Learning to Hash for Recommendation with Tensor Data

Qiyue Yin, Shu Wu and Liang Wang

Center for Research on Intelligent Perception and Computing,
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
{qyyin,shu.wu,wangliang}@nlpr.ia.ac.cn

Abstract. Recommender systems usually need to compare user interests and item characteristics in the context of large user and item space, making hashing based algorithms a promising strategy to speed up recommendation. Existing hashing based recommendation methods only model the users and items and dealing with the matrix data, e.g., user-item rating matrix. In practice, recommendation scenarios can be rather complex, e.g., collaborative retrieval and personalized tag recommendation. The above scenarios generally need fast search for one type of entities (target entities) using multiple types of entities (source entities). The resulting three or higher order tensor data makes conventional hashing algorithms fail for the above scenarios. In this paper, a novel hashing method is accordingly proposed to solve the above problem, where the tensor data is approached by properly designing the similarities between the source entities and target entities in Hamming space. Besides, operator matrices are further developed to explore the relationship between different types of source entities, resulting in auxiliary codes for source entities. Extensive experiments on two tasks, i.e., personalized tag recommendation and collaborative retrieval, demonstrate that the proposed method performs well for tensor data.

Keywords: Learning to hash; Tensor data; Recommendation

1 Introduction

To reduce the information overload and help users select potential appealing products, recommender systems play an important role by recommending relevant products to users. Till now, promising progresses have been made from both academia and industry. Generally, recommendation needs to compare a large number of items to determine the users' most preferred ones and this process will be very time consuming in the context of large user and item space. Efficiency in recommendation is accordingly becoming a challenging problem.

Several works have been proposed dealing with the efficiency problem, which can be roughly classified into data structure based methods and hashing based methods [22]. As for the data structure based methods, researchers try to shrink the search space to reduce the comparison [20] [18] [7]. Typical examples are the kd-tree methods and some other partition methods [6] [8] [7]. Recently, hashing based methods show more superior than the data structure based methods [11]. For example, they can obtain low time complexity that is irrespective of the size of datasets and they need little memory space for the storage of the binary hash codes. Few researchers have tried using the hashing technology for fast recommendation [13] [25] [24]. Readers can refer to Section 2 for more details.

To the best of our knowledge, exiting hashing based recommendation methods only learn binary codes for users and items, and they mainly focus on matrix data. For example, they learn hash codes for users and items so that their similarity in Hamming space can approach the rating matrix. In practice, recommendation can be rather complex to be represented by matrix data, for example, recommending items to users under specific queries in collaborative retrieval [17] [4] and recommending tags to images by specific users in personalized tag recommendation [10]. All these scenarios results in three or higher order tensor data with each dimension indicating one type of entities. Taking personalized tag recommendation as an example, the resulting three order tensor has values 1 or 0 indicating whether a tag is related to an image by a specific user or not.

In summary, the above scenarios involve fast search for one type of entities (target entities) using multiple types of entities (source entities) and usually tensor data is given representing the implicit or explicit relationship between the source entities and target entities. Then conventional hashing based recommendation methods may fail for their disability dealing with tensor data. Besides, traditional hashing methods developed for multiple types of entities, i.e., cross-model hashing methods [9] [15], usually model the relationship between each two types of entities, which is different from our scenario and cannot be applied directly. Readers can refer to Section 2 for more details. Hence, in this paper, a novel hashing method for tensor data is proposed. We learn hash codes for each dimension of the tensor data. To preserve the relationships between source entities and target entities, their similarity calculation in the Hamming space is properly designed by bringing in operator matrices. Finally, extensive experiments on two complex recommendation tasks validate our proposed method.

Our contributions: 1) To the best of our knowledge, this is the first time to learn hash codes for tensor data and meanwhile it is successfully applied to complex recommendation problems. 2) We design operator matrices to explore the relationship between different types of source entities and auxiliary codes are accordingly constructed to enhance the recommendation performance. 3) We test our model in terms of personalized tag recommendation and collaborative retrieval, and extensive experimental results validate our proposed method.

