11-2017

Indexable Bayesian personalized ranking for efficient top-k recommendation

Dung D. LE
Singapore Management University, ddle.2015@phdis.smu.edu.sg

Hady W. LAUW
Singapore Management University, hadywlauw@smu.edu.sg

Published in CIKM '17: Proceedings of the ACM Conference on Information and Knowledge Management:
Singapore, November 6-10, Pages 1389-1398
https://doi.org/10.1145/3132847.3132913

Indexable Bayesian Personalized Ranking for Efficient Top-k Recommendation

Dung D. Le
Singapore Management University
80 Stamford Road
Singapore 178902
ddle.2015@phdis.smu.edu.sg

Hady W. Lauw
Singapore Management University
80 Stamford Road
Singapore 178902
hadywlauw@smu.edu.sg

ABSTRACT

Top-k recommendation seeks to deliver a personalized recommendation list of k items to a user. The dual objectives are (1) accuracy in identifying the items a user is likely to prefer, and (2) efficiency in constructing the recommendation list in real time. One direction towards retrieval efficiency is to formulate retrieval as approximate k nearest neighbor (kNN) search aided by indexing schemes, such as locality-sensitive hashing, spatial trees, and inverted index. These schemes, applied on the output representations of recommendation algorithms, speed up the retrieval process by automatically discarding a large number of potentially irrelevant items when given a user query vector. However, many previous recommendation algorithms produce representations that may not necessarily align well with the structural properties of these indexing schemes, eventually resulting in a significant loss of accuracy post-indexing. In this paper, we introduce *Indexable Bayesian Personalized Ranking* (INDEXABLE BPR) that learns from ordinal preference to produce representation that is inherently compatible with the aforesaid indices. Experiments on publicly available datasets show superior performance of the proposed model compared to state-of-the-art methods on top-k recommendation retrieval task, achieving significant speedup while maintaining high accuracy.

1 INTRODUCTION

Today, we face a multitude of options in various spheres of life, e.g., deciding which product to buy at Amazon, selecting which movie to watch on Netflix, choosing which article to read on social media, etc. The number of possibilities is immense. Driven by necessity, service providers rely on recommendation algorithms to identify a manageable number k of the most preferred options to be presented to each user. Due to the limited screen real estate of devices (increasingly likely to be ever smaller mobile devices), the value of k may be relatively small (e.g., $k = 10$), yet the selection of items to be recommended are personalized to each individual.

To construct such personalized recommendation lists, we learn from users' historical feedback, which may be explicit (e.g., ratings) [12] or implicit (e.g., click behaviors) [22]. An established methodology in the literature based on matrix factorization [23, 26] derives a latent vector $x_u \in \mathbb{R}^D$ for each user u, and a latent vector $y_i \in \mathbb{R}^D$ for each item i, where D is the dimensionality. The degree of preference of user u for item i is modeled as the inner product ${x_u}^T y_i$. To arrive at the recommendation for u, we need to identify the top-k items with the maximum inner product to x_u.

There are two overriding goals for such top-k recommendations. One is *accuracy*, to maximize the correctness in placing items that user u prefers most into u's recommendation list. Another is *efficiency*; in particular we are primarily concerned with *retrieval effiency*, to minimize the time taken to deliver the recommendation list upon request. Faster retrieval helps the system to cope with a large number of consumers, and minimize their waiting time to receive recommendations. In contrast, learning efficiency or minimizing model learning time, while useful, is arguably less mission-critical, as it can be done offline and involves mainly machine time, rather than human time. Therefore, we seek to keep the learning time manageable, while improving retrieval efficiency.

Many previous recommendation algorithms focus mainly on accuracy. One challenge in practice is the need for exhaustive search over all candidate items to identify the top-k, which is time-consuming when the number of items N is extremely large [11].

Problem. In this paper, we pay equal attention to both goals, i.e., optimizing retrieval efficiency of top-k recommendation without losing sight of accuracy. An effective approach to improve efficiency is to use indexing structures such as locality-sensitive hashing (LSH) [24], spatial trees (e.g., KD-tree [3]), and inverted index [4]. By indexing items' latent vectors, we can quickly retrieve a small candidate set for k "most relevant" items to the user query vector, probably in sub-linear time w.r.t. the number of items N. This avoids an exhaustive search, and saves on the computation for those large number of items that the index considers irrelevant.

Here, we focus on indexing as an alternative to exhaustive search in real time. Indexing is preferred over pre-computation of recommendation lists for all users, which is impractical [4, 11]. User interests change over time. New items appear. By indexing, we avoid the storage requirement of dealing with all possible user-item pairs. Index storage scales only with the number of items N, while the number of queries/users could be larger. Indexing flexibly allows the value of k to be specified at run-time.

However, most of the previous recommendation algorithms based on matrix factorization [12] are not designed with indexing in mind. The objective is to recommend to user u those items with maximum inner product ${x_u}^T y_i$ is not geometrically compatible with aforementioned index structures. For one thing, it has been established that there cannot exist any LSH family for maximum

inner product search [24]. For another, retrieval on a spatial tree index finds the nearest neighbors based on the Euclidean distance, which are not equivalent to those with maximum inner product [2]. In turn, [4] describes an inverted index scheme based on cosine similarity, which again is not equivalent to inner product search.

Approach. The key reason behind the incompatibility between inner product search that matrix factorization relies on, and the aforesaid index structures is how a user u's degree of preference for an item i, expressed as the inner product ${x_u}^T y_i$, is sensitive to the respective magnitude of the latent vectors $||x_u||, ||y_i||$. Therefore, one insight towards achieving geometric compatibility is to desensitize the effect of vector magnitudes. The challenge is how to do so while still preserving the accuracy of the top-k retrieval.

There are a couple of recent approaches in this direction. One approach [2] is a post-processing transformation that expands the latent vectors learnt from matrix factorization with an extra dimensionality to equalize the magnitude of all item vectors. Because the transformation is a separate process from learning the vectors, such a workaround would not be as effective as working with natively indexable vectors in the first place. Another approach [7] extends the Bayesian Probabilistic Matrix Factorization [23], by making the item latent vectors natively of fixed length. Fitting inner product to absolute rating value may not be suitable when only implicit feedback (not rating) is available. Moreover, we note that top-k recommendation is inherently an expression of "relative" rather than "absolute" preferences, i.e., the ranking among items is more important than the exact scores.

We propose to work with ordinal expressions of preferences. Ordinal preferences can be expressed as a triple (u, i, j), indicating that a user u prefers an item i to a different item j. Ordinal representation is prevalent in modeling preferences [22], and also accommodates both explicit (e.g., ratings) and implicit feedback.

Contributions. This paper makes the following contributions:

First, we propose *Indexable Bayesian Personalized Ranking* model or Indexable BPR in short, which produces native geometrically indexable latent vectors for accurate and efficient top-k recommendation. BPR [22] is a generic framework modeling ordinal triples. Each instantiation is based on a specific kernel [8, 13, 16, 20]. [22] had matrix factorization kernel, which is not well-fitted to indexing structures. In contrast, our Indexable BPR is formulated with a kernel based on angular distances (see Section 3). In addition to requiring a different learning algorithm, we will show how this engenders native compatibility with various index structures.

Second, we describe how the resulting vectors are used with LSH, spatial tree, and inverted index for top-k recommendation in Section 4. We conduct experiments with available datasets to compare Indexable BPR with baselines. Empirically, we observe that Indexable BPR achieves a balance of accuracy and run-time efficiency, achieving higher accuracy than the baselines at the same speedup level, and higher speedup at the same accuracy level.

Third, to support the observation on the robustness of Indexable BPR , we provide a theoretical analysis in the context of LSH, further bolstered with empirical evidence, on why our reliance on angular distances results in more index-friendly vectors, smaller loss of accuracy post-indexing, and balanced all-round performance.

2 RELATED WORK

We review the literature related to the problem of efficiently retrieving top-k recommendations using indexing schemes.

