07	4.17±0.11	3.44±0.08	7.71±0.07	**11.64**±0.15	19.99±0.18	21.10±0.20	**21.67**±0.27	22.26±0.20	**23.02**±0.21	23.24±0.23
Amazon	1.24±0.03	6.50±0.03	3.07±0.10	9.21±0.09	**13.62**±0.10	17.08±0.13	17.85±0.15	**18.63**±0.16	19.94±0.17	**20.61**±0.12	22.24±0.12
MovieLens	4.38±0.05	7.33±0.03	11.74±0.13	10.46±0.16	**38.51**±0.06	48.02±0.07	50.03±0.05	**52.43**±0.08	54.06±0.11	**56.39**±0.07	56.67±0.05
Netflix	2.33±0.02	4.63±0.01	6.88±0.06	7.37±0.15	**23.58**±0.07	34.89±0.05	36.01±0.04	**38.29**±0.02	40.30±0.07	**42.47**±0.01	42.50±0.02
						AUC					
Yelp	67.51±0.23	**85.41**±0.08	77.25±0.46	77.37±0.15	73.75±0.11	90.84±0.04	91.51±0.03	**91.75**±0.08	**92.47**±0.08	92.45±0.09	92.58±0.09
Amazon	69.49±0.11	81.39±0.07	82.48±0.20	83.39±0.10	**88.50**±0.04	93.27±0.02	93.98±0.04	**94.31**±0.05	95.01±0.04	**95.14**±0.02	95.46±0.03
MovieLens	74.09±0.19	74.76±0.03	82.37±0.05	68.36±0.13	**90.25**±0.05	93.97±0.05	95.33±0.03	**95.85**±0.02	96.05±0.03	**96.35**±0.02	96.55±0.02
Netflix	71.90±0.11	70.03±0.02	82.25±0.05	67.24±0.17	**85.62**±0.03	94.23±0.01	94.43±0.01	**95.20**±0.01	95.64±0.05	**96.07**±0.01	96.31±0.01

TABLE 3
Comparison with the state-of-the-art on 6 implicit feedback datasets in terms of NDCG@50, Recall@50 and AUC (×100).