2 Related work

In the context of large user and item space, it becomes urgent to improve the recommendation efficiency when compares user interests and item characteristics. As described before, the methods can be roughly classified into data structure based methods [20] [18] [19] [7] and hashing based methods [25] [24]. As for the former ones, some researches utilize simple partitioning based methods, which partition the items or users into groups [6] [8]. These methods can reduce the item space or user space, but they may inevitable harm the recommendation performance. Recently, some special data structures, such as kd-tree and its variants are utilized for fast search [7] [5]. However, these methods are still time consuming when dealing with high dimensional and large scale datasets [3].

Recently, hashing technology has been brought to recommendation field and shows superior than the data structure based methods. Generally, hashing is one kind of the most popular methods for large scale nearest neighbor search problems, which learns binary codes as the new representations of entities [11]. Initiated by Locality Sensitive Hashing (LSH) [1], which is a family of hash functions mapping similar points to the same hash codes with high probability, some machine learning methods are now employed to design more effective hash codes [16] [2] [23]. As in recommendation field, Zhou and Zha [25] gave the first try and learned binary codes for users and items for fast item ranking. Zhang et al. [24] learned hash codes given the implict or explicit feedback matrix to preserve the users' preferences over items. Furthermore, Wang et al. [13] [12] proposed to learn compact hashing codes for efficient tag completion and recommendation. Generally, previous hashing based recommendation methods mainly focus on the matrix data, e.g., user-item rating matrix and image-tag tag matrix. However, recommendation scenarios are more complex and usually tensor data is given, making conventional hashing methods failed. This is the motivation that promotes us to propose new hashing methods for tensor data.

Since we are learning to hash for multiple modalities, the research of cross-model hashing proposed to meet the need of similarity search across different types of entities is one of the most related works [9] [15]. However, exiting cross-model hashing methods mainly focus on modeling the pairwise relationship, i.e., the relationship between two types of entities. For example, when people uses texts, images and videos for cross-model retrieval, the pairwise relationships between texts and images, between texts and videos and between images and videos are developed. Instead we are concentrating on the scenarios that use multiple types of entities to retrieve one type of entities and typical examples are personalized tag recommendation and collaborative retrieval. Generally, different scenarios lead to distrinct hashing methods to adjust their data structures, which makes previous cross-model hashing methods cannot be applied directly to solve the problem we are focusing on. Thus, we develop new hashing methods for multiple modalities and meanwhile enable for fast recommendation.

3 Model

3.1 Notations

We are given n order tensor data R with each dimension representing a type of entities and their values indicating the relationships between the $n-1$ types of entities and the other one type of entities. We call the $n-1$ types of entities source entities and the other one type of entities target entities. Our goal is to learn hash codes for each type of entities to preserve their relationship and meanwhile enable for fast similarity search. For example, for personalized tag recommendation, we have three types of entities: users and images as source entities and tags as target entities. Accordingly, a three order tensor is given indicating whether a tag is annotated to an image under a specific user or not. We need to learn hash codes for each user, each image and each tag so that tags can be rapidly recommended to images given specific users.

Suppose the $n-1$ source entities and the target entities are represented as $X^{(k)}\,(k=1,...,n-1)$ and $X^{(n)}$ respectively and $m_k(k=1,...,n)$ denotes the number of entities of the k-th type. Then we need to learn hash codes $H^{(k)}\,(k=1,...,n)$ for all of them. Apart from the binary constraint on $H^{(k)}$, the key issue is to keep the similarity between the source entities and the target entities in Hamming space so that their relationship is consistent with the n order tensor data R. In the following, we will elaborate our strategy achieving this goal.

3.2 Similarity preserving

Since the tensor data R indicates the implict or explicit relationship between all the source entities and the target entities, a natural way to calculate similarity between them is to compute the similarity between each type of source entities and the target entities and then sum them. By doing so, the relationship between the source entities and the target entities can be explored in a straight-forward manner. Then for the $s_1,...s_n$-th value of the tensor data R, its predicted value $\hat{R}_{s_1,...s_n}$ can be obtained by:

$$\hat{R}_{s_1,...s_n} = \sum_{k=[1:n-1]} sim(H^{(k)}_{s_k}, H^{(n)}_{s_n}) \tag{1}$$

where s_k is an indictor of an entity of the k-th type and $H^{(k)}_{s_k}$ is the s_k-th hash code for the k-th entity in the k-th type. sim is an operator calculating the similarity between two hash codes to be introduced later.