Matrix Factorization. Matrix factorization is the basis of many recommendation algorithms [12]. For such models, top-k retrieval is essentially reduced to maximum inner product search, with complexity proportional to the number of items in the (huge) collection. This motivates approaches to improve the retrieval efficiency of top-k recommendation. Of interest to us are those that yield user, item latent vectors to be used with geometric index structures. This engenders compatibility with both spatial tree index and inverted index, as well as with hashing schemes, and transforms the problem into k-nearest neighbor (kNN) search.

One approach is the transformation scheme applied to matrix factorization output. [2, 19] propose a post-processing step that extends the output latent vectors by one dimension to equalize the magnitude of item vectors. Theoretical comparisons show that this Euclidean transformation achieves better hashing quality as compared to the two previous methods in [24] and [25]. However, the Euclidean transformation results in high concentration of new item points, affecting the retrieval accuracy of the approximate kNN. As Indexable BPR relies on ordinal triples, one appropriate baseline is to use the transformation scheme above on a comparable algorithm that also relies on triples. We identify BPR [22] with inner product or matrix factorization (MF) kernel, whose implementation is available[1], and refer to the composite as BPR(MF)+.

Another approach is to learn indexable vectors that fits ratings, which would not work with implicit feedback, e.g., ordinal triples. Indexable Probabilistic Matrix Factorization or IPMF [7] is a rating-based model with constraints to place item vectors on a hypersphere. We will see that IPMF does not optimize for a high mean of the normal distribution in Eq. 15 (see Section 5), and ordinal-based Indexable BPR potentially performs better.

Others may not involve the standard index structures we study. [11] used representative queries identified by clustering. [21] invented another data structure (cone tree). [27–29] learnt binary codes, which are incompatible with l_2 distance used by spatial tree.

Euclidean Embedding. Euclidean embedding takes as input distances (or their ordinal relationships), and outputs low dimensional latent coordinates for each point that would preserve the input as much as possible [14]. Because they operate in the Euclidean space, the coordinates support nearest neighbor search using geometric index structures such as spatial trees.

There exist recent works on using Euclidean embedding to model user preferences over items, which we include as experimental baselines. The first method Collaborative Filtering via Euclidean Embedding or CFEE [10] fits a rating $\hat{r}_{ui}$ by user u on item i in terms of the squared Euclidean distance between x_u and y_i. Fitting ratings directly does not preserve the pairwise comparisons. The second method Collaborative Ordinal Embedding or COE [15] is based on ordinal triples. It expresses a triple t_{uij} through the Euclidean distance difference $||x_u - y_j|| - ||x_u - y_i||$. COE's objective is to maximize this difference for each observation t_{uij}.

[1] http://www.librec.net

3 INDEXABLE BPR

Problem. We consider a set users $\mathcal{U}$ and a set of items $\mathcal{I}$. We consider as input a set of triples $\mathcal{T} \subset \mathcal{U} \times \mathcal{I} \times \mathcal{I}$. A triple $t_{uij} \in \mathcal{T}$ relates one user $u \in \mathcal{U}$ and two different items $i, j \in \mathcal{I}$, indicating u's preferring item i to item j. Such ordinal preference is prevalent, encompassing explicit and implicit feedback scenarios. When ratings are available, we can induce an ordinal triple for each instance when user u rates item i higher than she rates item j. Triples can also model implicit feedback [22]. E.g., when searching on the Web, one may click on the website i and ignore j. When browsing products, one may choose to click or buy product i and skip j.

The goal is to derive a D-dimensional latent vector $x_u \in \mathbb{R}^D$ for each user $u \in \mathcal{U}$, and a latent vector $y_i \in \mathbb{R}^D$ for each item $i \in \mathcal{I}$, such that the relative preference of a user u over two items i and j can be expressed as a function (to be defined) of their corresponding latent vectors x_u, y_i, and y_j. We denote the collection of all user latent vectors and item latent vectors as X and Y respectively.

Framework. Given the input triples $\mathcal{T}$, we seek to learn the user and item vectors X, Y with the highest posterior probability.

$$\arg\max_{X,Y} P(X, Y|\mathcal{T}) \tag{1}$$

The Bayesian formulation for modeling this posterior probability is to decompose it into the likelihood of the triples $P(\mathcal{T}|X, Y)$ and the prior $P(X, Y)$, as shown in Eq. 2.

$$P(X, Y|\mathcal{T}) \propto P(\mathcal{T}|X, Y)P(X, Y) \tag{2}$$

We will define the prior later when we discuss the generative process. For now, we focus on defining the likelihood, which can be decomposed into the probability for individual triples $t_{uij} \in \mathcal{T}$.

$$P(\mathcal{T}|X, Y) = \prod_{t_{uij} \in \mathcal{T}} P(t_{uij}|x_u, y_i, y_j) \tag{3}$$

Weakness of Inner Product Kernel for Top-k Retrieval. To determine the probability for an individual triple, we need to define a kernel function. The kernel proposed by the matrix factorization-based (not natively indexable) BPR [22] is shown in Eq. 4 (σ is the sigmoid function). This assumes that if ${x_u}^T y_i$ is higher than ${x_u}^T y_j$, then user u is more likely to prefer item i to j.

$$P(t_{uij}|x_u, y_i, y_j) = \sigma({x_u}^T y_i - {x_u}^T y_j) \tag{4}$$

Since our intended application is top-k recommendation, once we learn the user and item latent vectors, the top-k recommendation task is reduced to searching for the k nearest neighbors to the query (user vector) among the potential answers (item vectors). A naive solution is to conduct exhaustive search over all the items.

An indexing-based approach could reduce the retrieval time significantly, by prioritizing or narrowing the search to a smaller search space. For the nearest neighbors identified by an index to be as accurate as possible, the notion of similarity (or distance) used by the index should be compatible with the notion of the similarity of the underlying model that yields that user and item vectors.

Therein lies the issue with the inner product kernel described in Eq. 4. It is not necessarily compatible with geometric index structures that rely on similarity functions other than inner products.

First, we examine its incompatibility with spatial tree index. Suppose that all item latent vectors y_i's are inserted into the index. To derive the recommendation for u, we use x_u as the query. Nearest neighbor search on spatial tree index is expected to return items that are closest in terms of Euclidean distance. The relationship between Euclidean distance and inner product is expressed in Eq. 5. It implies that items with the closest Euclidean distances may not have the highest inner products, due to the magnitudes $||x_u||$ and $||y_i||$. Spatial tree index retrieval may be inconsistent with Eq. 4.

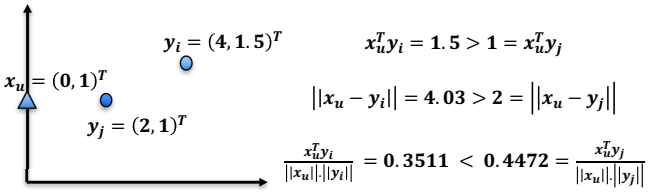


Figure 1: An illustration for the incompatibility of inner product kernel for spatial tree index (Euclidean distance) and inverted index (cosine similarity).

$$||x_u - y_i||^2 = ||x_u||^2 + ||y_i||^2 - 2{x_u}^T y_i \tag{5}$$

Second, we examine its incompatibility with inverted index that relies on cosine similarity (Eq. 6). Similarly, the pertinence of the magnitudes $||x_u||$ and $||y_i||$ implies that inverted index retrieval may be inconsistent with maximum inner product search.

$$cos(x_u, y_i) = \frac{{x_u}^T y_i}{||x_u|| \cdot ||y_i||} \tag{6}$$

Fig.1 shows an example to illustrate the above analysis. In Fig. 1, the inner product ${x_u}^T y_i$ is *greater* than ${x_u}^T y_j$, implying that u prefers i to j. However, the Euclidean distance computation shows that y_j is *closer* to x_u than y_i is to x_u. Also, the cosine similarity between x_u and y_i is *smaller* than that between x_u and y_j. This means that the inner product kernel of the model is not compatible with the operations of a spatial tree index relying on Euclidean distance, or an inverted index relying on cosine similarity.