Dataset	PPH	BCCF	DCF	CH	DMF	PQ	OPQ_CF	pQCF	OPQ_CF+	QCF	WRMF
						NDCG@50					
Yelp	1.34±0.11	2.02±0.02	3.72±0.05	2.42±0.03	**6.77**±0.09	8.67±0.11	9.23±0.13	**9.42**±0.08	9.83±0.13	**10.26**±0.15	10.53±0.17
Amazon	0.89±0.06	3.49±0.08	4.40±0.03	4.16±0.05	**7.96**±0.07	8.68±0.03	9.17±0.06	**9.49**±0.09	10.33±0.08	**10.74**±0.02	11.68±0.05
MovieLens	3.27±0.57	3.77±0.03	4.14±0.07	5.36±0.08	**24.08**±0.11	29.45±0.14	30.91±0.11	**32.29**±0.06	33.61±0.05	**35.82**±0.04	35.54±0.01
Netflix	3.02±1.08	2.24±0.01	0.25±0.01	5.45±0.10	**18.55**±0.15	24.51±0.07	26.11±0.03	**27.87**±0.02	29.73±0.03	**31.46**±0.02	31.50±0.01
Gowalla	1.45±0.10	6.45±0.07	6.77±0.12	4.18±0.07	**11.36**±0.14	11.96±0.09	12.67±0.08	**13.63**±0.08	13.49±0.08	**14.77**±0.06	15.58±0.07
LastFM	1.08±0.10	3.54±0.01	3.03±0.22	1.91±0.02	**14.69**±0.06	20.44±0.10	21.89±0.24	**23.63**±0.04	24.64±0.07	**26.35**±0.02	27.28±0.02
						Recall@50					
Yelp	3.15±0.21	4.62±0.04	8.64±0.09	5.45±0.08	**14.98**±0.16	17.82±0.20	18.87±0.13	**19.03**±0.12	19.87±0.19	**20.40**±0.19	20.87±0.23
Amazon	1.86±0.14	6.43±0.12	8.60±0.06	7.63±0.08	**15.13**±0.13	16.43±0.07	17.24±0.12	**17.53**±0.15	19.04±0.09	**19.41**±0.08	20.83±0.10
MovieLens	6.29±0.86	4.95±0.03	6.93±0.03	7.92±0.17	**38.83**±0.14	43.75±0.09	45.58±0.08	**47.26**±0.05	48.84±0.09	**51.05**±0.04	50.81±0.04
Netflix	4.03±1.56	2.12±0.00	0.34±0.01	6.19±0.12	**24.47**±0.24	30.46±0.05	32.17±0.02	**34.05**±0.04	36.03±0.04	**37.75**±0.02	37.69±0.02
Gowalla	3.20±0.22	12.83±0.09	14.14±0.20	8.37±0.14	**21.17**±0.22	22.24±0.21	22.91±0.09	**23.69**±0.17	24.15±0.10	**24.92**±0.13	25.92±0.14
LastFM	1.92±0.20	5.53±0.02	5.19±0.37	3.13±0.02	**22.04**±0.08	28.37±0.07	30.09±0.07	**32.01**±0.04	33.35±0.05	**34.86**±0.01	35.89±0.02
						AUC					
Yelp	84.45±0.74	88.92±0.07	92.37±0.05	74.88±0.38	**92.66**±0.05	89.16±0.06	**90.07**±0.12	89.96±0.07	**90.88**±0.10	90.46±0.06	90.76±0.08
Amazon	79.66±1.17	81.96±0.05	**92.25**±0.03	81.14±0.20	91.83±0.05	91.80±0.04	92.69±0.08	**92.78**±0.08	93.67±0.05	**93.70**±0.02	94.15±0.03
MovieLens	57.35±6.13	68.16±0.01	88.53±0.42	66.60±0.15	**94.24**±0.02	93.26±0.04	94.28±0.03	**94.97**±0.05	95.01±0.02	**95.42**±0.02	95.55±0.04
Netflix	57.58±2.50	63.55±0.02	72.30±0.42	65.76±0.43	**92.93**±0.06	91.08±0.01	92.44±0.03	**93.50**±0.01	93.70±0.01	**94.43**±0.01	94.59±0.01
Gowalla	85.39±1.42	89.75±0.06	**95.74**±0.03	84.43±0.06	95.38±0.05	95.62±0.02	95.89±0.05	**96.16**±0.03	96.76±0.02	**96.76**±0.01	96.80±0.02
LastFM	88.01±0.07	87.68±0.01	**96.60**±0.04	77.66±0.03	96.49±0.01	94.97±0.01	95.84±0.01	**96.48**±0.01	96.46±0.01	**96.90**±0.02	97.44±0.00

speedup ratio. Due to highly accurate recommendation, pQCF can be further integrated into ANNOY for reducing index size of ANNOY and further accelerating top-k item retrieval in this library.

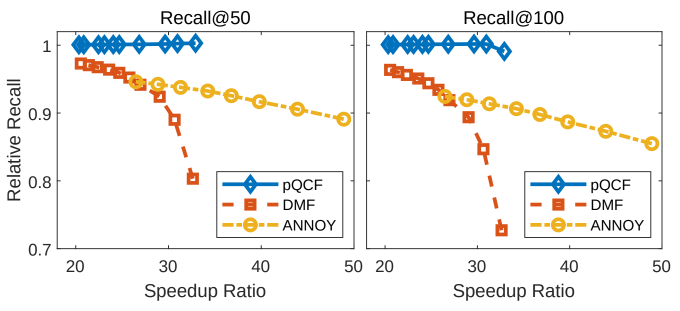

Fig. 5. Trade-off between Efficiency and Accuracy of Top-k Recommendation.

7.7 Visualization of Item Latent Factors

For better understanding why pQCF performs well, we visualize the learned item factors by MF, pQCF, QCF and DMF with the popular visualization tool t-SNE [59]. t-SNE converts latent factors into two-dimensional vectors by preserving cosine similarity between items. Then each item is plotted in a two-dimensional space and labeled as a unique color. The color of each item represents one of its genres, which is selected as the least frequent genre of the item. We finally choose 4 genres with a very small number of overlap movies, and visualize them in Fig. 6. Obviously, in DMF, the points of different genres are mixed with each other. In contrast, in pQCF and QCF, the points are better separated and clusters are formed to some extent. This validates the superiority of the quantized methods to the binarized methods. MF gives the best visualization result, indicating quantization reduces representation capacity.

