Preference Preserving Hashing for Efficient Recommendation

Zhiwei Zhang, Qifan Wang, Lingyun Ruan and Luo Si
Department of Computer Science, Purdue University
{zhan1187, wang868, ruan1, lsi}@purdue.edu

ABSTRACT

Recommender systems usually need to compare a large number of items before users' most preferred ones can be found. This process can be very costly if recommendations are frequently made on large scale datasets. In this paper, a novel hashing algorithm, named Preference Preserving Hashing (PPH), is proposed to speed up recommendation. Hashing has been widely utilized in large scale similarity search (e.g. similar image search), and the search speed with binary hashing code is significantly faster than that with real-valued features. However, one challenge of applying hashing to recommendation is that, recommendation concerns users' preferences over items rather than their similarities. To address this challenge, PPH contains two novel components that work with the popular matrix factorization (MF) algorithm. In MF, users' preferences over items are calculated as the inner product between the learned real-valued user/item features. The first component of PPH constrains the learning process, so that users' preferences can be well approximated by user-item similarities. The second component, which is a novel quantization algorithm, generates the binary hashing code from the learned real-valued user/item features. Finally, recommendation can be achieved efficiently via fast hashing code search. Experiments on three real world datasets show that the recommendation speed of the proposed PPH algorithm can be hundreds of times faster than original MF with real-valued features, and the recommendation accuracy is significantly better than previous work of hashing for recommendation.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

Keywords

Preference, Hashing, Recommendation, Efficiency

SIGIR'14, July 06 - 11 2014, Gold Coast, QLD, Australia

http://dx.doi.org/10.1145/2600428.2609578 .

1. INTRODUCTION

Recommendation has attracted intensive attention from both academia and industry [17, 1], and great progress has been made recently [13, 14]. However, it is still challenging to design an efficient recommender system that scales well with large datasets [33, 4].

In this paper, we focus on the popular matrix factorization (MF) algorithm [14], and propose new research for efficiency in recommendation. The recommendation process of MF can be divided into two parts: (1) real-valued features are learned for each user and item, which we call *preference modeling*; (2) all candidate items are compared and ranked so that the top ones are returned as recommendation, which we call *preference ranking*. For the majority of MF-like algorithms [13, 14, 17, 22], the cost of preference modeling for a specific user is $O(m)$, where m is the number of rated items. In comparison, the cost of preference ranking is usually $O(M)$, where M is the item number in the entire dataset. In general case, there is $M >> m$.

Although the efficiency of preference modeling has been extensively studied [5, 33], limited research exists for efficiency of preference ranking. Yet the complexity of $O(M)$ can become a critical issue in practice. For instance, for large scale datasets like Netflix [1], it can take hours to do preference ranking for all users based on the *pre-trained* preference model of MF (i.e. the real-valued user and item features). The situation will be much worse if we consider that even the 100-million-rating Netflix dataset is only a very tiny part (less than 1%) of the real world one. Furthermore, many recommender systems [36, 3, 25, 38] update users' preference models frequently, either explicitly requested by users themselves or implicitly updated by their item-rating behaviors. A good recommender system shall make recommendations that meet users' latest preference models. In this case, preference ranking will be conducted frequently. Although sometimes engineering strategies may help, e.g. parallel computing, the overall computational burden remains unreduced. Therefore new research is needed to reduce the cost of preference ranking.

Based on the above consideration, in this paper we study the efficiency issue of preference ranking, and propose a novel hashing algorithm as solution. Hashing [6, 24, 31] has now become a very popular technique for large scale similarity search. By converting real-valued data features into binary hashing codes, hashing search can be very fast. In the best case [24, 8], the cost of finding similar data is independent of the dataset size. Such a nice property makes hashing outperform many other fast search techniques (e.g. kd-

tree) [8]. Hence it is promising to consider utilizing hashing to speed up the preference ranking. In previous work [37], Zhou et al. directly applied traditional hashing methods for similarity search, and significant speedup (e.g. 100 times) was reported. We refer to their method as Zhou's method. However, an overlooked fact is that preference ranking in recommendation is not equivalent to similarity search in traditional hashing. For MF algorithm, users' preferences over items are calculated as the inner product between user and item features. Inner product between two vectors is fundamentally different from their similarity (see Sec. 3). Therefore, Zhou's method [37] suffers great accuracy loss (see Sec. 5), despite the significant speedup.

The proposed hashing algorithm, which we name Preference Preserving Hashing (PPH), provides a new perspective of applying hashing to recommendation. Like traditional hashing, PPH also converts real-valued user and item features into binary hashing codes. Yet differently, the hashing code in PPH is designed to preserve users' preferences over items (ranking order of inner product), rather than their similarities. In short, our PPH is ranking oriented while traditional hashing is similarity oriented. PPH consists of two novel components, the Constant Feature Norm (CFN) constraint and Magnitude and Phase Quantization (MPQ), which will be elaborated in Sec. 4. Compared with Zhou's method, the proposed PPH algorithm achieves similar speedup, yet the recommendation accuracy is significantly higher. Extensive experiments are conducted on three real world datasets, i.e. MovieLens-1M, MovieLens-10M and the 100M Netflix dataset, and the results clearly demonstrate the advantage of the proposed PPH algorithm.

The rest of the paper is organized as follows. Sec. 2 reviews previous works related with traditional hashing and recommendation efficiency. Sec. 3 discusses why our PPH is not traditional hashing. The algorithm details are given in Sec. 4, and extensive experimental results are presented in Sec. 5. We conclude the paper in Sec. 6 and point out some future work directions.

2. RELATED WORK

2.1 Traditional Hashing for Similarity Search

We call previous hashing works that preserve data similarity *traditional hashing*, which have been widely utilized in applications such as similar images [27, 31, 19] and similar documents [34, 29] search. Generally speaking, traditional hashing consists of two steps: hashing code construction, and fast search in hashing code space.

In hashing code construction, the underlying principle is that similar data should have similar hashing codes. Direct hashing code derivation is NP-hard [31]. Instead, most hashing algorithms first derive real-valued hashing features according to certain criteria, then use quantization methods to get binary hashing codes. Some hashing criteria are data-independent, which means the derived hashing features do not rely on specific data distribution. For example, the famous Locality Sensitive Hashing (LSH) [6] and its variants (e.g. [15]) derive hashing features via random projection. Other criteria are data-dependent, which means the derivation are learned based on certain datasets. Typical algorithms of this kind include Iterative Quantization (ITQ) [7], Kernel Supervised Hashing (KSH) [19], Self-Taught Hashing (STH) [35], Spectral Hashing (SpH) [31], etc.

The process of quantization is quite standard. Usually the real-valued hashing features are binarized w.r.t certain thresholds. Most methods adopt one single threshold for each feature dimension, which is usually set as feature mean or median value [35]. If the data have been normalized to have zero mean, then zero will be the threshold [31]. This is known as entropy maximization principle [27, 35], because such quantization makes the variance of each dimension maximized so that more information can be recorded. Some other works, like Manhattan hashing [12], learn multiple thresholds for each dimension.

The components of PPH, namely CFN and MPQ, correspond to the above two steps accordingly, yet with distinct novelty. First, the CFN constraint forms the criteria of deriving real-valued hashing features, which is actually the derivation of user and item features (preference model). In particular, CFN approximates preference via similarity, while traditional hashing criteria listed above only consider data similarity. Second, the MPQ algorithm explicitly models preference via hashing code, while the above quantization algorithms in traditional hashing are designed to preserve data similarity (via entropy maximization). We will present the details in Sec. 4.