Third, in terms of its incompatibility with LSH, we note that it has been established that there cannot exist any LSH family for maximum inner product search [24], while there exist LSH families for Euclidean distances and cosine similarity respectively.

Proposed Angular Distance Kernel. To circumvent the limitation of the inner product kernel, we propose a new kernel to express the probability for a triple t_{uij} in a way that is insensitive to vector magnitudes. A different kernel is a non-trivial, even significant, change as it requires a different learning algorithm.

Our proposed kernel is based on angular distance. Let θ_{xy} denote the angular distance between vectors x and y, evaluated as the arccos of the inner product between the normalized vectors.

$$\theta_{xy} = \cos^{-1}(\frac{x^T y}{||x||.||y||}) \tag{7}$$

Proposing the angular distance, i.e., the arccos of the cosine similarity, to formulate the user-item association is a novel and appropriate design choice for the following reasons.

- *Firstly*, since arccos is a monotone function, the closest point according to the angular distance is the same as the point with the highest cosine similarity, resulting in its compatibility with the inverted index structure.

- *Secondly*, since angular distances are not affected by magnitudes, it preserves all the information learnt by the model. Before indexing, the learnt vectors could be normalized to unit length for compatibility with indexing that relies on either Euclidean distance or cosine similarity.
- *Lastly*, the angular distance is also compatible to LSH indexing. A theoretical analysis and empirical evidence on this compatibility is provided in Section 5.

While the user x_u and item y_i vectors we learn could be of varying lengths, the magnitudes are uninformative as far as the user preferences encoded by the triples are concerned. This advantageously allows greater flexibility in parameter learning, while still controlling the vectors via the regularization terms, as opposed to constraining vectors to fixed length during learning (as in [7]).

We formulate the probability of a triple t_{uij} for INDEXABLE BPR as in Eq. 8. The probability is higher when the difference $\theta_{x_u y_j} - \theta_{x_u y_i}$ is larger. If u prefers i to j, the angular distance between x_u and y_i is expected to be smaller than between x_u and y_j.

$$P(t_{uij}|x_u, y_i, y_j) = \sigma(\theta_{x_u y_j} - \theta_{x_u y_i}) \tag{8}$$

Generative Process. The proposed model INDEXABLE BPR as a whole could be expressed by the following generative process:

(1) For each user $u \in \mathcal{U}$: Draw $x_u \sim \text{Normal}(0, \eta^2\mathbf{I})$,
(2) For each item $i \in \mathcal{I}$: Draw $y_i \sim \text{Normal}(0, \eta^2\mathbf{I})$,
(3) For each triple of one user $u \in \mathcal{U}$ and two items $i, j \in \mathcal{I}$:
   - Draw a trial from Bernoulli($\text{P}(t_{uij}|x_u, y_i, y_j)$),
   - If "success", generate a triple instance t_{uij},
   - Otherwise, generate a triple instance t_{uji}.

The first two steps place zero-mean multi-variate spherical Gaussian priors on the user and item latent vectors. η^2 denotes the variance of the Normal distributions; for simplicity we use the same variance for users and items. $\mathbf{I}$ denotes the identity matrix. This acts as regularizers for the vectors, and defines the prior $P(X, Y)$.

$$P(X, Y) = (2\pi\eta^2)^{-\frac{D}{2}} \prod_{u \in \mathcal{U}} e^{-\frac{1}{2\eta^2}||x_u||^2} \prod_{i \in \mathcal{I}} e^{-\frac{1}{2\eta^2}||y_i||^2} \tag{9}$$

Triples in $\mathcal{T}$ are generated from users and items' latent vectors according to the probability $\text{P}(t_{uij}|x_u, y_i, y_j)$ as defined in Eq. 8.

Parameter Learning. Maximizing the posterior as outlined in Eq. 2 is equivalent to maximizing its logarithm, shown below.

$$\begin{aligned}\mathcal{L} &= \ln P(\mathcal{T}|X, Y) + \ln P(X, Y) \\ &\propto \ln \text{P}(\mathcal{T}|X, Y) - \frac{1}{\eta^2} \sum_{u \in \mathcal{U}} ||x_u||^2 - \frac{1}{\eta^2} \sum_{i \in \mathcal{I}} ||y_i||^2 \end{aligned} \tag{10}$$

Let us denote $\Delta_{uij} = \theta_{x_u y_j} - \theta_{x_u y_i}$, $\tilde{x}_u = \frac{x_u}{||x_u||}$ $\forall u \in \mathcal{U}$ and $\tilde{y}_i = \frac{y_i}{||y_i||}$ $\forall i \in \mathcal{I}$. The gradient of $\mathcal{L}$ w.r.t each user vector x_u is:

$$\frac{\partial \mathcal{L}}{\partial x_u} = \sum_{\{i,j:\, t_{uij} \in \mathcal{T}\}} \frac{1}{||x_u||^2} \frac{e^{-\Delta_{uij}}}{1 + e^{-\Delta_{uij}}} \times \left(\frac{-\tilde{y}_j.||x_u|| + \cos(x_u, y_j).x_u}{\sqrt{1 - \cos(x_u, y_j)^2}} - \frac{-\tilde{y}_i.||x_u|| + \cos(x_u, y_i).x_u}{\sqrt{1 - \cos(x_u, y_i)^2}} \right),$$

in which, $\cos(x_u, y_i) = \frac{x_u^T y_i}{||x_u||.||y_i||}$ $\forall u \in \mathcal{U}$ and $\forall i \in \mathcal{I}$.

Algorithm 1 Gradient Ascent for INDEXABLE BPR

Input: Ordinal triples set $\mathcal{T} = \{t_{uij}, \forall u \in \mathcal{U}, i \neq j \in \mathcal{I}\}$.
1: Initialize x_u for $u \in \mathcal{U}$, y_i for $i \in \mathcal{I}$
2: **while** not converged **do**
3: **for** each $u \in \mathcal{U}$ **do**
4: $x_u \leftarrow x_u + \epsilon.\frac{\partial \mathcal{L}}{\partial x_u}$
5: **for** each $i \in \mathcal{I}$ **do**
6: $y_i \leftarrow y_i + \epsilon.\frac{\partial \mathcal{L}}{\partial y_i}$
7: Return $\{\tilde{x}_u = \frac{x_u}{||x_u||}\}_{u \in \mathcal{U}}$ and $\{\tilde{y}_i = \frac{y_i}{||y_i||}\}_{i \in \mathcal{I}}$

The gradient of $\mathcal{L}$ w.r.t each item vector y_k is:

$$\frac{\partial \mathcal{L}}{\partial y_k} = \sum_{\{u,j:\, t_{ukj} \in \mathcal{T}\}} \frac{1}{||y_k||^2} \frac{e^{-\Delta_{ukj}}}{1 + e^{-\Delta_{ukj}}} . \frac{\tilde{x}_u.||y_k|| - \cos(x_u, y_k).y_k}{\sqrt{1 - \cos(x_u, y_k)^2}}$$
$$+ \sum_{\{u,i:\, t_{uik} \in \mathcal{T}\}} \frac{1}{||y_k||^2} \frac{e^{-\Delta_{uik}}}{1 + e^{-\Delta_{uik}}} . \frac{-\tilde{x}_u.||y_k|| + \cos(x_u, y_k).y_k}{\sqrt{1 - \cos(x_u, y_k)^2}} .$$

Algorithm 1 describes the learning algorithm with full gradient ascent. It first initializes the users and items' latent vectors. In each iteration, the model parameters are updated based on the gradients, with a decaying learning rate ϵ over time. The output is the set of normalized user vectors $\tilde{x}_u$ and item vectors $\tilde{y}_i$. On one hand, this normalization does not affect the accuracy of the top-k recommendation produced by INDEXABLE BPR , since the magnitude of the latent vectors does not affect the ranking. On the other hand, normalized vectors can be used for approximate kNN search using various indexing data structures later. The time complexity of the algorithm is linear to the number of triples in $\mathcal{T}$, i.e., $O(|\mathcal{U}| \times |\mathcal{I}|^2)$.