In the above formulation, we directly consider the relationship between each type of source entities and the target entities. However, the interaction between different types of source entities are ignored, which may be essential to explore the relationship between the source entities and the target entities. For example, for collaborative retrieval, users behave quite different under different queries and clearly queries can influence users. To model the influence, we propose operator matrix for each source entity. For example, $T^{(j)}_{s_j}$, served as the operator matrix of the s_j-th entity in the j-th type, represents the influence this entity will bring to other types of source entities. Furthermore, given an entity $R_{s_1,...s_n}$, we model the influence the other source entities to the s_k-th source entity of the k-th type as $\sum_{j\neq k} T^{(j)}_{s_j}$. By multiplying this matrix with the hash codes of the s_k-th entity in the k-th type and binary it, we can obtain a new hash codes as auxiliary codes for $H_{s_k}^{(k)}$:

$$aux(H_{s_k}^{(k)}) = sign\left(\sum_{j=[1:n-1],j\neq k} T_{s_j}^{(j)} H_{s_k}^{(k)}\right) \tag{2}$$

where *sign* is an element-wise symbolic operation for a vector with each element returning 1 or -1 based on whether its value is bigger than 0 or not.

Taking the interaction between different types of source entities into consideration, we modify Equation 1 as:

$$\hat{R}_{s_1,\ldots s_n} = \sum_{k=[1:n-1]} sim(H_{s_k}^{(k)}, H_{s_n}^{(n)}) + \mu \sum_{k=[1:n-1]} sim\left(aux(H_{s_k}^{(k)}), H_{s_n}^{(n)}\right) \tag{3}$$

where μ is a positive value balancing the effect of the auxiliary similarity term.

In Equation 3, a natural way to define the similarity between two hash codes is the fraction of common bits between them, which is widely used in many hashing based methods [25] [13]. As an example, given hash codes $H_{s_k}^{(k)}$ and $H_{s_n}^{(n)}$, their similarity is defined as:

$$sim\left(H_{s_k}^{(k)}, H_{s_n}^{(n)}\right) = \frac{1}{B}\sum_{i=1}^{B} \mathrm{I}\left(\left(H_{s_k}^{(k)}\right)_i, \left(H_{s_n}^{(n)}\right)_i\right) = \frac{1}{2} + \frac{1}{2B}\left(H_{s_k}^{(k)}\right)^T H_{s_n}^{(n)} \tag{4}$$

where B is the number of bits of the hash codes and $\left(H_{s_k}^{(k)}\right)_i$ is the i-th bit of $H_{s_k}^{(k)}$. I is an indicator function returning 1 if the two operator numbers are the same otherwise 0.

After the calculation of similarity between the source entities and the target entities, many kinds of loss functions can be applied to construct the objective. Here we just use the simple square loss and it is written as

$$\min_{\{H^{(k)}\}_{k=1}^{n}, \{T^{(i)}\}_{i=1}^{n-1}} \sum_{(s_1\ldots s_n)\in \mathrm{O}} (R_{s_1\ldots s_n} - \sigma \hat{R}_{s_1\ldots s_n})^2 + \lambda \sum_{i=[1:n-1]} \sum_{j\in\{s_j\}} ||T_j^{(i)}||^2 \tag{5}$$

where O is the observed entities of the tensor data R. σ is a scaler parameter to limit the predicted similarity to be in the interval [0,1]. The last term is a regular regularization on the operator matrices with a positive tradeoff parameter λ.

As for the constraints imposed on the hash codes, apart from the constraint that the elements of all the hash codes are {-1, +1}, it is necessary to restrict the hash codes to be balanced, which makes sure that each bit carries as much information as possible. In summary, the constraints are defined as $H^{(k)} \in \Omega^{(k)}, \forall k = 1, ..., n$ with $\Omega^{(k)}$ being:

$$\Omega^{(k)} = \left\{H^{(k)} \in \{-1, 1\}^{B\times m_k}, H^{(k)}\mathbf{1} = 0\right\} \tag{6}$$

where the constraint $H^{(k)}1 = 0$ is used to preserve the balance of each bit.