After hashing codes are obtained, searching nearest neighbors is extremely fast. Hamming ranking (e.g. [19]) and hashing lookup [24] are two popular methods. In hamming ranking, data points are sorted by their hamming distances with the query. Although it has linear time complexity w.r.t dataset size, practically it is very fast due to the fast operation of hashing code. In hashing lookup, similar data are searched within a r-radius hamming ball centered at the query hashing code. The time complexity is $\sum_{i=0}^{r} \binom{D}{i}$ with D being the code length, which is constant w.r.t the dataset size. Moreover, we identify a recently proposed method, Multi-Index Hashing (MIH) [21], is more appropriate for the task of efficient recommendation. MIH can be viewed as a generalized hashing lookup method with very nice properties. Further analysis and comparisons of the above search methods are given in Sec. 4.3.

2.2 Efficient Recommendation

As analyzed in Sec. 1, the efficiency of recommendation consists of the efficiency of preference modeling and the efficiency of preference ranking. The former has been extensively studied [33, 5], among which stochastic optimization and parallel computing are most popular.

This paper focuses on the efficiency of preference ranking, which, although critical, attracts much less attention. We categorize previous works as non-hashing methods and hashing methods. We first review non-hashing methods. In [17], Linden et al. discussed the idea of item-space partitioning, which makes recommendation from some subsets of all items. They concluded such a naive strategy would produce recommendations of low quality. User clustering was adopted in [4, 11, 22] so that similar users share the same recommendation results. This strategy essentially reduces the user-space, which may potentially degrade the performance of personalized recommendation. Special data structures, such as kd-tree or variants (e.g. [18, 11]), could also be solutions. However, Koenigstein et al. [11] showed that the speedup using pure tree structure is rather limited, and Grauman et al. [8] showed hashing is much more supe-

rior than trees in many real world applications, particularly those that handle high-dimensional data.

Another potential disadvantage of the above algorithms is that they all need to learn a speedup model (e.g. clusters, trees) based on some training dataset of user and item features. Yet as analyzed in Sec. 1, if users' preferences are frequently updated, the learned speedup model will soon become outdated. Extra computational burdens are imposed to learn new speedup models, which inevitably degrade the efficiency of preference ranking. In comparison, our PPH does not suffer such a problem, as will be analyzed in Sec. 4.4 and Sec. 5.3.1.

When hashing meets recommendation, it provides a new perspective that is fundamentally different from the non-hashing methods. Yet only very limited prior works exist. CF-Budget [10] might be the first work that designed binary hashing code for collaborative filtering. Yet they mainly focused on reducing storage cost, and did not discuss recommendation speedup. Zhou et al. [37] applied hashing technique for efficient recommendation, and the final speedup is substantial (e.g. 100 times was reported). However, as analyzed above, [37] did not realize the difference between preference and similarity. Therefore, they incur much larger accuracy loss than our PPH algorithm.

3. PRELIMINARY

3.1 Problem Statement

Assume from MF algorithm $\mathcal{M}$, we derive a set of user features $\mathcal{U} = \{u_n \in \mathbf{R}^D | n = 1 : N\}$ and a set of item features $\mathcal{V} = \{v_m \in \mathbf{R}^D | m = 1 : M\}$, where D is the feature dimension. From now on, we use notation u, v for real-valued features, while $\tilde{u}, \tilde{v} \in \{\pm 1\}^{D'}$ for binary hashing code of length D'.

In MF, users' preferences over items are calculated as inner product $u^T v$. Therefore, the preference ranking can be expressed as,

$$u : v_1 \succ v_2 \succ ... \succ v_M \longrightarrow u^T v_1 > u^T v_2 > ... > u^T v_M \quad (1)$$

where $u : v_i \succ v_j$ means user u prefers v_i to v_j. Then the inner product of $u^T v_i$ should be larger than $u^T v_j$.

The goal of PPH is to find a hashing method $\mathcal{H}$ to convert $\mathcal{U}, \mathcal{V}$ into binary codes $\tilde{\mathcal{U}}, \tilde{\mathcal{V}} \in \{\pm 1\}^{D'}$, so that (1) the preference order in Eq. 1 can be preserved, and (2) the advantage of fast hashing code operation can be utilized. Further notice, the inner product of two hashing codes $\tilde{u}^T \tilde{v}$ is equivalent to their hamming distance $Ham(\tilde{u}, \tilde{v})$ [1]:

$$Ham(\tilde{u}, \tilde{v}) = \sum_{i=1}^{D} I(\tilde{u}^{(i)} \neq \tilde{v}^{(i)}) = \frac{1}{2}(D - \tilde{u}^T \tilde{v}) \quad (2)$$

Then our goal is to find $\mathcal{H}$ so that we have

$$u^T v_1 > u^T v_2 > ... > u^T v_M \xrightarrow{\tilde{\mathcal{U}}=\mathcal{H}(\mathcal{U}), \tilde{\mathcal{V}}=\mathcal{H}(\mathcal{V})} Ham(\tilde{u}, \tilde{v}_1) < Ham(\tilde{u}, \tilde{v}_2) < ... < Ham(\tilde{u}, \tilde{v}_M) \quad (3)$$

We give the following definition to summarize Eq. 3:

DEFINITION 1. *Function $f(u, v_i)$ is called ranking consistent w.r.t some hashing function $\mathcal{H}$, if the descending ranking order of $f(u, v_i)$ is the same as the ascending ranking order of $Ham(\mathcal{H}(u), \mathcal{H}(v_i))$. Namely, $f(u, v_1) > f(u, v_2)$ if and only if $Ham(\mathcal{H}(u), \mathcal{H}(v_1)) < Ham(\mathcal{H}(u), \mathcal{H}(v_2))$. We use $f \leftrightarrow \mathcal{H}$ to represent the ranking consistency.*

With this definition, the goal of PPH is to find proper $\mathcal{H}$ so that function $f(u, v) = u^T v$ is ranking consistent w.r.t $\mathcal{H}$ as much as possible.

3.2 Why is PPH not traditional hashing?

Eq. 3 of our PPH algorithm is much more challenging than it appears, and directly applying traditional hashing algorithms will not work well. To see why PPH is not traditional hashing, there are two main arguments.

(1) *PPH and traditional hashing focus on different problems.*

To see this, recall that the goal of traditional hashing is to preserve data similarity. Similar data should be assigned similar hashing codes; therefore, large similarity indicates small hamming distance. If we use the same notation as in Eq. 3, and let $sim(u, v)$ be a valid similarity function over u, v, the goal of traditional hashing shall be expressed as

$$sim(u, v_1) > sim(u, v_2) > ... > sim(u, v_M) \xrightarrow{\tilde{\mathcal{U}}=\mathcal{H}(\mathcal{U}), \tilde{\mathcal{V}}=\mathcal{H}(\mathcal{V})} Ham(\tilde{u}, \tilde{v}_1) < Ham(\tilde{u}, \tilde{v}_2) < ... < Ham(\tilde{u}, \tilde{v}_M) \quad (4)$$

However, Eq. 4 is different from Eq. 3, since the following claim.

CLAIM 1. *Inner product is not a valid similarity metric.*

PROOF. For the sake of clarity, here we only present the most evident proof. According to [2, 16], valid similarity metric shall follow self similarity rule, i.e.

$$\text{Given } u, \forall v, sim(u, u) \geq sim(u, v) \quad (5)$$

That means, no other v is more similar with u than u itself. However, $\forall v, u^T u \geq u^T v$ doesn't hold in general. Therefore, inner product is not a valid similarity metric. □

For most hashing algorithms, similarity is defined in the sense of small Euclidean distances (e.g. [31]); for some other works, cosine similarity is utilized (e.g. [35, 20]). It is very easy to verify Claim. 1 w.r.t inner product and Euclidean or cosine similarity. Therefore, traditional hashing and PPH are different algorithms that focus on different problems (Eq. 4 vs Eq. 3).