4 EXPERIMENTS ON TOP-K RECOMMENDATION WITH INDEXING

The key idea in this paper is achieving speedup in the retrieval time of top-k recommendation via indexing, while still maintaining high accuracies via better representations that minimize any loss of information post-indexing. Hence, in the following evaluation, we are interested in both the accuracy of the top-k recommendation returned by the index, and the speedup in retrieval time due to indexing as compared to exhaustive search.

To showcase the generality of INDEXABLE BPR in accommodating various index structures, we experiment with three indexing schemes: locality-sensitive hashing, spatial tree index, and inverted index. Note that our focus is on the relative merits of recommendation algorithms, rather than on the relative merits of index structures. It is our objective to investigate the effectiveness of INDEXABLE BPR , as compared to other algorithms, for top-k recommendation when using these index structures. Yet, it is *not* our objective to compare the index structures among themselves.

Comparative Methods. We compare our proposed INDEXABLE BPR with the following recommendation algorithm baselines:

- BPR(MF): the non-index friendly BPR with inner product (MF) kernel [22]. This would validate whether our angular distance kernel is more index-friendly.

Table 1: Datasets

	#users	#items	#ratings	#training ordinal triples
MovieLens 20M	138,493	27,278	20,000,263	5.46×10^8
Netflix	480,189	17,770	100,480,507	2.29×10^{10}

- BPR(MF)+: a composite of BPR(MF) and the Euclidean transformation described in [2] to make the item vectors indexable as post-processing. This allows validation of our learning inherently indexable vectors in the first place.
- IPMF: matrix factorization that learns fixed-length item vectors but fits rating scores [7]. This allows validation of our modeling of ordinal triples.
- CFEE: Euclidean embedding that fits rating scores [10]. This allows validation of our modeling of ordinal triples.
- COE: Euclidean embedding that fits ordinal triples [15]. Comparison to CFEE and COE allows validation of our compatibility with non-spatial indices such as some LSH families as well as inverted index.

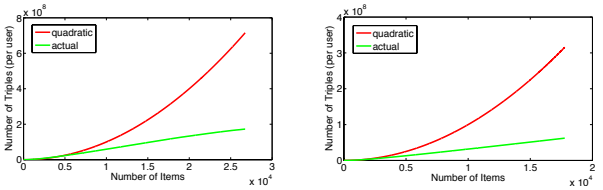
We tune the hyper-parameters of all models for the best performance. For IPMF, we adopt the parameters provided by its authors for *Netflix* dataset. For the ordinal-based algorithms (BPR, COE, and Indexable BPR), the learning rate and the regularization are 0.05 and 0.001. For CFEE, they are 0.1 and 0.001. All models use $D = 20$ dimensionalities in their latent representations. Similar trends are observed across other dimensionalities (see Sec. 5).

Datasets. We experiment on two publicly available rating-based datasets and derive ordinal triples accordingly. One is *MovieLens 20M*[2], the largest among the MovieLens collection. The other is *Netflix*[3]. Table 1 shows a summary of these datasets. By default, *MovieLens 20M* includes only users with at least 20 ratings. For consistency, we apply the same to *Netflix*. For each dataset, we randomly keep 60% of the ratings for training and hide 40% for testing. We conduct stratified sampling to maintain the same ratio for each user. We report the average results over five training/testing splits. For training, we generate a triple t_{uij} if user u has higher rating for item i than for j, and triples are formed within the training set.

As earlier mentioned, our focus in this work is on online retrieval speedup. We find that the model learning time, which is offline, is manageable. Our learning times for *MovieLens 20M* and *Netflix* are 5.2 and 9.3 hours respectively on a computer with Intel Xeon E2650v4 2.20GHz CPU and 256GB RAM. Algorithm 1 scales with the number of triples, which in practice grows slower than its theoretical complexity of $O(|\mathcal{U}| \times |\mathcal{I}|^2)$. Figure 2 shows how the average number of triples per user grows with the number of items, showing that the actual growth is closer to linear and lower than the quadratic curve provided as reference.

Recall. We assume that the goal of top-k recommendation is to recommend *new* items to a user, among the items not seen in the training set. When retrieval is based on an index, the evaluation of top-k necessarily takes into account the operation of the index. Because we maintain one index for all items to be used with all

[2]http://grouplens.org/datasets/movielens/20m/
[3]http://academictorrents.com/details/9b13183dc4d60676b773c9e2cd6de5e5542cee9a



(a) MovieLens 20M **(b) Netflix**

Figure 2: Number of triples (per user) vs. number of items.

users, conceivably items returned by a top-k query may belong to one of three categories: those in the training set (to be excluded for new item recommendation), those in the test set (of interest as these are the known ground-truth of which items users prefer), and those not seen/rated in either set (for which no ground-truth of user preference is available). It is important to note the latter may not necessarily be bad recommendations, they are simply unknown. Precision of the top-k may penalize such items.

We reason that among the rated items in the test set, those that have been assigned the maximum rating possible by a user would be expected to appear in the top-k recommendation list for that user. A suitable metric is the recall of items in the test set with maximum rating. For each user u with at least one highest rating item in the test set (for the two datasets, the highest possible rating value is 5), we compute the percentage of these items that are returned in the top-k by the index. The higher the percentage, the better is the performance of the model at identifying the items a user prefers the most. Eq. 11 presents the formula for *Recall@k*:

$$\text{Recall@}k = \frac{1}{|\mathcal{U}_{\max}|} \sum_{u \in \mathcal{U}_{\max}} \frac{|\{i \in \psi_k^u : r_{ui} = \max \text{ rating}\}|}{|\{i \in \mathcal{I} : r_{ui} = \max \text{ rating}\}|}, \quad (11)$$

in which $\mathcal{U}_{\max}$ is the set of users who have given at least one item with rating of 5 and ψ_k^u is the top-k returned by the index. We exclude training items for u from both numerator and denominator. We normalize *Recall@k* with the ideal *Recall@k* that a perfect algorithm can achieve, and denote the metric as *nRecall@k*.

Speedup. To investigate the efficacy of using the indexing schemes for top-k recommendation, we introduce the second metric *speedup*, which is the ratio between the time taken by exhaustive search to return the top-k, to the time taken by an index.

$$\text{Speedup} = \frac{\text{Retrieval time taken by exhaustive search}}{\text{Retrieval time taken by the index}}. \quad (12)$$

We will discuss the results in terms of trade-off between recall and speedup. There are index parameters that control the degree of approximation, i.e., higher speedup at the expense of lower recall. Among the comparative recommendation algorithms, a better trade-off means higher speedup at the same recall, or higher recall at the same speedup. For each comparison below, we control for the indexing scheme, as different schemes vary in ways of achieving approximation, implementations, and deployment scenarios.

4.1 Top-k Recommendation with LSH Index

We first briefly review LSH and how it is used for top-k recommendation. Let $\mathbf{h} = (h_1, h_2, \ldots, h_b)$ be a set of LSH hash functions.

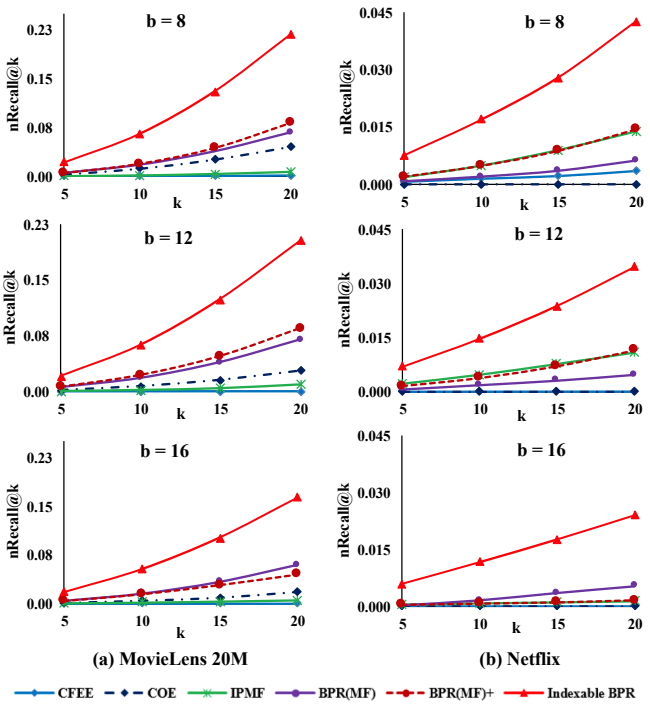


Figure 3: nRecall@k with Hash table lookup strategy ($T = 10$ hash tables).