3.3 Final objective and optimization

Bringing Equation 3, 4 and 6 into Equation 5, we can obtain the final cost function. Because of the binary constraint on the variables, the cost function in Equation 5 is not differentiable. Moreover, the balance constraint and the *sign* operator make the optimization of the objective a non-trivial problem. To solve these problems, we firstly relax the *sign* operator to real values as most hashing leaning methods have done [21] [14]. Furthermore, the two constraints are converted into soft penalty terms as in [9]. Specifically, we add the following two terms to the cost function.

$$\theta_1(\{H^{(k)}\}) = \sum_{k=1}^{n} ||H^{(k)} \odot H^{(k)} - E||^2 \qquad \theta_2(\{H^{(k)}\}) = \sum_{k=1}^{n} ||H^{(k)}1||^2 \tag{7}$$

where $E \in R^{B\times m_k}$ is an all-one matrix and $1 \in R^{m_k \times 1}$ is an all one vector.

And the final cost is written as:

$$\begin{aligned} L = & \sum_{(s_1 s_2 \dots s_n)\in O} (R_{s_1\dots s_n} - \sigma \overset{\wedge}{R_{s_1\dots s_n}})^2 + \\ & \lambda \sum_{i=[1:n-1]} \sum_{j\in\{s_j\}} ||T_j^{(i)}||^2 + \theta_1(\{H^{(k)}\}) + \theta_2(\{H^{(k)}\}) \end{aligned} \tag{8}$$

Since the final cost is in Equation 8 is not jointly convex with respect to all the variables, we use the stochastic gradient descent method to obtain a local optimal solution and the gradients are calculated as

$$\begin{aligned} \frac{\partial L}{\partial H_{s_k}^{(k)}} = & \sum_{\{s_j\}_{j=1}^{n-1}, j\neq k} -2\sigma(R_{s_1\dots s_n} - \sigma \overset{\wedge}{R_{s_1\dots s_n}}) \frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial H_{s_k}^{(k)}} \\ & + 4\theta_1(H_{s_k}^{(k)} \odot H_{s_k}^{(k)} - E_{s_k})H_{s_k}^{(k)} + 2\theta_2 H^{(k)}1 \end{aligned} \tag{9}$$

$$\begin{aligned} \frac{\partial L}{\partial H_{s_n}^{(n)}} = & \sum_{\{s_j\}_{j=1}^{n-1}} -2\sigma(R_{s_1\dots s_n} - \sigma \overset{\wedge}{R_{s_1\dots s_n}}) \frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial H_{s_n}^{(n)}} \\ & + 4\theta_1(H_{s_n}^{(n)} \odot H_{s_n}^{(n)} - E_{s_n})H_{s_n}^{(n)} + 2\theta_2 H^{(n)}1 \end{aligned} \tag{10}$$

$$\frac{\partial L}{\partial T_{s_j}^{(j)}} = \sum_{\{s_k\}_{k=1}^{n-1}, k\neq j} -2\sigma(R_{s_1\dots s_n} - \sigma \overset{\wedge}{R_{s_1\dots s_n}}) \frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial T_{s_j}^{(j)}} + 2\lambda T_{s_j}^{(j)} \tag{11}$$

where the gradient components in the above equation are given as

$$\frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial H_{s_k}^{(k)}} = \frac{1}{2B} H_{s_n}^{(n)} + \frac{u}{2B} \sum_{j=[1:n-1], j\neq k} (T_{s_j}^{(j)})^T H_{s_n}^{(n)} \tag{12}$$

$$\frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial H_{s_n}^{(n)}} = \frac{1}{2B} \sum_{k=[1:n-1]} H_{s_k}^{(k)} + \frac{u}{2B} \sum_{k=[1:n-1]} \sum_{j=[1:n-1], j\neq k} (T_{s_j}^{(j)}) H_{s_k}^{(k)} \tag{13}$$

$$\frac{\partial(\overset{\wedge}{R_{s_1\dots s_n}})}{\partial T_{s_j}^{(j)}} = \frac{u}{2B} H_{s_n}^{(n)} \sum_{k=[1:n-1], k\neq j} (H_{s_k}^{(k)})^T \tag{14}$$