(2) *It is reasonable to assume $sim(u, v) \leftrightarrow \mathcal{H}$ for some $\mathcal{H}$. Yet $u^T v \leftrightarrow \mathcal{H}$ does not hold in general.*

As analyzed in Sec. 2.1, the underlying assumption in traditional hashing is that, if $sim(u, v)$ is high (e.g. Euclidean or cosine similarity), hashing function $\mathcal{H}$ will produce similar hashing code $\tilde{u}, \tilde{v}$, and small $Ham(\tilde{u}, \tilde{v})$. Therefore, it is reasonable to assume similarity functions are approximately ranking consistent w.r.t some hashing function $\mathcal{H}$. This is the very reason why hashing techniques succeed in similarity search [8].

In contrast, however, since inner product is not a valid similarity metric, there is usually no guarantee to claim that for function $\mathcal{H}$ with two input u, v, if $u^T v$ is high then the output hashing code $\tilde{u}, \tilde{v}$ are similar, thus small hamming distance $Ham(\tilde{u}, \tilde{v})$. Therefore, hashing code and hashing function essentially only preserve data similarity rather than preference.

[1] This holds only when $u^{(i)}, v^{(i)} \in \{\pm 1\}$. At the end we need to convert -1 to 0 for fast hashing code search. But that won't affect the derivation here.

This argument seems quite contradictory to our goal in Eq. 3. Although Eq. 3 cannot be exactly satisfied, a good approximation will be sufficient for our purpose. In the following section, we will show how inner product (approximately) becomes ranking consistent w.r.t certain hashing function under certain constraint.

4. PREFERENCE PRESERVING HASHING

We propose the following two-step principle, under which our Preference Preserving Hashing (PPH) algorithm is designed:

- First, design a special set of $\mathcal{U}, \mathcal{V}$ from $\mathcal{M}$, so that their inner products (preferences) are equivalent to their similarities, i.e. the ranking order of $u^T v$ is approximately the same as the ranking order of $sim(u, v)$.
- Second, transform the derived $\mathcal{U}, \mathcal{V}$ into hashing codes $\tilde{\mathcal{U}}, \tilde{\mathcal{V}}$. $Ham(\tilde{\mathcal{U}}, \tilde{\mathcal{V}})$ will preserve the similarities of $\mathcal{U}, \mathcal{V}$; and because of step 1, this will be equivalent to the preference of $\mathcal{U}$ over $\mathcal{V}$.

We achieve step 1 by proposing a novel Constant Feature Norm (CFN) constraint to MF algorithm $\mathcal{M}$, which will be detailed in Sec. 4.1. Moreover, since the first step cannot be exactly satisfied, in the second step, we propose a novel quantization algorithm that further approximates inner product via hashing code. In general situation, such approximation is difficult to achieve. But with the help of step 1, the approximation can be achieved with satisfactory results. We call it Magnitude and Phase Quantization (MPQ), which will be explained in Sec. 4.2.

4.1 Constant Feature Norm (CFN) constraint

Before giving Constant Feature Norm (CFN) constraint, we first analyze what's the difference between inner product and similarity.

Inner product of real-valued u, v can be expressed as $u^T v = ||u||_F ||v||_F cos\theta_{u,v}$, where $cos\theta_{u,v}$ is a widely adopted similarity metric (cosine similarity). It is the existence of norm $||u||_F, ||v||_F$ that deviates inner product from similarity. This is more clear if we further check the binary hashing codes $\tilde{u}$ and $\tilde{v}$. Since their norms are constantly $\sqrt{D}$, there is $\tilde{u}^T \tilde{v} = D cos\theta_{\tilde{u},\tilde{v}}$. Inner product is now equivalent to cosine similarity, which also indicates Euclidean similarity ($||\tilde{u} - \tilde{v}||_F^2 = 2D - 2\tilde{u}^T\tilde{v}$).

Therefore, to utilize hashing code to preserve inner product, we need to constrain u, v to possess constant norms, i.e. $||u||_F = ||v||_F = c$. In such a way, inner product is totally equivalent to both Euclidean and cosine similarity as

$$u^T v = c^2 cos\theta_{u,v} = c^2 - 0.5 * ||u - v||_F^2. \quad (6)$$

What value should c be? Assume $r_{i,j} \in [0, r_{max}]$ is the rating (continuous or discrete) assigned to item v_j by user u_i, where 0 and r_{max} represent unseen and maximal rating. In MF algorithm $\mathcal{M}$, $u_i^T v_j$ is expected to approach $r_{i,j}$ as much as possible (see Eq. 9). Recall $u^T v = c^2 cos\theta_{u,v} \in [-c^2, c^2]$ while $r_{i,j} \in [0, r_{max}]$. To match the two ranges, we let $c = \sqrt{\frac{r_{max}}{2}}$ so that

$$\frac{r_{max}}{2} + u^T v = \frac{r_{max}}{2} + \frac{r_{max}}{2} cos\theta_{u,v} \in [0, r_{max}] \quad (7)$$

We call this constraint as Constant Feature Norm (CFN) constraint. If CFN is ideally satisfied, then any traditional hashing methods for similarity search can be applied to preserving preference via hashing codes, because similarity is now equivalent to inner product (i.e. preference). Practically, however, exactly satisfying CFN constraint is quite challenging for traditional MF to optimize. Instead of treating CFN as a hard constraint, we treat CFN as a soft regularizer in the objective of $\mathcal{M}$. Based on MF algorithm and the above analysis, we present the final objective as follows:

$$\begin{aligned} \text{MF-CFN} : \mathcal{U}, \mathcal{V} = \arg\min \sum_{i=1}^{N} \sum_{j=1}^{M} I_{i,j} (r_{i,j} - \frac{r_{max}}{2} - u_i^T v_j)^2 \\ + \lambda \big(\sum_{n=1}^{N} (||u_n||_F^2 - \frac{r_{max}}{2})^2 + \sum_{m=1}^{M} (||v_m||_F^2 - \frac{r_{max}}{2})^2 \big) \end{aligned} \quad (8)$$

where the first line is the adapted least square loss function of traditional MF objective [14], and the second line is the soft regularizer of CFN constraint. $I_{i,j} = 1$ if user u_i has rated v_j; otherwise $I_{i,j} = 0$. λ is the regularizer coefficient. To make the comparison clear, we further present the traditional MF objective with L2 regularizer, as well as the objective in [37] with SumZero regularizer: [2]

$$\begin{aligned} \text{MF-L2} : \mathcal{U}, \mathcal{V} = \arg\min \sum_{i=1}^{N} \sum_{j=1}^{M} I_{i,j} (r_{i,j} - u_i^T v_j)^2 \\ + \lambda \big(\sum_{n=1}^{N} ||u_n||_F^2 + \sum_{m=1}^{M} ||v_m||_F^2 \big) \end{aligned} \quad (9)$$

$$\begin{aligned} \text{MF-SumZero} : \mathcal{U}, \mathcal{V} = \arg\min \sum_{i=1}^{N} \sum_{j=1}^{M} I_{i,j} (r_{i,j}^* - 0.5 - \frac{u_i^T v_j}{2D})^2 \\ + \lambda \big(|| \sum_{n=1}^{N} u_n ||_F^2 + || \sum_{m=1}^{M} v_m ||_F^2 \big) \end{aligned} \quad (10)$$

where $r^* \in [0, 1]$ is the rescaled rating.

Since their loss objectives are similar except the regularizer part, from now on we will use CFN, L2 and SumZero to denote the objective in Eq. 8, Eq. 9 and Eq. 10. All these objectives can be easily optimized by coordinate descent.