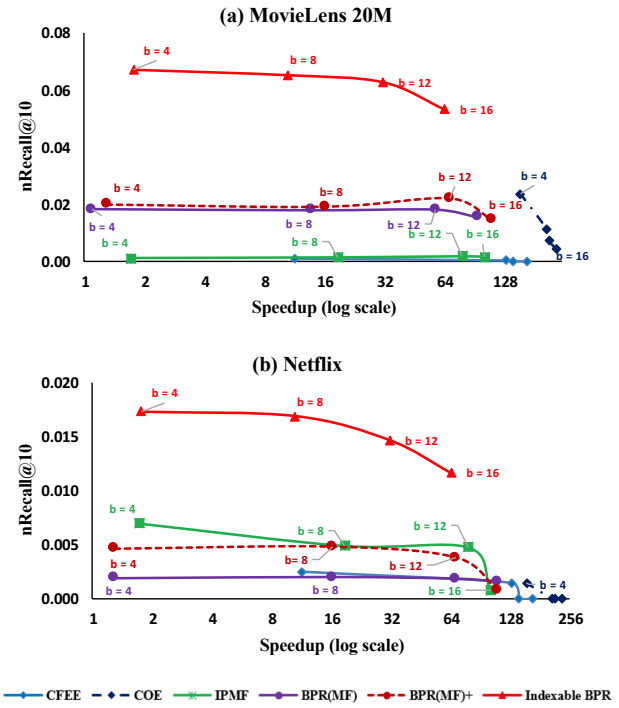


Figure 4: nRecall@10 vs. speedup with Hash table lookup strategy ($T = 10$ hash tables).

Each function assigns a bit for each vector. $\mathbf{h}$ will assign each user u a binary code $\mathbf{h}(x_u)$, and each item i a binary hashcode $\mathbf{h}(y_i)$, all of length b. Assuming that x_u prefers y_i to y_j, $\mathbf{h}$ is expected to produce binary hashcodes with a smaller Hamming distance $||\mathbf{h}(x_u) - \mathbf{h}(y_i)||_H$ than the Hamming distance $||\mathbf{h}(x_u) - \mathbf{h}(y_j)||_H$.

The most frequent indexing strategy for LSH is *hash table lookup*. We store item codes in hash tables, with items having the same code in the same bucket. Given a query (user) code, we can determine the corresponding bucket in constant time. We search for the top-k only among items in that bucket, reducing the number of items on which we need to perform exact similarity computations.

We use the LSH package developed by [1]. The LSH family for Indexable BPR for generating hashcodes is SRP-LSH, which is also used for IPMF following [7]. We apply it to BPR(MF) and BPR(MF)+, as [25], [19] claim it to be the more suitable family for transformed vectors. In turn, the LSH scheme for COE and CFEE is L2-LSH, since both use l_2 distance. In Section 5, we will elaborate with theoretical analysis and empirical evidence how more compatible representations tend to produce better results.

When using hash tables, one specifies the number of tables T and the code length b. We experiment with various T, and $T = 10$ returns the best performance (consistent with [7]). We also vary b and larger b is expected to lead to fewer items in each bucket.

Figure 3(a) shows the *nRecall@k* using *hash table lookup* with $T = 10$ tables and different values of code length $b = 8, 12, 16$ for *MovieLens20M*. Across the b's, the trends are similar. Indexable BPR has the highest *nRecall@k* values across all k. It outperforms BPR(MF)+ that conducts vector transformation as post-processing, which indicates that learning inherently indexable vectors is helpful. In turn, BPR(MF)+ outperforms BPR(MF), which indicates that the inner product kernel is not conducive for indexing. Interestingly, Indexable BPR also performs better than models that fit ratings (IPMF, CFEE), suggesting that learning from relative comparisons may be more suitable for top-k recommendation.

Figure 3(b) shows the results for *Netflix*. Again, Indexable BPR has the highest *nRecall@k* values across all k. The relative comparisons among the baselines are as before, except that IPMF now is more competitive, though still lower than Indexable BPR .

We also investigate the tradeoff between the speedup achieved and the accuracy of the top-k returned by the index. Fig. 4 shows the *nRecall@10*s and the speedup when varying the value of b. Given the same speedup, Indexable BPR can achieve significantly higher performance compared to the baselines. As b increases, the speedup increases and *nRecall@10* decreases. This is expected, as the longer the hashcodes, the smaller the set of items on which the system needs to perform similarity computation. This reflects the trade-off of speedup and approximation quality.

4.2 Top-k Recommendation with KD-Tree Index

Spatial trees refer to a family of methods that recursively partition the data space towards a balanced binary search tree, in which each node encompasses a subset of the data points [17]. For algorithms that model the user-item association by l_2 distance, spatial trees can be used to index the item vectors. Top-k recommendation is thus equivalent to finding kNN to the query. The tree will locate the

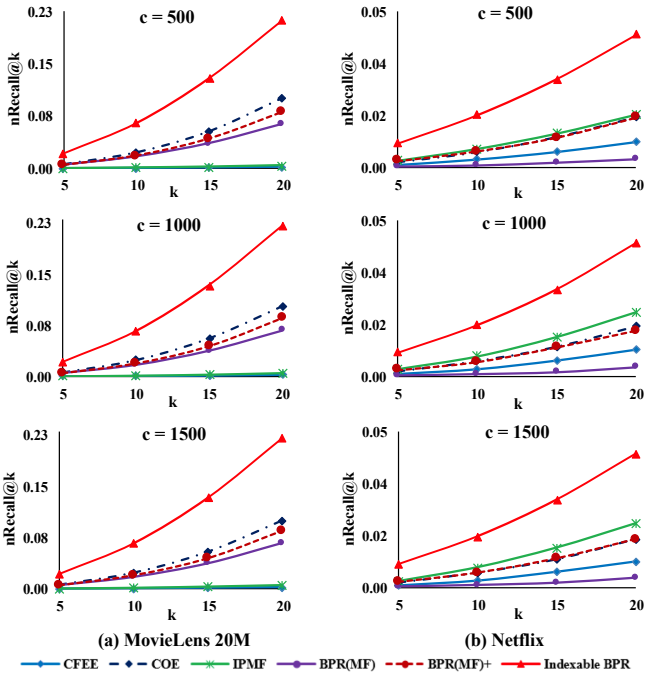


Figure 5: nRecall@k with KD-Tree indexing.

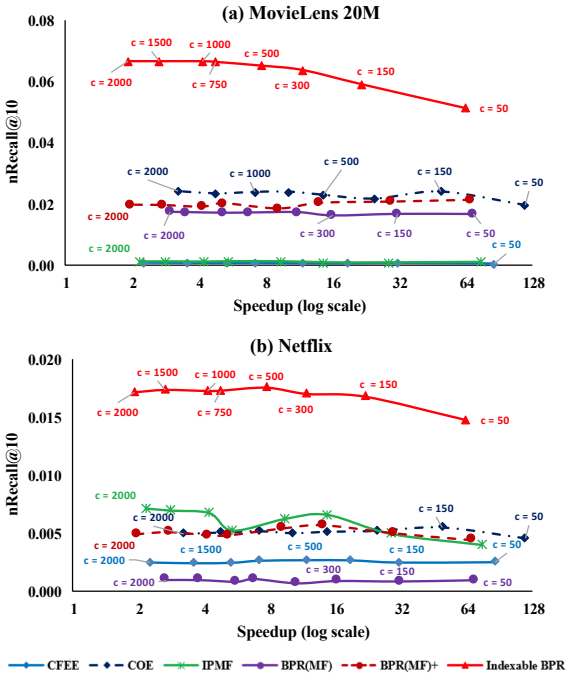


Figure 6: nRecall@10 vs. speedup with KD-tree indexing.

nodes that the query belongs to, and exact similarity computation is performed only on the points indexed by those nodes.