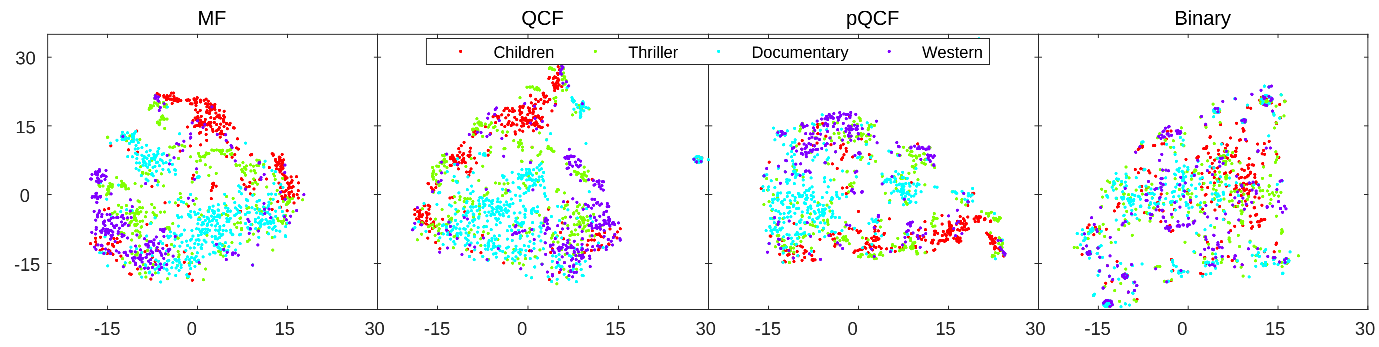

Fig. 6. t-SNE embedding of movies of four genres on the MovieLens dataset based on cosine similarity. Each point indicates embedding of a movie, and its color indicate one of movie genres.

8 Conclusions and Future Work

We proposed product quantized collaborative filtering and its variant to learn semi-structured latent factors for items (or users) from rating data. They were efficiently optimized based on block coordinate descent, whose time complexity is linearly proportional to the number of ratings. The algorithms were evaluated against 6 real-world explicit or implicit datasets. The results showed that the proposed algorithms significantly outperformed the state-of-the-art hashing-based collaborative filtering with comparable retrieval time and just a few extra memories. pQCF also showed higher recommendation accuracy than one of the best ANN libraries with comparable retrieval time, indicating that the proposed algorithms lead to better trade-off between efficiency and accuracy of top-k recommendation.

A wide range of future work can be explored. For example, since we observed that ANNOY using the inner product metric performed poor, it is very interesting to design new index structures, such as inverted index and hierarchical 2-means tree, for maximum inner product search. It is also interesting to investigate deep quantized collaborative filtering and the application of quantization for graph embedding. Finally, in most cases, item recommendation should be quickly adaptive to users' interest evolving, so it is also worth studying online quantized collaborative filtering.

Acknowledgments

The work was partially supported by grants from the National Natural Science Foundation of China (Grant No. 61976198, U1605251, 61832017 and 61631005).

References

[1] C.-K. Hsieh, L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin, "Collaborative metric learning," in *Proceedings of WWW'17*. International World Wide Web Conferences Steering Committee, 2017, pp. 193–201.

[2] M. Kula, "Metadata embeddings for user and item cold-start recommendations," *arXiv preprint arXiv:1507.08439*, 2015.

[3] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of CIKM'13*. ACM, 2013, pp. 2333–2338.

[4] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of KDD'15*. ACM, 2015, pp. 1235–1244.

[5] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of KDD'16*. ACM, 2016, pp. 353–362.

[6] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," in *Proceedings of IJCAI'17*. AAAI Press, 2017, pp. 1725–1731.

[7] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *Proceedings of KDD'18*. ACM, 2018, pp. 1754–1763.

[8] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of WWW'07*. ACM, 2007, pp. 271–280.