Remark 1. Different regularizers will constrain $\mathcal{U}, \mathcal{V}$ differently. The key question is, which constraint is more appropriate for hashing code to apply in recommendation? L2 regularizer constrains $\mathcal{U}, \mathcal{V}$ within a sphere, since it penalizes large magnitude of $||\mathcal{U}||_F$, $||\mathcal{V}||_F$. Obviously, it has no direct connection with hashing. SumZero is a common regularizer in traditional hashing (e.g. [31]), and it constrains $\mathcal{U}, \mathcal{V}$ to be symmetric around coordinate origin. In such a way, the derived $\mathcal{U}, \mathcal{V}$ are expected to meet entropy maximization principle [35]. However, this is a principle designed for similarity oriented hashing [31]; while in recommendation, we need to preserve preference rather than similarity. Since $||\mathcal{U}||_F, ||\mathcal{V}||_F$ are constant, CFN regularizer actually constrains $\mathcal{U}, \mathcal{V}$ to reside on the surface of a sphere with radius $\sqrt{\frac{r_{max}}{2}}$. Based on the above analysis, we believe CFN is more appropriate for hashing code to apply in recommendation, which is further supported by experimental results in Sec. 5.

[2] All these regularizers can be easily applied to other loss functions. But since loss function is not the primary concern of this paper, we focus on the least square loss of MF, which is one of the most popular loss functions in recommendation.

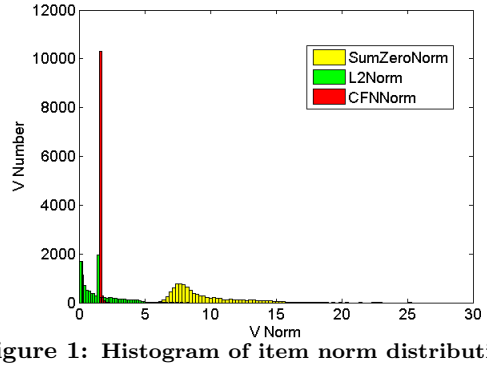


Figure 1: Histogram of item norm distribution.

Remark 2. At first glance it may appear that any c will satisfy Eq. 6. However, inappropriate c will affect the least square regression part in Eq. 8, 9, 10. In contrast, our derivation in Eq. 7 is more reasonable, where we explicitly address the issue.

In Fig. 1 we show the distribution of derived item feature norms (i.e. $||\mathcal{V}||_F$) for all three constraints, where 40 dimension real-valued features are trained on MovieLens-10M dataset. Clearly, CFN generates much more consistent norms than L2 and SumZero. Here only item norms are shown, since user norms will not affect the final preference ranking, as will be analyzed later in Eq. 15. For unseen items in the training set, their features are calculated as the average of seen item features. They form the spike in the curve of L2 (nearby CFN norms) and the spike of SumZero (nearby zero, behind L2 curve).

4.2 Magnitude and Phase Quantization (MPQ)

One drawback of treating CFN as soft regularizer is that it cannot be exactly satisfied, as shown in Fig. 1. Therefore, inner product based preference is still not exactly equivalent to similarity. However, we can expect the difference has been greatly reduced. Thereby, in this section, we will propose a novel quantization method that explicitly models preference within hashing code.

The principle is as follows. Recall $u^T v = ||u||_F ||v||_F cos\theta_{u,v}$. If ideally we obtain $||u||_F = ||v||_F = \sqrt{\frac{r_{max}}{2}}$ via CFN, then only $cos\theta_{u,v}$ is effective in final preference ranking. So our first task is to model $\theta_{u,v}$ in hashing code. Moreover, since CFN cannot be exactly satisfied, we also need to make some adjustment w.r.t $||u||_F, ||v||_F$.

Firstly, we omit $||u||_F, ||v||_F$ and focus on $\theta_{u,v}$. We assume function $\varphi(u,v) = cos\theta_{u,v}$, and we aim to find hashing function $\mathcal{H}^\varphi$ so that $\varphi(u,v)$ is ranking consistent w.r.t $\mathcal{H}^\varphi$. Notice here smaller $\theta_{u,v}$ means higher $\varphi(u,v)$, thus higher inner product $u^T v$. We denote the hashing code of u, v under $\mathcal{H}^\varphi$ as $\tilde{u}^\varphi, \tilde{v}^\varphi$.

Then, we omit $\theta_{u,v}$ and focus on $||u||_F, ||v||_F$. We assume function $\xi(u,v) = ||u||_F ||v||_F$, and again, we aim to find hashing function $\mathcal{H}^\xi$ so that $\xi(u,v)$ is ranking consistent w.r.t $\mathcal{H}^\xi$. Notice here larger $\xi(u,v)$ means higher inner product $u^T v$. We denote the hashing code of u, v under $\mathcal{H}^\xi$ as $\tilde{u}^\xi, \tilde{v}^\xi$.

We propose the following linear approximation as the final hashing code of u, v:

$$\tilde{u} = [\tilde{u}^\xi \ \tilde{u}^\varphi], \tilde{v} = [\tilde{v}^\xi \ \tilde{v}^\varphi] \tag{11}$$

which indicates the overall hashing function $\mathcal{H}$ has:

$$u^T v \leftrightarrow \mathcal{H} = \mathcal{H}^\xi + \mathcal{H}^\varphi \tag{12}$$

where

$$\begin{aligned} Ham(\tilde{u}, \tilde{v}) &= Ham(\mathcal{H}(u), \mathcal{H}(v)) \\ &= Ham(\mathcal{H}^\xi(u), \mathcal{H}^\xi(v)) + Ham(\mathcal{H}^\varphi(u), \mathcal{H}^\varphi(v)) \\ &= Ham(\tilde{u}^\xi, \tilde{v}^\xi) + Ham(\tilde{u}^\varphi, \tilde{v}^\varphi) \end{aligned} \tag{13}$$

Since $||u||_F, ||v||_F$ are expected to be similar (like in Fig. 1), it is reasonable to assume the above linear approximation is sufficient. $||\cdot||_F, \theta$ essentially represent the magnitude and phase information of signals, therefore, we name our algorithm Magnitude and Phase Quantization (MPQ). In Fig. 2 we show an example code. Below we will explain how $\mathcal{H}^\xi, \mathcal{H}^\varphi$ are designed.

4.2.1 Phase Quantization

We first give the phase quantization algorithm for $\mathcal{H}^\varphi$. $\mathcal{H}^\varphi$ should guarantee small hamming distance $Ham(\tilde{u}^\varphi, \tilde{v}^\varphi)$ if $\theta_{u,v}$ is small. One straightforward solution is to encode the quadrants of u, v within hashing code $\tilde{u}^\varphi, \tilde{v}^\varphi$. Formally, for the kth entry of $\tilde{u}^\varphi, \tilde{v}^\varphi \in \{\pm 1\}^D$, we have

$$\begin{aligned} \tilde{u}^\varphi = \mathcal{H}^\varphi(u) = [\tilde{u}^{\varphi(1)}, ..., \tilde{u}^{\varphi(D)}], \quad \tilde{u}^{\varphi(k)} = \begin{cases} 1 & \text{if } u^{(k)} >= 0 \\ -1 & \text{if } u^{(k)} < 0 \end{cases} \\ \tilde{v}^\varphi = \mathcal{H}^\varphi(v) = [\tilde{v}^{\varphi(1)}, ..., \tilde{v}^{\varphi(D)}], \quad \tilde{v}^{\varphi(k)} = \begin{cases} 1 & \text{if } v^{(k)} >= 0 \\ -1 & \text{if } v^{(k)} < 0 \end{cases} \end{aligned} \tag{14}$$

The underlying assumption is that, if many quadrants of u, v are different, then $\theta_{u,v}$ will be large. In practice, we find such an approximation works quite well, as will be shown in Sec. 5.3.2.