For INDEXABLE BPR , Algorithm 1 returns two sets of normalized vectors $\tilde{x}_u \forall u \in \mathcal{U}$ and $\tilde{y}_i \forall i \in \mathcal{I}$. We observe that:

$$||\tilde{x}_u - \tilde{y}_i|| < ||\tilde{x}_u - \tilde{y}_j|| \Leftrightarrow \tilde{x}_u^T \tilde{y}_i > \tilde{x}_u^T \tilde{y}_j \Leftrightarrow \theta_{\tilde{x}_u \tilde{y}_i} < \theta_{\tilde{x}_u \tilde{y}_j}, \quad (13)$$

i.e., the ranking of items according to l_2 distance on normalized vectors is compatible to that according to angular distance, implying INDEXABLE BPR 's output can support kNN using spatial tree.

In this paper, we consider a well-known tree structure, KD-tree. Approximate kNN retrieval can be achieved by restricting the searching time on the tree ([7]). The implementation of KD-tree in [18] controls this by c, the number of nodes to explore on the tree.

Figure 5 shows the *nRecall@k* with various $c \in \{500, 1000, 1500\}$. We also experimented with $c \in \{50, 150, 300, 750, 2000\}$ and get similar trends. INDEXABLE BPR consistently outperforms the baselines at all values of c. Notably, INDEXABLE BPR outperforms BPR(MF)+, which in turn outperforms BPR(MF), validating the point made earlier about native indexability. Figure 6 plots the accuracy in terms of *nRecall@10* vs. the retrieval efficiency in terms of speedup. As we increase c, a longer searching time on KD-tree is allowed, resulting in higher quality of the returned top-k. Here too, INDEXABLE BPR achieves higher accuracy at the same speedup, higher speedup at the same accuracy, as compared to the baselines.

4.3 Top-k Recommendation with Inverted Index

For recommendation retrieval, [4] presents an inverted index scheme, where every user or item is represented with a sparse vector derived from their respective dense real-valued latent vectors via a transformation. Given the user sparse vector as query, the inverted index will return items with at least one common non-zero element with the query as candidates. Exact similarity computation will be performed only on those candidates to find out the top-k.

Here, we describe very briefly the indexing scheme. For an extended treatment, please refer to [4]. The sparse representations for users and items are obtained from their dense latent vectors (learnt by the recommendation algorithm, e.g., INDEXABLE BPR) through a set of geometry-aware permutation maps Φ defined on a tessellated unit sphere. The tessellating vectors are generated from a base set $\mathcal{B}_d = \{-1, -\frac{d-1}{d}, \ldots, -\frac{1}{d}, 0, \frac{1}{d}, \ldots, \frac{d-1}{d}, 1\}$, characterized by a parameter d. The obtained sparse vectors have the sparsity patterns that are related to the angular closeness between the original latent vectors. The angular closeness between user vector x_u and item vector y_i is defined as $\mathrm{d_{ac}}(x_u, y_i) = 1 - \frac{x_u^T y_i}{||x_u||.||y_i||}$.

In the case of $||x_u|| = ||y_i|| = 1 \; \forall u \in \mathcal{U}, i \in \mathcal{I}$, we have $(\forall i \neq j \in \mathcal{I})$:

$$\mathrm{d_{ac}}(x_u, y_i) < \mathrm{d_{ac}}(x_u, y_j) \Leftrightarrow \underbrace{\frac{{x_u}^T y_i}{||x_u||.||y_i||}}_{\theta_{x_u y_i}} > \underbrace{\frac{{x_u}^T y_j}{||x_u||.||y_j||}}_{\theta_{x_u y_j}} \quad (14)$$

The item ranking according to $\mathrm{d_{ac}}$ is equivalent to that according to θ-angular distance. We hypothesize that INDEXABLE BPR based on angular distance would be compatible with this structure.

The parameter d can be managed to control the trade-off between the efficiency and the quality of approximation of kNN retrieval. Increasing the value of d leads to a higher number of discarded items using the inverted index, which leads to higher speedup of the top-k recommendation retrieval.

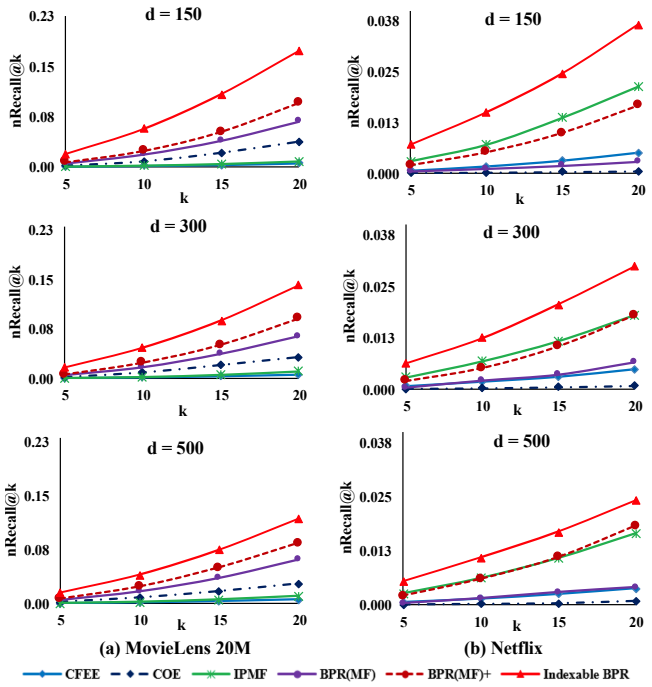


Figure 7: nRecall@k with inverted indexing.

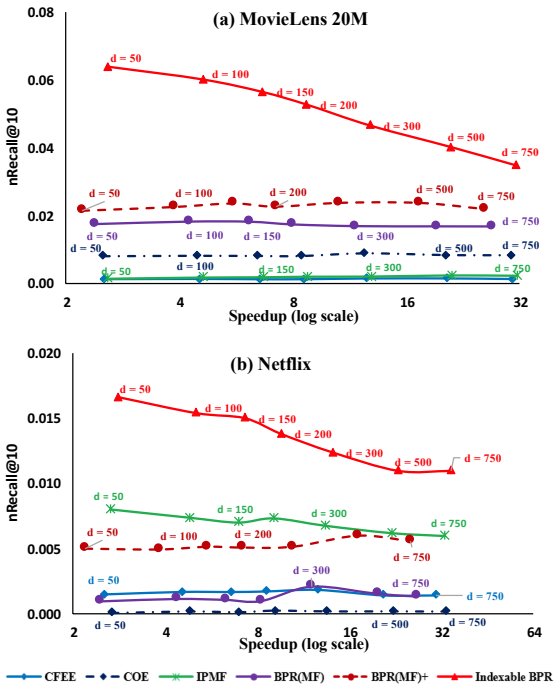


Figure 8: nRecall@10 vs. speedup with inverted indexing.

We run the experiments with different values of parameter d to explore the trade-off between speed and accuracy. Figure 7 presents the *nRecall@k* of the two datasets at $d \in \{150, 300, 500\}$. In all cases, Indexable BPR outperforms the baselines in terms of *nRecall@k*. This suggests that Indexable BPR produces a representation that has greater degree of compatibility in terms of angular closeness d_{ac} between users and their preferred items. As a result, the corresponding sparse vectors will have highly similar sparsity patterns, which enhances the quality of kNN using inverted indexing. Figure 8 shows the speedup using the inverted index as we vary the value of parameter d. We observe that the speedup increases as d increases. Indexable BPR shows superior performance as compared to other models, given the same speedup.