[9] M. Khoshneshin and W. N. Street, "Collaborative filtering via euclidean embedding," in *Proceedings of RecSys'10*. ACM, 2010, pp. 87–94.

[10] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," in *Proceedings of NIPS'14*, 2014, pp. 2321–2329.

[11] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet, "Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces," in *Proceedings of RecSys'14*. ACM, 2014, pp. 257–264.

[12] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. Tung, "Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search," in *Proceedings of KDD'18*. ACM, 2018, pp. 1561–1570.

[13] N. Koenigstein, P. Ram, and Y. Shavitt, "Efficient retrieval of recommendations in a matrix factorization framework," in *Proceedings of CIKM'12*. ACM, 2012, pp. 535–544.

[14] A. Karatzoglou, A. J. Smola, and M. Weimer, "Collaborative filtering on a budget," in *Proceedings of AISTATS'10*, 2010, pp. 389–396.

[15] K. Zhou and H. Zha, "Learning binary codes for collaborative filtering," in *Proceedings of KDD'12*. ACM, 2012, pp. 498–506.

[16] Z. Zhang, Q. Wang, L. Ruan, and L. Si, "Preference preserving hashing for efficient recommendation," in *Proceedings of SIGIR'14*. ACM, 2014, pp. 183–192.

[17] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua, "Discrete collaborative filtering," in *Proceedings of SIGIR'16*. ACM, 2016, pp. 325–334.

[18] D. Lian, R. Liu, Y. Ge, K. Zheng, X. Xie, and L. Cao, "Discrete content-aware matrix factorization," in *Proceedings of KDD'17*, 2017, pp. 325–334.

[19] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[20] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proceedings of CVPR'13*, 2013, pp. 2946–2953.

[21] R. Guo, S. Kumar, K. Choromanski, and D. Simcha, "Quantization based fast inner product search," in *Artificial Intelligence and Statistics*, 2016, pp. 482–490.

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

[23] S. Rendle and C. Freudenthaler, "Improving pairwise learning for item recommendation from implicit feedback," in *Proceedings of WSDM'14*. ACM, 2014, pp. 273–282.

[24] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for

implicit feedback datasets," in *Proceedings of ICDM'08*. IEEE, 2008, pp. 263–272.

[25] I. Bayer, X. He, B. Kanagal, and S. Rendle, "A generic coordinate descent framework for learning from implicit feedback," in *Proceedings of WWW'17*, 2017, pp. 1341–1350.

[26] W. Krichene, N. Mayoraz, S. Rendle, L. Zhang, X. Yi, L. Hong, E. Chi, and J. Anderson, "Efficient training on very large corpora via gramian estimation," *arXiv preprint arXiv:1807.07187*, 2018.

[27] J. Weston, S. Bengio, and N. Usunier, "Large scale image annotation: learning to rank with joint word-image embeddings," *Machine learning*, vol. 81, no. 1, pp. 21–35, 2010.

[28] J. Chen, D. Lian, and K. Zheng, "Improving one-class collaborative filtering via ranking-based implicit regularizer," in *Proceedings of AAAI'19*, vol. 33, 2019, pp. 37–44.

[29] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic, "Climf: collaborative less-is-more filtering," in *Proceedings of IJCAI'13*, 2013.

[30] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of WWW'17*, 2017, pp. 173–182.

[31] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, "Product-based neural networks for user response prediction," in *Proceedings of ICDM'16*. IEEE, 2016, pp. 1149–1154.

[32] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*. ACM, 2017, p. 12.

[33] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016, pp. 7–10.

[34] P. Sun, L. Wu, and M. Wang, "Attentive recurrent social recommendation," in *Proceedings of SIGIR'18*. ACM, 2018, pp. 185–194.

[35] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of KDD'18*. ACM, 2018, pp. 974–983.

[36] A. Shrivastava and P. Li, "Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips)," *arXiv preprint arXiv:1410.5410*, 2014.

[37] B. Neyshabur and N. Srebro, "On symmetric and asymmetric lshs for inner product search," in *Proceedings of ICML'15*, 2015, pp. 1926–1934.

[38] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. ACM, 2004, pp. 253–262.