4.2.2 Magnitude Quantization

The preference ranking process in Eq. 3 can be decomposed into a series of pairwise comparison. For a specific user u and two items v_1 and v_2, their relative preference is our concern:

$$u^T(v_1 - v_2) = ||u||_F(||v_1||_F cos\theta_{u,v_1} - ||v_2||_F cos\theta_{u,v_2}) \tag{15}$$

We notice that the norm of u does not affect user's preference between v_1 and v_2. [3]. Only $||v||_F$ (and θ) will affect the item ranking for a specific user. So for magnitude quantization, we should focus on $||v||_F$ rather than $||u||_F$.

As shown in Fig. 1, $||v||_F$ are now close to a constant value. This inspires a simple method to model the variance of $||v||_F$. The distribution of $||v||_F$ is viewed as Gaussian. Then if one bit is utilized, the mean value acts as the threshold. If two bits are utilized, then we use $\mu - \sigma$ and $\mu + \sigma$ as two thresholds:

$$\tilde{v}^\xi = \mathcal{H}^\xi(v) = \begin{cases} \{-1, -1\} & \text{if } ||v||_F < \mu - \sigma \\ \{-1, +1\} & \text{if } \mu - \sigma \le ||v||_F \le \mu + \sigma \\ \{+1, +1\} & \text{if } ||v||_F > \mu + \sigma \end{cases} \tag{16}$$

Here range $[\mu - \sigma, \mu + \sigma]$ corresponds to the area of 68.2% probability in Gaussian distribution. Encoding with more bits can be similarly deduced by setting more thresholds. But in practice we find 2 bits are sufficient for magnitude quantization (see Sec. 5.3.3).

One important issue is that, $\tilde{u}^\xi$ cannot be directly quantized by Eq. 16. If both u, v are quantized by Eq. 16, similar $||u||_F, ||v||_F$ will produce similar $\tilde{u}^\xi, \tilde{v}^\xi$, thus small hamming distance and high preference. However, with $||u||_F$ being ineffective (see Eq. 15), it is larger $||v||_F$ that produces larger

[3]In this case, the CFN constraint on $||u||$ in Eq. 8 only serves as regularizer.

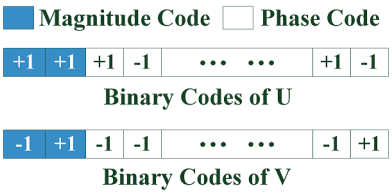


Figure 2: An example of PPH hashing code. There are $D+2$ bits in the final code for $u, v \in R^D$. Notice the magnitude code for u is always $\{+1, +1\}$.

preference. It doesn't matter whether $||v||_F$ is similar with $||u||_F$ or not. Therefore, we propose the following constant magnitude quantization method for u:

$$\tilde{u}^\xi = \mathcal{H}^\xi(u) = \{+1, +1\} \tag{17}$$

u is assigned the same hashing code as the maximum $||v||_F$ in Eq. 16. In such a way, the hamming distance between $\tilde{v}^\xi$ and $\tilde{u}^\xi$ reflects how large $||v||_F$ is. Small hamming distance now equals high $||u||_F||v||_F$, thus high $u^T v$.

4.3 Efficient Top-K Recommendation via Multi-Index Hashing

So far we have elaborated the design of PPH algorithm. Ideally, we expect Eq. 3 is approximately satisfied. Then for user u, the most preferred items v are the nearest neighbors $\tilde{v}$ from $\tilde{u}$. Now we explain the adopted searching method in hashing code space.

As reviewed in Sec. 2.1, hashing lookup is widely utilized, due to its time cost that is independent of dataset size. However, we find it inappropriate for recommendation for two main reasons. First, recall the search complexity of hashing lookup is $\sum_{i=1}^{r} \binom{D}{i}$. Since it depends only on the given radius r, the number of returned items is unpredictable. Usually there are too many items for some users but too few items for the others, as we observe from the experiments in Zhou's method [37]. Yet for many recommendation tasks, the number of returned items is expected to be constant (e.g. Top-K recommendation [32, 30]). Second, hashing lookup will soon become inefficient when code length D or searching radius r becomes large: the exploration space $\binom{D}{r}$ is prohibitive even when $D > 32$ or $r > 2$ [8]. Yet for MF algorithms, longer dimension is usually desired for better recommendation accuracy, and it is very common that $\mathcal{U}, \mathcal{V}$ have dimensions bigger than 32 (e.g. [14, 13]).

Hamming ranking would be a better choice for recommendation. It ranks items according to their hamming distance with the user, and guarantees to return items of fixed number. Speedup is achieved by fast hamming distance computation (i.e. _popcnt function). Yet one disadvantage is that its time complexity is still linear w.r.t dataset size.

We propose to utilize Multi-Index Hashing (MIH)[21] for our scenario. The basic principle of MIH is to split the original hashing code of length D into s subcodes, and conduct hashing lookup for each subcode. For r hamming searching radius, the exploration space of $s * \binom{D/s}{\lceil r/s \rceil}$ is much smaller than $\binom{D}{r}$. For generating recommendations of fixed number, MIH progressively enlarges searching radius, so that a certain number of nearest neighbors can be found. When hashing codes are uniformly distributed, the time complexity of MIH is sub-linear w.r.t the dataset size. Overall, MIH can be viewed as a generalized hashing lookup, with much nicer properties. Please refer to [21] for more details.

4.4 Discussion

Below we give some relevant discussions for the sake of clarity and better understanding.

CFN for MF. PPH contains both CFN and MPQ: the former belongs to preference modeling, while the latter belongs to preference ranking. Although PPH aims to speed up only preference ranking, it modifies the part of preference modeling by introducing CFN to MF objective.

The modification is well justified for two reasons. First, Sec. 5.2.1 shows that the recommendation accuracy of CFN does not degrade too much compared with traditional L2 regularizer. Second, the optimization process of MF-CFN remains the same as MF-L2, which means CFN does not increase the time cost of preference modeling. Therefore, CFN is a reasonable alternative to L2 regularizer, yet more appropriate for hashing code derivation.

MPQ without learning. MPQ is designed to transform real-valued u, v into hashing code $\tilde{u}, \tilde{v}$ without any learning process, which contrasts the trend of learning hashing codes in hashing literature (e.g. SpH [31], STH [35], ITQ [7]).

As analyzed in Sec. 2.2, we argue that the learning process may be harmful to efficient preference ranking. Once users update their preferences, the old speedup model becomes outdated and the learning process has to be repeated. For hashing, the learned hashing function is the speedup model. Previous works (e.g. [37, 22]) neglect the learning cost of speedup model when discussing recommendation efficiency, which is inappropriate. If recommendations are desired to meet users' latest preference models, speedup model learning has to be repeated frequently, whose cost can never be neglected. In contrast, MPQ does not involve any learning process. Therefore, even if users' preferences are updated, they can be readily transformed into hashing code without extra cost. In Sec. 5.3.1 we will show a case study of MPQ vs ITQ (used in Zhou's method [37]) to support our statement.

5. EXPERIMENT

We present experimental results to answer the following questions: (1) Is PPH effective? Namely, how's the recommendation accuracy of PPH compared with traditional MF? (2) Is PPH efficient? Namely, how much faster it can achieve than traditional MF? Other related issues are discussed when appropriate.

5.1 Experimental Settings

5.1.1 *Evaluation Metric*

Recent studies (e.g. [30, 26]) argue that recommendation is better treated as learning to ranking problem, rather than a rating prediction problem. Recommendation is evaluated based on the ranking order of items regardless of their actual preference scores. We find such a perspective of recommendation well fits our goal in Eq. 3. Moreover, it is impossible to recover the exact ratings from hashing code.