After solving the relaxed optimization problem as in Equation 8, we can obtain real-valued representations for all the entities denoted as $\widetilde{H^{(k)}}(k = 1, ..., n)$. Finally, using the constraints of Equation 6, we can obtain the final hashing codes as:

$$H_{ij}^{(k)} = \begin{cases} 1 & \widetilde{H_{ij}^{(k)}} > median(\widetilde{H_{tj}^{(k)}} : t \in 1 : m_k) \\ -1 & Otherwise \end{cases} \tag{15}$$

where $H_{ij}^{(k)}$ is the j-th bit of the i-th entity in type of k. The whole algorithm is summarized in Algorithm 1.

Algorithm 1 Learning to Hash for Recommendation with Tensor Data

```
Input:
   Observed tensor data $R$, parameters $\lambda$, $\theta_1$, $\theta_2$ and $\mu$.
1: Initialize $H^{(k)}$ $(k = 1, ..., n)$ and $T^{(k)}$ $(k = 1, ..., n-1)$;
2: Optimize Equation 8 using stochastic gradient method with their gradients calcu-
   lated by Equation 9, 10 and 11;
3: Obtain the binary hashing codes using Equation 15 with the relaxed solution ob-
   tained from the above step;
4: Obtain the auxiliary codes using Equation 2;
Output:
   Hash codes for all the entities $H^{(k)}$ $(k = 1, ..., n)$, auxiliary codes for all source
   entities $H^{(k)}$ $(k = 1, ..., n-1)$.
```

4 Experiments

4.1 Datasets

We report experimental results on two widely used recommendation datasets and the statistics of the databases are summarized in Table 1.

Last_fm: It contains music artist listening information and tagging information sampled from Last.fm online music system. Each user has tagged some music artists and data tuples [user, artist, tag] are given. We use this dataset to test the performance of our method in terms of personalized tag recommendation. Similar to [10], we use a p-core[1] for Last_fm to filter the dataset and p is chosen as 20 here.

MovieLens: It is published by GroupLens research group and contains personalized ratings to movies. Besides, the user tagging information is also provided and accordingly data tuples [tag, user, movie] can be obtained. We use this dataset for the experiments of collaborative retrieval. Similar to [4], the most common 50 tags are selected as the genre of the movies and [genre, user, movie] triple are utilized to mimic [query, user, item] triples for collaborative retrieval. Besides, the data preprocessing is the same as in [4].

[1]The p-core of a dataset D is the largest subset of D with the property that each entity has to occur in at least p posts. And a post is defined as a combination of different types of source entities [10].

Last_fm			MovieLens		
# of posts	# of pairs	sparsity	# of posts	# of pairs	sparsity
$19,938$	$54,019$	99.95%	$2,111$	$20,583$	99.54%

Table 1. Information of the Last_fm and MovieLens datasets.

4.2 Experimental settings

To evaluate the performance of the proposed model, we compare our method with the following state-of-the-art hashing based recommendation algorithms.

CFCodeReg: Zhou and Zha [25] proposed to learn binary codes for collaborative filtering. **BCE-FIT:** Wang et al. [13] learned binary codes embedding for tag recommendation. It should be noted that the above two methods learn hash codes for matrix data, i.e., rating matrix and tag matrix respectively. However, for personalized tag recommendation and collaborative retrieval problems, three order tensor data is provided. So we may compress the tensor data into matrix form for a comparison. Like in traditional recommendation, the factors of user and query are ignored in personalized tag recommendation and collaborative retrieval respectively. **LCR-B:** Weston et al. [17] proposed the first latent collaborative retrieval algorithm (LSR). However, the learned latent representations are real values, which makes the comparison with our method unfair. Usually, the learned latent representations are in the interval [-1,+1], so we binary the real values with the constraints in Equation 6 that we have used. The learned binary codes is then compared with our method. **LHTD:** Our proposed **L**earning to **H**ash for recommendation with **T**ensor **D**ata with the similarity calculated in Equation 1. **LHTDi:** Our proposed **L**earning to **H**ash for recommendation with **T**ensor **D**ata with the similarity calculated considering the **i**nteraction between different types of source entities as in Equation 3.