Overall, Indexable BPR works well on the indexing schemes. Effectively, we develop a model that work with multiple indices, and leave the choice of index structure to the respective application based on need. Our focus is on indexable recommendation algorithms. Here, several consistent observations emerge. Indexable BPR produces representations that are more amenable to indexing, as compared to baselines BPR(MF)+ and BPR(MF). This validates the aim of Indexable BPR in learning natively indexable vectors for users and items. It also outperforms models that fit ratings, as opposed to ordinal triples, for top-k recommendations.

5 ANALYSIS ON LSH-FRIENDLINESS OF INDEXABLE BPR

In an effort to further explain the outperformance by Indexable BPR when used with LSH, we analyze the compatibility betweeen recommendation algorithms and hashing functions. Since LSH is inherently an approximate method, the loss of information caused by random hash functions is inevitable. Informally, a representation is LSH-friendly if the loss after hashing is as minimal as possible. To achieve such small loss, a user's ranking of items based on the latent vectors should be preserved by the hashcodes.

Analysis. For x_u, y_i, y_j in $\mathbb{R}^D$, one can estimate the probability of the corresponding hashcodes to preserve the correct ordering between them. Let us consider the probability of Hamming distance $\Pr(||\mathbf{h}(x_u) - \mathbf{h}(y_i)||_H)$. Since the hash functions $h_1, h_2, \ldots, h_b$ are independent of one another, $||\mathbf{h}(x_u) - \mathbf{h}(y_i)||_H$ follows the binomial distribution with mean $bp_{x_u y_i}$ and variance $bp_{x_u y_i}(1 - p_{x_u y_i})$, where $p_{x_u y_i}$ is the probability of x_u and y_i having different hash values (this probability depends on the specific family of hash functions). Since binomial distribution can be approximated by a normal distribution with same mean and variance, and the difference between two normal distributions is another normal distribution, we have:

$$\Pr(||\mathbf{h}(x_u) - \mathbf{h}(y_j)||_H - ||\mathbf{h}(x_u) - \mathbf{h}(y_i)||_H > 0) \quad (15)$$
$$\sim \text{Normal}(bp_{x_u y_j} - bp_{x_u y_i}, bp_{x_u y_j}(1 - p_{x_u y_j}) + bp_{x_u y_i}(1 - p_{x_u y_i}))$$

Due to the shape of the normal distribution, Eq. 15 implies that a higher mean and smaller variance would lead to a higher probability of the hashcode of x_u is more similar to the hashcode of y_i than to the that of y_j. Therefore, for a fixed length b, if indeed u prefers i to j, we say that x_u, y_i, y_j is a more LSH-friendly representation for u, i, and j if the mean value $(p_{x_u y_j} - p_{x_u y_i})$ is higher and the variance $(p_{x_u y_j}(1 - p_{x_u y_j}) + p_{x_u y_i}(1 - p_{x_u y_i}))$ is smaller.

Hence, the mean and the variance in Eq. 15 could potentially reveal which representation is more LSH-friendly, i.e., preserves information better after hashing. For each user $u \in \mathcal{U}$, let τ_k^u be the set of items in the top-k by a method before hashing, and $\bar{\tau}_k^u$ be

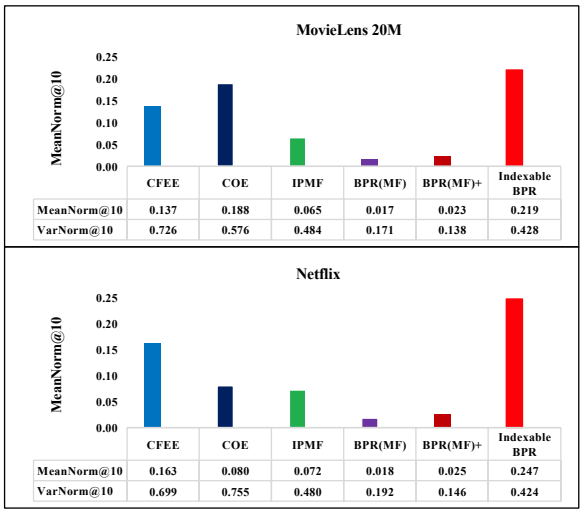


Figure 9: LSH friendly measurement at $D = 20$.

all the other items *not* returned by the models. We are interested in whether after hashing, the items in τ_k^u would be closer to the user than the items in $\bar{\tau}_k^u$. To account for this potential, we introduce two measures: *MeanNorm@k* and *VarNorm@k*.

$$\text{MeanNorm@k} = \frac{1}{|\mathcal{U}|}\sum_{i\in\tau_k^u}\sum_{j\in\bar{\tau}_k^u}\frac{(p_{x_u y_j} - p_{x_u y_i})}{|\tau_k^u|.|\bar{\tau}_k^u|}$$

$$\text{VarNorm@k} = \frac{1}{|\mathcal{U}|}\sum_{i\in\tau_k^u}\sum_{j\in\bar{\tau}_k^u}\frac{p_{x_u y_j}(1-p_{x_u y_j}) + p_{x_u y_i}(1-p_{x_u y_i})}{|\tau_k^u|.|\bar{\tau}_k^u|}$$

To achieve LSH-friendly representation, *MeanNorm@k* should be high and *VarNorm@k* should be low. Fig. 9 shows the bar charts displaying values of those metrics. From Fig. 9, Indexable BPR shows higher mean values *MeanNorm@10* (i.e., $k = 10$) at $D = 20$ (we observe the same results with other values of D and k). Though BPR(MF) and BPR(MF)+ have smaller variance, their mean values are among the lowest. This result gives us a hint that Indexable BPR can preserve information after hashing more effectively.

Compatible Hash Function. There is an explanation for the superior numbers of Indexable BPR in Fig. 9. Specifically, the probability $p_{x_u y_i}$ depends on the LSH family. In particular, signed random projections [5, 9] or SRP-LSH is meant for angular similarity. The angular similarity between x, y is defined as $sim_\angle(x, y) = 1 - \cos^{-1}(\frac{x^T y}{||x||.||y||})/\pi$. The parameter a is a random vector chosen with each component from i.i.d normal. The hash function is defined as $h_a^{\text{srp}}(x) = \text{sign}(a^T x)$ and the probability of x, y having different hash values is:

$$p_{xy} = \Pr(h_a^{\text{srp}}(x) \neq h_a^{\text{srp}}(y)) = \cos^{-1}(\frac{x^T y}{||x||.||y||})/\pi = \frac{\theta_{xy}}{\pi}, \quad (16)$$

For Indexable BPR , as shown in Eq. 8, for each observation "*u prefers i to j*", we would like to maximize the difference $\theta_{x_u y_j} - \theta_{x_u y_i}$. From Eq. 16, we observe that the probability $p_{x_u y_i}$ is a *linear* function of the angular distance $\theta_{x_u y_i}$. Thus, we can infer that Indexable BPR 's objective corresponds to maximizing $p_{x_u y_j} - p_{x_u y_i}$.

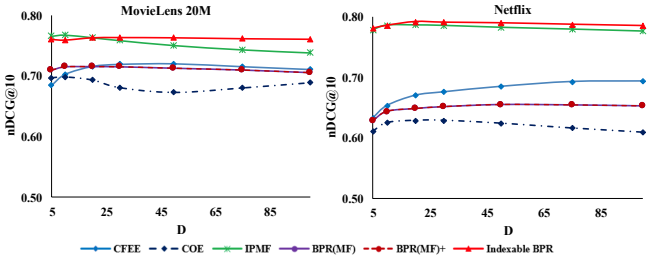


Figure 10: nDCG@10 at $D \in \{5, 10, 20, 30, 50, 75, 100\}$.

According to Eq. 15, this increases the probability that the Hamming distance between u and i is smaller than that between u and j. In other words, the hashcodes are likely to preserve the ranking order. This alignment between the objective of Indexable BPR and the structural property of SRP-LSH implies that Indexable BPR is more LSH-friendly, which helps the model minimize information loss, and show better post-indexing performance.