We choose NDCG@K [9] as our evaluation metric, which has been widely utilized in learning to rank and ranking oriented recommendation (e.g. [26, 30]). We empirically set $K = 10$, then the average NDCG@10 over all users is our primary evaluation metric of recommendation accuracy.

5.1.2 *Dataset*

We utilize the following three publicly available datasets for experiments.

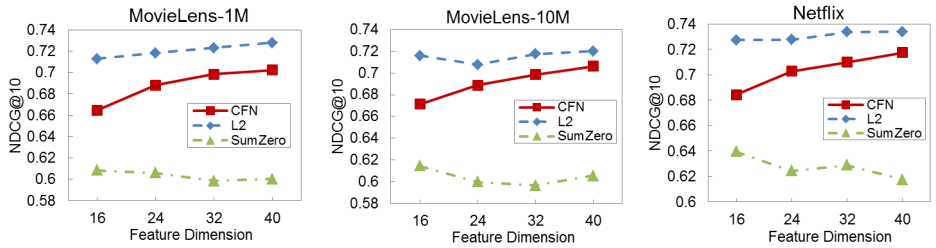


Figure 3: Performance of CFN, L2 and SumZero with real-valued feature $\mathcal{U}, \mathcal{V}$.

- MovieLens-1M. This dataset contains 1,000,209 ratings from 6,040 users to 3,952 movies. The ratings are integers from 1 to 5.
- MovieLens-10M. This dataset contains 10,000,054 ratings from 71,567 users to 10,681 movies. The ratings are from 0.5 to 5 with 0.5 granularity.
- Netflix. This is one of the largest recommendation datasets that are publicly available. 100,480,507 ratings are available from 480,189 users to 17,770 movies. The ratings are integers from 1 to 5.

For all datasets, we select 20 ratings per user for training; 10 ratings for validation, on which λ in Eq. 8, Eq. 9 and Eq. 10 is tuned; the rest (at least 10 items) are used for testing, due to the NDCG@10 evaluation metric. Users having less than 40 ratings are not considered. The dataset split is randomly repeated 5 times to report the average results.

5.1.3 Comparison Methods

As shown in Sec. 4, PPH consists of regularizer (CFN) and quantization (MPQ). The comparison will also occur in these two parts. Specifically,

- For regularizer part, CFN is compared with L2 regularizer for traditional MF and SumZero regularizer for traditional hashing. The corresponding objectives are given in Eq. 8, Eq. 9 and Eq. 10.
- For quantization part, MPQ is compared with Median Threshold (MedThresh) quantization and ITQ quantization. Both MedThresh and ITQ are implemented as in [37]. These two are traditional hashing methods for similarity search.

For the above algorithms, we derive real-valued u, v with four different dimensions, namely, $\{16, 24, 32, 40\}$. Notice MPQ will produce hashing code 2 bits longer than MedThresh and ITQ method. We argue this is a fair comparison, since all of MPQ, MedThresh and ITQ utilize the same real-valued features of the same dimension. Actually, as will be shown in Sec. 5.3.2, even without the two bits of magnitude codes, our phase quantization is already significantly better than MedThresh and ITQ. Interestingly, we find the performance of PPH with dim=16 is already better than other algorithms with dim=40.

5.2 Experimental Results

5.2.1 Performance with Real-Valued Features

We firstly examine the recommendation accuracy with real-valued feature $\mathcal{U}, \mathcal{V}$, as shown in Fig. 3. The accuracy with CFN regularizer is close to the L2 regularizer. Especially when the feature dimension grows high, the difference between CFN and L2 becomes small. This well justifies the statement in Sec. 4.4, that the modification to preference modeling is reasonable.

On contrary, SumZero incurs much larger accuracy loss. It shows that the SumZero constraint, which requires features being symmetric around coordinate origin, is not helpful to maintain the accuracy of MF algorithm. Another possible reason for its large loss is the few training ratings on large scale datasets, while in [37] huge training ratings (up to $20\% \sim 80\%$ of all ratings) on small datasets are utilized. Yet this is very impractical, since recommeder systems cannot ask users to rate too many items before it can make any recommendation. For example, the MovieLens website[4] only requires users to input 15 movies for start up. Therefore, it suggests that SumZero is not suitable for this cold start situation.

5.2.2 Performance with Hashing Code

In Tab. 1, 2, 3, we give the recommendation accuracy with hashing code. For each feature dimension, the highest accuracy among all methods is marked as bold. * indicates the improvement over the second highest accuracy (with subscripted $\circ$) is statistically significant at $p<0.05$. We can make several observations from the results:

CFN vs L2 and SumZero. Compared with L2 and SumZero, CFN achieves higher accuracy with MPQ for all dimensions, and almost all dimensions with MedThresh and ITQ (except dim=16 on Netflix). As analyzed in Sec. 4.1, CFN makes preference resemble similarity. The produced hashing codes will be more suitable to preserve preference than those produced by L2 and SumZero regularizer.

MPQ vs MedThresh and ITQ. For CFN objective, MPQ produces much higher accuracy than MedThresh and ITQ. As analyzed above, MedThresh and ITQ are designed to preserve similarity rather than preference. As a soft regularizer, CFN constraint is not fully satisfied. Therefore, for the produced hashing code of CFN, preference is not exactly equivalent to similarity. That's why MedThresh and ITQ do not work well for CFN, because they are only suitable for preserving similarity. In comparison, MPQ works best because it explicitly models preference within hashing code. Moreover, MPQ only boost the performance of CFN significantly. This is reasonable, because our MPQ is based on the assumption of CFN, that magnitude information is suppressed while phase information takes effect. As shown in Fig. 1, clearly L2 and SumZero do not support the assumption of MPQ, therefore they still do not perform well with MPQ.

Overall. Our PPH algorithm (including CFN and MPQ) achieves the best accuracy on all datasets with various feature dimensions. More noticeably, the accuracy loss is much smaller than Zhou's method in [37] (SumZero+MedThresh and SumZero+ITQ).

[4]http://movielens.umn.edu

Table 1: Performance of Hashing Code on MovieLens-1M

NDCG@10	MedThresh				ITQ				MPQ			
dim	16	24	32	40	16	24	32	40	16	24	32	40
CFN	0.5564	0.5582	0.5583	0.5609	0.5598	0.5629	0.5637	0.5659	**0.637***	**0.642***	**0.651***	**0.652***
L2	0.527	0.5229	0.5275	0.5284	0.5256	0.525	0.5291	0.5292	0.5828$_\circ$	0.5777$_\circ$	0.5809$_\circ$	0.5796$_\circ$
SumZero	0.5403	0.5387	0.5369	0.536	0.5402	0.5385	0.5355	0.5355	0.5782	0.5681	0.5613	0.5578

Table 2: Performance of Hashing Code on MovieLens-10M

NDCG@10	MedThresh				ITQ				MPQ			
dim	16	24	32	40	16	24	32	40	16	24	32	40
CFN	0.5643	0.5683	0.5724	0.5737	0.5651	0.571$_\circ$	0.574$_\circ$	0.577$_\circ$	**0.623***	**0.628***	**0.638***	**0.641***
L2	0.549	0.5529	0.5554	0.5559	0.5527	0.5557	0.5549	0.5533	0.5356	0.5426	0.5436	0.5337
SumZero	0.542	0.5395	0.5391	0.532	0.5324	0.5398	0.5391	0.5338	0.5789$_\circ$	0.5623	0.5536	0.5534

Table 3: Performance of Hashing Code on Netflix

NDCG@10	MedThresh				ITQ				MPQ			
dim	16	24	32	40	16	24	32	40	16	24	32	40
CFN	0.5554	0.5676	0.5673	0.5687	0.5557	0.5647	0.5682	0.571$_\circ$	**0.622***	**0.629***	**0.641***	**0.642***
L2	0.5589	0.56	0.5594	0.554	0.5435	0.5461	0.5497	0.5471	0.5845$_\circ$	0.571$_\circ$	0.5811$_\circ$	0.5705
SumZero	0.5581	0.5549	0.554	0.5443	0.5591	0.555	0.5558	0.5457	0.5605	0.5637	0.577	0.5576

Hashing Code vs Real-Valued Features. Comparison with the results of real-valued features in Fig. 3 reveals the accuracy loss by introducing hashing code to recommendation. Specifically, we compare traditional MF algorithm (L2 objective) with real-valued features and PPH with hashing codes. In best cases (dim=32 or 40), the accuracy loss on MovieLens-1M/10M and Netflix is 0.072, 0.079 and 0.092 (for NDCG@10), which is significantly better than [37]. We argue that this is reasonable accuracy loss, since the task of replacing real-valued features with hashing code of almost the same length is very challenging. We believe there's still room for further improvement, and we will discuss some future works in Sec. 6.