In all the experiments, we use Recall as in [4] as the evaluation metric. For a given test triple $(X_{s_1}^{(1)}, X_{s_2}^{(2)}, ..., X_{s_{n-1}}^{(n-1)}, X_i^{(n)})$, we calculate the similarity for all i and sort the target entities in descending order. Then, we measure Recall@k, which is 1 if the target entity in the top k and 0 otherwise. Finally, the mean Recall@k over the whole test dataset is reported. In the parameter setting, we empirically set $\lambda = 1$ and $\theta_1 = \theta_2 = 0.001$ in all the datasets. As for μ, it controls the importance of the interaction term and is searched in the interval [0, 1]. For all the compared methods, we follow the suggests their authors have given to achieve their best performance. All the experiments are run 10 times with randomly choosing 80% of the observed entities as the training set and the remaining as the testing set.

4.3 Numerical results and analysis

From Figure 1, it can be seen that our method outperforms all the compared methods in both databases in terms of personalized tag recommendation and collaborative retrieval. Compared with LHTD, LHTDi considers the interaction

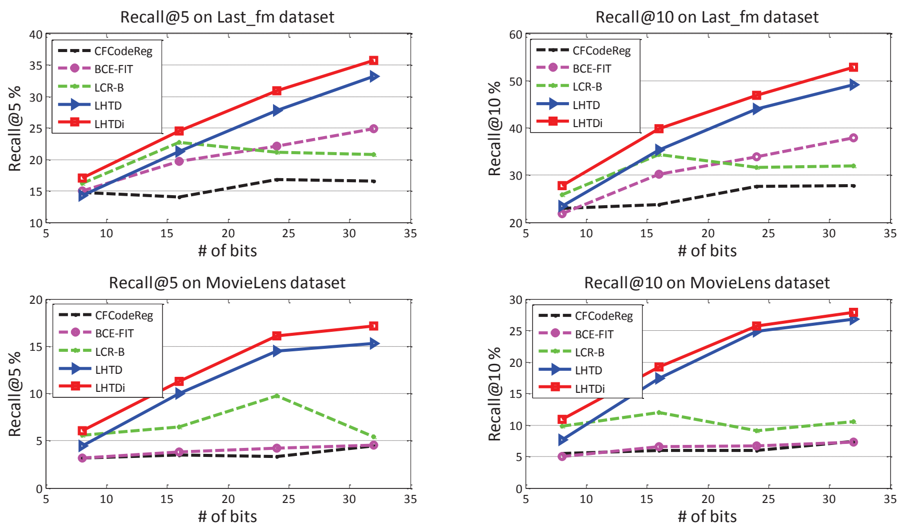

Fig. 1. Recommendation results on Last-fm and MovieLens datasets.

between different types of source entities and the resulting auxiliary codes can well explore the relationship between the source entities, thus its performance is better than that of LHTD.

Compared with our method, CFCodeReg and BCE-FIT cannot consider hashing for all types of entities and this may inevitable harm the relationship between them. So the hash codes learned by their methods can not properly approach the tensor data and their performance is relatively poor. As for LCR-B, it learns latent representations for all types of entities and considers one-side interaction between different types of source entities. Compared with it, our method takes all pairwise interactions between different types of source entities into consideration and can better reflect their relationship. Besides, the hash codes learning of LCR-B is a two-stage process, and more information will be lost compared with our method that considers learning the hash codes in one objective. And these may be the reasons that our method performs better than that of LCR-B.

4.4 Parameter selection

In our model, λ is a regularization parameter used to avoid over-fitting and θ_1 and θ_2 are parameters controlling the soft penalty terms. All these variables are empirically set as in Section 4.2. There is also an important parameter μ that balances the effect of the auxiliary similarity term. And in this section, we test how parameter u influences the performance of our method. We choose the number of hash codes to be 8 and 32 and vary u in the interval [0, 1] on both datasets. The results are shown in Figure 2. When the number of hash codes is small, the performance of LHTDi is becoming better with the increasing of u.