[39] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement (v1. 0)," *arXiv preprint arXiv:1610.02455*, 2016.

[40] M. Wang, W. Fu, S. Hao, H. Liu, and X. Wu, "Learning on big graph: Label inference and regularization with anchor hierarchy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 5, pp. 1101–1114, 2017.

[41] H. Yin, B. Cui, X. Zhou, W. Wang, Z. Huang, and S. Sadiq, "Joint modeling of user check-in behaviors for real-time point-of-interest recommendation," *ACM Transactions on Information Systems (TOIS)*, 2016.

[42] H. Yin, B. Cui, Y. Sun, Z. Hu, and L. Chen, "Lcars: A spatial item recommender system," *ACM Transactions on Information Systems (TOIS)*, vol. 32, no. 3, p. 11, 2014.

[43] H. Yin, X. Zhou, B. Cui, H. Wang, K. Zheng, and Q. V. H. Nguyen, "Adapting to user interest drift for poi recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2566–2581, 2016.

[44] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proceedings of CVPR'14*, 2014, pp. 931–938.

[45] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proceedings of ICML'14*, 2014, pp. 838–846.

[46] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.

[47] W. Wu, B. Li, L. Chen, and C. Zhang, "Consistent weighted sampling made more practical," in *Proceedings of WWW'17*, 2017, pp. 1035–1043.

[48] W. Wu, B. Li, L. Chen, J. Gao, and C. Zhang, "A review for weighted minhash algorithms," *arXiv preprint arXiv:1811.04633*, 2018.

[49] X. Liu, J. He, C. Deng, and B. Lang, "Collaborative hashing," in *Proceedings of CVPR'14*, 2014, pp. 2139–2146.

[50] Y. Zhang, D. Lian, and G. Yang, "Discrete personalized ranking for fast collaborative filtering from implicit feedback," in *Proceedings of AAAI'17*, 2017, pp. 1669–1675.

[51] D. Lian, X. Xie, and E. Chen, "Discrete matrix factorization and extension for fast item recommendation," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2019.

[52] R. Devooght, N. Kourtellis, and A. Mantrach, "Dynamic matrix factorization with priors on unknown values," in *Proceedings of KDD'15*. ACM, 2015, pp. 189–198.

[53] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, 2014.

[54] L. Eldén and H. Park, "A procrustes problem on the stiefel manifold," *Numerische Mathematik*, vol. 82, no. 4, pp. 599–619, 1999.

[55] A. Beck and L. Tetruashvili, "On the convergence of block coordinate descent type methods," *SIAM journal on Optimization*, vol. 23, no. 4, pp. 2037–2060, 2013.

[56] J. McAuley, R. Pandey, and J. Leskovec, "Inferring networks of substitutable and complementary products," in *Proceedings of KDD'15*. ACM, 2015, pp. 785–794.

[57] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, 2013.

[58] M. Wang, H. Li, D. Tao, K. Lu, and X. Wu, "Multimodal graph-based reranking for web image search," *IEEE Transactions on Image Processing*, vol. 21, no. 11, pp. 4649–4661, 2012.

[59] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

Defu Lian is a research professor in the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei. He received the B.E. and Ph.D. degrees in computer science from University of Science and Technology of China (USTC) in 2009 and 2014, respectively. His research interest includes spatial data mining, recommender systems, and learning to hash.

Xing Xie (SM'09) is currently a principle researcher in Microsoft Research Asia, and a guest PhD advisor with USTC. His research interest include spatial data mining, location-based services, social networks, and ubiquitous computing. He was recently involved in the program or organizing committees of more than 70 conferences and works.

Enhong Chen (SM'07) received the PhD degree from USTC. He is a professor and vice dean of the School of Computer Science, USTC. His general area of research includes data mining and machine learning, social network analysis, and recommender systems. He has published more than 100 papers in refereed conferences and journals, including TKDE, KDD, and NIPS.

Hui Xiong (SM'07) is currently a Full Professor and Vice Chair of the Management Science and Information Systems Department at the Rutgers, the State University of New Jersey. He received the B.E. degree from the University of Science and Technology of China (USTC), and the Ph.D. degree from the University of Minnesota (UMN). His general area of research is data and knowledge engineering.