Moreover, the accuracy loss is a necessary trade-off for fast preference ranking. Below, we will present the efficiency experiments to show the benefits of introducing hashing code to recommendation.

5.2.3 *Efficiency Results*

For efficiency experiments, we examine three methods of preference ranking: (1)brute force linear scan with real-valued features u, v, in which inner product (preference) operations are conducted and preference scores are sorted to get top-10 results; (2) hamming ranking, where hamming distance are calculated with hashing code $\tilde{u}, \tilde{v}$ and sorted to get top-10 results; notice hamming distance can only be integers, so sorting degrades to finding items with the smallest hamming distance, which is much faster than sorting real-valued scores; (3) Multi-Index Hashing (MIH) [21], the generalized hashing lookup method; its major parameter is the subcode number s, which we empirically set as $\{1, 2, 2, 3\}$ for dim=$\{16, 24, 32, 40\}$. We do not consider basic hashing lookup because it is inappropriate for recommendation scenario, as explained in Sec. 4.3.

We take the Netflix dataset for illustration. To simulate real world applications, we retrieve top-10 items for every user from a pool of 17,770 candidate items. CFN and MPQ are used for the following efficiency experiments, which are conducted on a computer with AMD 64 bit Opteron CPU and 12G memory. The time cost (in seconds) over all users will be the major criteria of efficiency. Notice no parallel computing is involved, so the time cost reflects overall computational workload.

Time Cost vs Dimension. In Tab. 4 we give the time cost statistics when feature dimension varies on the entire item set (17,770 items). Clearly we can observe the application of hashing code can significantly speed up preference ranking. For linear scan with real-valued features, the time cost of inner product operation is $O(NMD)$, and the cost of top-K item sorting is $O(NM)$. Obviously, the overall cost for real-valued features is huge (42min$\sim$1.6 hours). Hamming ranking is much faster due to the fast calculation of hamming distance. On a AMD 64 bit cpu, hashing codes no longer than 64 bits can be processed by a single function call (_popcnt). Its complexity of finding top-K items is $O(N(K + D))$, as analyzed above. Therefore, the time cost of hamming ranking is constant for all feature dimensions. Yet still, hamming ranking needs to scan all possible items, which may be inefficient. MIH achieves even higher speedup than hamming ranking, because hashing lookup is conducted for each subcode. That means, MIH doesn't need to scan all candidate items.

Time Cost vs Item Number. We further investigate how different preference ranking algorithms behave when item number varies. On Netflix dataset, we randomly select 12.5%, 25%, 50% of the entire 17,770 items, and show the corresponding time cost in Fig. 4. In left figure, linear scan of real-valued features scales linearly w.r.t item number. In comparison, the costs of hamming ranking and MIH are almost negligible. We further show the time cost at a finer scale in the right figure. Now the time cost of real-valued features soon goes beyond the axis range. Meanwhile the difference between hamming ranking and MIH begins to emerge. We can see the cost increase of MIH is slower than hamming ranking, because MIH does not need to scan all candidate items yet hamming ranking does. As shown in [21], MIH has sub-linear time complexity for uniformly distributed hashing codes, which is better than linear complexity of hamming ranking.

In summary, the application of hashing code indeed can speed up preference ranking significantly. Especially with MIH, we achieve the maximal speedup and best scaling complexity w.r.t item number. Similar conclusions are also obtained from MovieLens-1M/10M datasets. Due to space limitation, here we simply omit the detailed time statistics.

Table 4: Time Cost of Top-10 Recommendation

Real-valued feature				
Seconds	dim=16	dim=24	dim=32	dim=40
Time	2564.1	3585.7	4717.3	5716.4
Hamming Ranking (HR)				
Time	153.8	153.7	153.9	153.2
Multi-Index Hashing (MIH)				
Time	4.08	10.54	21.52	27.35
Speedup				
HR vs Real	×16.67	× 23.32	× 30.65	× 37.31
MIH vs Real	× 628.45	× 340.19	× 219.21	× 209.0

5.3 More Experimental Results

We further present more experimental results, regarding details like statement supporting and parameter tuning.

5.3.1 MPQ vs ITQ: A case study of Learning vs Non-Learning of Binary Quantization

To support our argument in Sec. 4.4, in this section we conduct a case study of MPQ vs ITQ, to see how the learning process of ITQ brings extra computational burden to preference ranking. On Netflix dataset, we test the transformation time for CFN objective with 40 dimension. For MPQ, the time cost is 1.014 seconds; while for ITQ, the time cost is 305.03 seconds, 300 times slower than MPQ! Moreover, the learned ITQ model will become outdated if many users update their preferences, which is very common in practice. Thus the learning has to be repeated, and the computational burden will be even heavier.

It is beyond the scope of this paper to discuss when and how users' preferences shall be updated. But for a reasonable recommender system, such updates are believed to be conducted regularly and frequently (e.g. [25]). In this case, non-learning based algorithms (like our MPQ), will have efficiency advantage than learning algorithms (like ITQ).

5.3.2 Magnitude Code vs Phase Code

In Fig. 5 we show how magnitude and phase code contribute to the the overall MPQ performance separately. We can see the actual performances meet our expectation. Due to CFN, all $||v||_F$ become similar. Therefore, phase information contribute most to preserve users' preferences. Yet by incorporating magnitude code, the overall MPQ achieves even better performance than phase code alone. That means, magnitude code still captures a small part of users' preferences, because CFN is only a soft regularizer.

5.3.3 Magnitude Code Bit

In the above experiments, we empirically set the bit number of magnitude code as 2. In Fig. 6(a) we show the recommendation accuracy of CFN+MPQ w.r.t different bit number of magnitude code on Netflix dataset. Clearly, 2 bit has obvious improvement over 1 bit, yet the difference between 2 bit and 3 bit is very small. To make the code length small, we believe 2 bit is a good choice. Again, similar conclusions are also drawn on other datasets.

5.3.4 λ Selection

It's interesting to show how regularizer coefficient λ is selected. In Fig. 6(b) we show the NDCG@10 curve as λ varies on MovieLens-10M validation set, with CFN objective and dim=40 in real-valued features. The curve is very smooth, and the accuracy is quite stable within a large range of λ around the optimal one. This means the CFN objective is very robust w.r.t λ selection. We also observe the accuracy on validation set is much higher than that on testing set, because validation set (10 ratings per user) is much smaller than the testing set. Similar phenomena are also observed for other dimensions and datasets, and we omit them due to space limitation.