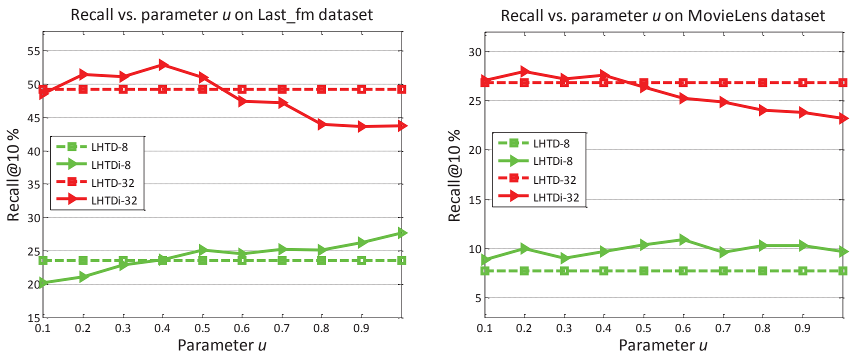

Fig. 2. Recommendation results vs. parameter u on Last_fm and MovieLens datasets. The numbers behind LHTD and LHTDi are the number of bits selected.

However, once the number of hash codes is big enough, we can see that LHTDi reaches the best performance when u is relatively small. This is reasonable because the basic similarity between source entities and target entities can not well approach the tensor data using very few bits and the interaction term is accordingly becoming important. When the number of hash codes is relatively big, the basic hash codes can well embed the information of different entities and the effect of auxiliary codes is accordingly inapparent.

4.5 Operation ability of different types of operator matrices

In our method, we propose operator matrices to model the interaction between different types of source entities, resulting in auxiliary codes to further improve the recommendation performance. In this section, we quantize the effect of the operator matrices and observe their relationship. For an entity of one type, we firstly calculate the change between its operator matrix multiplying the hash codes of the other types of entities and the hash codes of other types of entities themselves. Then we average the changes obtained by all this type of entities as the final operating effect of this type of entities. The comparison between different types of source entities are listed in Table 2.

From Table 2, it can be seen that operator matrices of different types of source entities are not equally important. 1) In personalized tag recommendation, user operator matrices are more important than that of artist. Compared with conventional tagging system, personalized tag recommendation adds the factor of user, which influences the tagging for the same artist. This may be the reason that user operator matrices are the main factors for the interaction. 2) As in collaborative retrieval, we observe that query operator matrices are more important than that of user. Since collaborative retrieval has the entities of queries compared with traditional recommendation, it shares the similar reason as in personalized tag recommendation for the comparison.

Last_fm					MovieLens				
Bit	8	16	24	32	Bit	8	16	24	32
User Operator	3.17	6.59	9.54	12.87	Query Operator	3.59	7.17	10.63	14.31
Artist Operator	2.76	5.80	8.18	11.55	User Operator	3.06	6.34	8.91	12.11

Table 2. Influence of operator matrices on Last_fm and MovieLens databases.

5 Conclusion

In this paper, we have proposed a novel hashing method for tensor data, which retrieve one type of entities (target entities) using the query of many other types of entities (source entities) and has been successfully applied for complex recommendation problems, i.e., personalized tag recommendation and collaborative retrieval. In our model, we learn hash codes for each dimension of entities and properly design the similarity between them to approach the tensor data. Furthermore, to explore the relationship between different types of source entities, operator matrices are developed, resulting in auxiliary codes for all the source entities to further enhance the recommendation performance. Extensive experiments have demonstrated the effectiveness of our method compared with several state-of-the-art hashing algorithms. Since different source entities of the same type may share common characteristics, it may be necessary to build an operator tensor as a dictionary instead of a list of operator matrices to reduce the computation. We leave this as future work.

6 Acknowledgments

This work is jointly supported by National Basic Research Program of China (2012CB316300), and National Natural Science Foundation of China (61175003, 61420106015, U1435221, 61403390).