Also, the appropriate LSH family for methods based on l_2 distance, which includes COE, is L2-LSH [6]. However, there is a question as to how compatible the objective of COE is with the hash functions. The hash function of L2-LSH is defined as follows:

$$h_{a,b}^{L_2}(x) = \lfloor\frac{a^T x + b}{r}\rfloor; \quad (17)$$

where r - the window size, a - random vector with each component from i.i.d normal and a scalar $b \sim \text{Uni}(0, r)$. The probability of two points x, y having different hash values under L2-LSH function is:

$$\begin{aligned} F_r^{L_2}(d_{xy}) &= \Pr(h_{a,b}^{L_2}(x) \neq h_{a,b}^{L_2}(y)) \\ &= 2\phi(-\frac{r}{d_{xy}}) + \frac{1}{\sqrt{(2\pi)}(r/d_{xy})}(1 - \exp(-(\frac{r}{d_{xy}})^2/2)); \end{aligned} \quad (18)$$

where $\phi(x)$ is cumulative probability function of normal distribution and $d_{xy} = ||x - y||$ is the l_2 distance between x, y. From Eq. 18, we see that $F_r^{L_2}(d_{xy})$ is a *nonlinear* monotonically increasing function of d_{xy}. COE's objective to maximize $d_{x_u y_j} - d_{x_u y_i}$ does not directly maximize the corresponding mean value of the normal distribution (see Eq.15), i.e., $F_r^{L2}(d_{x_u y_j}) - F_r^{L2}(d_{x_u y_i})$, since $F_r^{L_2}(d_{x_u y_j})$ is *not a linear* function of l_2 distance $d_{x_u y_j}$. Our hypothesis is that though both rely on ordinal triples, COE may not be as compatible with LSH as Indexable BPR .

Empirical Evidence. For each user u, we rank the items that u has rated in the test set, and measure how closely the ranked list is to the ordering by ground-truth ratings. As metric, we turn to the well-established metric for ranking *nDCG@k*, where k is the cut-off point for the ranked list. Its definition can be found in [26].

Fig. 10 shows the *nDCG@10* values for *MovieLens 20M* and *Netflix* respectively at various dimensionality of the latent vectors D. We observe that, Indexable BPR is among the best, with the most competitive baseline being IPMF (which fits ratings). More important is whether the models will still perform well when used with index structures. As similar trends are observed with other values of D, subsequently we show results based on $D = 20$.

Here, the objective is to investigate the effectiveness of the LSH hashcodes in preserving the ranking among the *rated* items in the test set. We use *Hamming ranking*, repeating the same experiment

Table 2: *Absolute nDCG@10* and *Relative nDCG@10* of all models as the length of LSH codes (b) varies.

	MovieLens 20M						Netflix					
	Absolute nDCG@10			Relative nDCG@10			Absolute nDCG@10			Relative nDCG@10		
b	**8**	**12**	**16**	**8**	**12**	**16**	**8**	**12**	**16**	**8**	**12**	**16**
CFEE	0.582	0.582	0.585	0.805	0.806	0.809	0.559	0.561	0.562	0.834	0.836	0.838
COE	0.605	0.609	0.608	0.886	0.891	0.890	0.570	0.565	0.575	0.906	0.898	0.914
IPMF	0.702	0.728	0.704	0.920	0.955	0.923	0.705	0.737	0.747	0.896	0.936	0.949
BPR(MF)	0.599	0.603	0.605	0.831	0.837	0.840	0.560	0.551	0.553	0.863	0.849	0.853
BPR(MF)+	0.603	0.604	0.606	0.837	0.840	0.841	0.569	0.569	0.566	0.877	0.877	0.873
Indexable BPR	**0.743**	**0.745**	**0.754**	**0.977**	**0.980**	**0.991**	**0.732**	**0.761**	**0.756**	**0.924**	**0.960**	**0.954**

in Fig.10, but using Hamming distances over hashcodes. This is to investigate how well Indexable BPR preserves the ranking compared to the baselines. As hashing relies on random hash functions, we average results over 10 different sets of functions.

Table 2 shows the performances of all models. The two metrics are: *Absolute nDCG@10* is the nDCG@10 of LSH hashcodes, and *Relative nDCG@10* is the relative ratio between the Absolute nDCG@10 and that of original real-valued latent vectors. Indexable BPR consistently shows better *Absolute nDCG@10* values than the baselines when using LSH indexing. This implies that Indexable BPR coupled with SRP-LSH produces more compact and informative hashcodes. Also, the *Relative nDCG@10* of Indexable BPR are close to 1 and higher than those of the baselines. These observations validate our hypotheses that not only is Indexable BPR competitively effective pre-indexing, but it is also more LSH-friendly, resulting in less loss in the ranking accuracy post-indexing.

6 CONCLUSION

We propose a probabilistic method for modeling user preferences based on ordinal triples, which is geared towards top-k recommendation via approximate kNN search using indexing. The proposed model Indexable BPR produces an indexing-friendly representation, which results in significant speedups in top-k retrieval, while still maintaining high accuracy due to its compatibility with indexing structures such as LSH, spatial tree, and inverted index. As future work, a potential direction is to go beyond achieving representations more compatible with existing indexing schemes, to designing novel data structures or indexing schemes that would better support efficient and accurate recommendation retrieval.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its NRF Fellowship Programme (Award No. NRF-NRFF2016-07).

REFERENCES

[1] Mohamed Aly, Mario Munich, and Pietro Perona. 2011. Indexing in large scale image collections: Scaling properties and benchmark. In *IEEE Workshop on Applications of Computer Vision (WACV)*. 418–425.
[2] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys*. ACM, 257–264.
[3] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
[4] Avradeep Bhowmik, Nathan Liu, Erheng Zhong, Badri Narayan Bhaskar, and Suju Rajan. 2016. Geometry Aware Mappings for High Dimensional Sparse Factors. In *AISTATS*.
[5] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. ACM, 380–388.
[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCOG*. ACM, 253–262.
[7] Marco Fraccaro, Ulrich Paquet, and Ole Winther. 2016. Indexable Probabilistic Matrix Factorization for Maximum Inner Product Search. In *AAAI*.
[8] Ruining He and Julian McAuley. 2016. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In *AAAI*.
[9] Jianqiu Ji, Jianmin Li, Shuicheng Yan, Bo Zhang, and Qi Tian. 2012. Super-bit locality-sensitive hashing. In *NIPS*. 108–116.
[10] Mohammad Khoshneshin and W Nick Street. 2010. Collaborative filtering via euclidean embedding. In *RecSys*. ACM, 87–94.
[11] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*. ACM, 535–544.
[12] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).
[13] Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2012. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *WSDM*. 173–182.
[14] J. B. Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29, 1 (1964).
[15] Dung D Le and Hady W Lauw. 2016. Euclidean Co-Embedding of Ordinal Data for Multi-Type Visualization. In *SDM*. SIAM, 396–404.
[16] Lukas Lerche and Dietmar Jannach. 2014. Using graded implicit feedback for bayesian personalized ranking. In *RecSys*. 353–356.
[17] Brian McFee and Gert R. G. Lanckriet. 2011. Large-scale music similarity search with spatial trees. In *ISMIR*.
[18] Marius Muja and David G. Lowe. 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 331–340.
[19] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *ICML*.
[20] Weike Pan and Li Chen. 2013. GBPR: Group Preference Based Bayesian Personalized Ranking for One-Class Collaborative Filtering.. In *IJCAI*, Vol. 13. 2691–2697.
[21] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 931–939.
[22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. AUAI Press, 452–461.
[23] Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *ICML*. ACM, 880–887.
[24] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.
[25] Anshumali Shrivastava and Ping Li. 2015. Improved Asymmetric Locality Sensitive Hashing (ALSH) for Maximum Inner Product Search (MIPS). In *UAI*.
[26] Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alexander J. Smola. 2007. COFI RANK - Maximum Margin Matrix Factorization for Collaborative Ranking. In *NIPS*.
[27] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proc. of SIGIR*, Vol. 16.
[28] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *SIGIR*. ACM, 183–192.
[29] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *KDD*. ACM, 498–506.