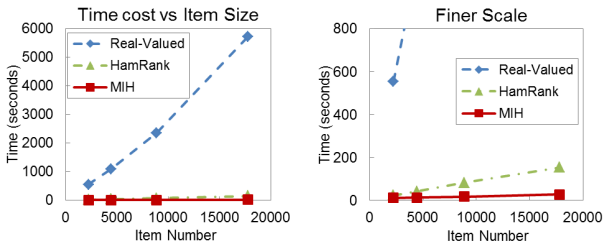


Figure 4: Time cost when item number varies. Left: CFN with 40 feature dimension is utilized on Netflix dataset. Right: left figure re-plotted at a finer scale.

5.3.5 Training with Less Data

Finally, we examine the situation where less training data is utilized. In this section, we select 10, 5 and at least 10 items for training, validation and testing, where users with less than 25 items will be removed. In Fig. 6(c) we show the results of L2 (with real-valued features) and PPH (CFN+MPQ) with 20 and 10 training items for each user. It is clear that by incorporating more training data, the recommendation accuracy can be significantly improved. However, one drawback is that more user input is required. This tradeoff between user effort and recommendation accuracy has to be considered wisely in real world applications.

6. CONCLUSION & FUTURE WORK

In this paper, we propose a novel Preference Preserving Hashing (PPH) algorithm for efficient recommendation. We discuss the key difference between similarity and preference, and formulate PPH to approximate preference via hashing code. Extensive experiments show that our algorithm can achieve hundreds of times speedup than traditional MF algorithm, while suffering much less accuracy loss than previous work of hashing for recommendation.

We believe there's still room for further improvement. For example, special loss functions can be designed that fit hashing code better. Moreover, besides CFN, there may exist other methods to better approximate preference via similarity. It is also interesting to test other recommendation scenarios, like tag [28] or context-aware [23] recommendation, to see if PPH can generalize well. We plan to explore those possibilities in near future.

7. ACKNOWLEDGEMENT

This work is partially supported by NSF research grants IIS-0746830, CNS-1012208, IIS-1017837 and CNS-1314688. This work is also partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

8. REFERENCES

[1] J. Bennett and S. Lanning. The netflix prize. *KDD Cup and Workshop*, 2007.

[2] S. Chen, B. Ma, and K. Zhang. On the similarity metric and the distance metric. *Theoretical Computer Science*, pages 2365–2376, 2009.

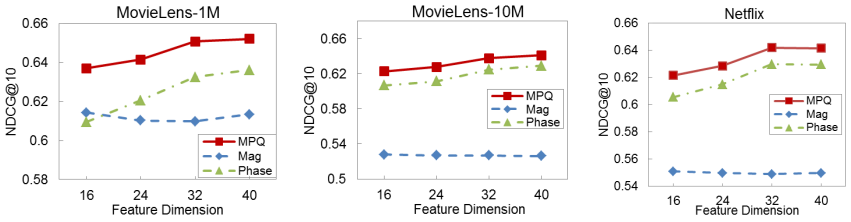
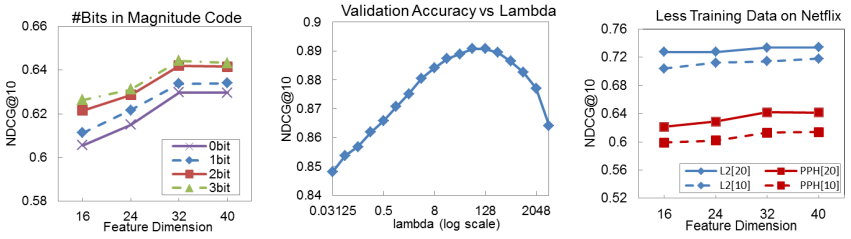


Figure 5: Performance of Magnitude code, Phase code, and overall MPQ code, for CFN objective.



(a) Magnitude Code Bit (b) Lambda Selection (c) Train with Less Data

Figure 6: (a) Accuracy of different magnitude code bits on Netflix dataset. (b) Accuracy vs λ on MovieLens-10M validation set. (c) Training with less item number for each user, which is indicated in brackets.

[3] W. Chen, W. Hsu, and M. L. Lee. Modeling users' receptiveness over time for recommendation. *SIGIR*, pages 373–382, 2013.

[4] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. *WWW*, pages 271–280, 2007.

[5] R. Gemulla, P. Haas, E. Nijkamp, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. *KDD*, pages 69–77, 2011.

[6] A. Gionis, P. Indyk, , and R. Motwani. Similarity search in high dimensions via hashing. *VLDB*, pages 518–529, 1999.

[7] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, pages 2916–2929, 2012.

[8] K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. *MLCV*, pages 49–87, 2013.

[9] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. *SIGIR*, 2000.

[10] A. Karatzoglou, A. Smola, and M. Weimer. Collaborative filtering on a budget. *AISTAT*, pages 389–396, 2010.

[11] N. Koenigstein, P. Ram, and Y. Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. *CIKM*, pages 535–544, 2012.

[12] W. Kong, W. Li, and M. Guo. Manhattan hashing for large-scale image retrieval. *SIGIR*, pages 45–54, 2012.

[13] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *KDD*, pages 426–434, 2008.

[14] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, pages 30–37, 2009.

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. *ICCV*, 2009.

[16] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi. The similarity metric. *IEEE Trans on Information Theory*, pages 3250–3264, 2004.

[17] G. Linden, B. Smith, and J. York. Amazon.com recommendations item-to-item collaborative filtering. *IEEE Internet Computing*, pages 76–80, 2003.

[18] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. *NIPS*, 2005.

[19] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. *CVPR*, pages 2074–2081, 2012.

[20] W. Liu, J. Wang, Y. Mu, S. Kumar, and S. Chang. Compact hyperplane hashing with bilinear functions. *ICML*, 2012.

[21] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, pages 3108–3115, 2012.

[22] E. Ntoutsi, K. Stefanidis, K. Nørvåg, , and H.-P. Kriegel. Fast group recommendations by applying user clustering. *Int'l Conf. on Conceptual Modeling*, pages 126–140, 2012.

[23] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. *SIGIR*, pages 635–644, 2011.

[24] R. Salakhutdinov and G. Hinton. Semantic hashing. *SIGIR*, pages 969–978, 2007.

[25] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. *WWW*, pages 675–684, 2004.

[26] M. N. Volkovs and R. S. Zemel. Collaborative ranking with 17 parameters. *NIPS*, pages 2303–2311, 2012.

[27] J. Wang, S. Kumar, and S. Chang. Semi-supervised hashing for large-scale search. *TPAMI*, pages 2393–2406, 2012.

[28] Q. Wang, L. Ruan, Z. Zhang, and L. Si. Learning compact hashing codes for efficient tag completion and prediction. *CIKM*, pages 1789–1794, 2013.

[29] Q. Wang, D. Zhang, and L. Si. Semantic hashing using tags and topic modeling. *SIGIR*, pages 213–222, 2013.

[30] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. COFI^{RANK}: Maximum margin matrix factorization for collaborative ranking. *NIPS*, 2007.

[31] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *NIPS*, 2008.

[32] X. Yang, H. Steck, Y. Guo, and Y. Liu. On top-k recommendation using social networks. *RecSys*, pages 67–74, 2012.

[33] H. Yu, C. Hsieh, S. Si, and I. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. *ICDM*, pages 765–774, 2012.

[34] D. Zhang, F. Wang, and L. Si. Composite hashing with multiple information sources. *SIGIR*, pages 225–234, 2011.

[35] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. *SIGIR*, pages 18–25, 2010.

[36] G. Zhao, M. L. Lee, W. Hsu, and W. Chen. Increasing temporal diversity with purchase intervals. *SIGIR*, pages 165–174, 2012.

[37] K. Zhou and H. Zha. Learning binary codes for collaborative filtering. *KDD*, pages 498–506, 2012.

[38] P. Zigoris and Y. Zhang. Bayesian adaptive user profiling with explicit & implicit feedback. *CIKM*, pages 397–404, 2006.