References

1. Gionis, A., Indyk, P., Motwani, R., et al.: Similarity search in high dimensions via hashing. International Conference on Very Large Data Bases 99, 518–529 (1999)
2. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. Computer Vision and Pattern Recognition pp. 817–824 (2011)
3. Grauman, K., Fergus, R.: Learning binary hash codes for large-scale image search. Machine Learning for Computer Vision pp. 49–87 (2013)
4. Hsiao, K.J., Kulesza, A., Hero, A.: Social collaborative retrieval. International Conference on Web Search and Data Mining pp. 293–302 (2014)
5. Koenigstein, N., Ram, P., Shavitt, Y.: Efficient retrieval of recommendations in a matrix factorization framework. International Conference on Information and Knowledge Management pp. 535–544 (2012)
6. Linden, G., Smith, B., York, J.: Amazon. com recommendations: Item-to-item collaborative filtering. Internet Computing 7(1), 76–80 (2003)

7. Liu, T., Moore, A.W., Yang, K., Gray, A.G.: An investigation of practical approximate nearest neighbor algorithms. Advances in Neural Information Processing Systems pp. 825–832 (2004)
8. Ntoutsi, E., Stefanidis, K., Nørvåg, K., Kriegel, H.P.: Fast group recommendations by applying user clustering. Conceptual Modeling pp. 126–140 (2012)
9. Ou, M., Cui, P., Wang, F., Wang, J., Zhu, W., Yang, S.: Comparing apples to oranges: a scalable solution with heterogeneous hashing. International Conference on Knowledge Discovery and Data Mining pp. 230–238 (2013)
10. Rendle, S., Balby Marinho, L., Nanopoulos, A., Schmidt-Thieme, L.: Learning optimal ranking with tensor factorization for tag recommendation. International Conference on Knowledge Discovery and Data Mining pp. 727–736 (2009)
11. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: A survey. arXiv preprint arXiv:1408.2927 (2014)
12. Wang, Q., Ruan, L., Zhang, Z., Si, L.: Learning compact hashing codes for efficient tag completion and prediction. International Conference on Information & Knowledge Management pp. 1789–1794 (2013)
13. Wang, Q., Shen, B., Wang, S., Li, L., Si, L.: Binary codes embedding for fast image tagging with incomplete labels. European Conference on Computer Vision pp. 425–439 (2014)
14. Wang, Q., Si, L., Zhang, D.: Learning to hash with partial tags: Exploring correlation between tags and hashing bits for large scale image retrieval. European Conference on Computer Vision pp. 378–392 (2014)
15. Wei, Y., Song, Y., Zhen, Y., Liu, B., Yang, Q.: Scalable heterogeneous translated hashing. International Conference on Knowledge Discovery and Data Mining pp. 791–800 (2014)
16. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. Advances in Neural Information Processing Systems pp. 1753–1760 (2009)
17. Weston, J., Wang, C., Weiss, R., Berenzweig, A.: Latent collaborative retrieval. International Conference on Machine Learning (2012)
18. Yin, H., Cui, B., Chen, L., Hu, Z., Huang, Z.: A temporal context-aware model for user behavior modeling in social media systems. pp. 1543–1554 (2014)
19. Yin, H., Cui, B., Chen, L., Hu, Z., Zhou, X.: Dynamic user modeling in social media systems. ACM Transactions on Information Systems 33(3), 10 (2015)
20. Yin, H., Sun, Y., Cui, B., Hu, Z., Chen, L.: Lcars: a location-content-aware recommender system. ACM SIGMOD International Conference on Knowledge discovery and data mining pp. 221–229 (2013)
21. Zhai, D., Chang, H., Zhen, Y., Liu, X., Chen, X., Gao, W.: Parametric local multimodal hashing for cross-view similarity search. International Joint Conference on Artificial Intelligence pp. 2754–2760 (2013)
22. Zhang, D., Yang, G., Hu, Y., Jin, Z., Cai, D., He, X.: A unified approximate nearest neighbor search scheme by combining data structure and hashing. International Joint Conference on Artificial Intelligence pp. 681–687 (2013)
23. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. International Conference on Research and Development in Information Retrieval pp. 18–25 (2010)
24. Zhang, Z., Wang, Q., Ruan, L., Si, L.: Preference preserving hashing for efficient recommendation. International Conference on Research & Development in Information Retrieval pp. 183–192 (2014)
25. Zhou, K., Zha, H.: Learning binary codes for collaborative filtering. International Conference on Knowledge Discovery and Data Mining pp. 498–506 (2012)